tmssoftware.com

TMS Data Modeler Documentation

Table of Contents

Chapter

I

Introduction

1   Introduction

TMS Data Modeler is a visual database design system that integrates database design, modeling, creation and maintenance into a single environment.

The application combines professional features and a clear and simple user interface to offer the most efficient way to handle your database. It is developed and optimized for the most used database management systems (Microsoft SQL Server, Firebird, Oracle etc.) and generates clean scripts to create or update your projects.

TMS Data Modeler product page: http://www.tmssoftware.com/site/tmsdm.asp
TMS Software site: http://www.tmssoftware.com

Documentation Topics

- **Introduction**
  - Overview
  - What's New
  - Copyright Notice
  - Supported Databases
  - Concepts
- The User Interface
  - UI Overview
  - Ribbons
  - Project Explorer
  - Workspace
  - Messages Window
- Basic Operations
  - Creating a New Project
  - Creating Objects in Database Model
  - Comparing projects
  - Generating Database Creation Scripts
  - Project Validation
  - General Project Settings
  - Version Control
  - Reverse Engineering
  - Convert Project to Different Database
- Editors
  - Table Editor
  - Domains
  - Diagrams
  - Relationship Editor
  - Procedures and Functions
  - Generators/Sequences
  - Views
- TMS Aurelius Export
  - Overview
  - Export Dialog

1.1   Overview

TMS Data Modeler is a visual database design system that integrates database design, modeling, creation and maintenance into a single environment.

The application combines professional features and a clear and simple user interface to offer the most efficient way to handle your database. It is developed and optimized for the most used database management systems (Microsoft SQL Server, Firebird, Oracle etc.) and generates clean scripts to create or update your projects.

Features and benefits

- Generic tool that allows comprehensive modeling for databases.
- High performance and easy-to-use, intuitive interface.
- Supports the main database management systems (DBMS) such as SQL Server and Firebird.
- Stores the database structure in its own file format (DGP).
- Generates scripts to create complete databases or update existing databases.
- Performs reverse engineering to recover the structure of an existing database, allowing better visualization and adjustments.
- Controls database versions through history management: you can follow up changes and generate update scripts.
- Creates entity-relationship diagrams.
- Converts between different DBMS.
- Offers resources for integration with external systems (by reading the DGP file or by execution through command line).
- Checks consistency, validating all items on the data dictionary.
- Error report allows updates and improvements.
- Creates Delphi source code with exported classes for TMS Aurelius Object-Relational Mapping Framework.

1.2   What's New

Version 3.3.2 (Jan-2019)
- **New: PostgreSQL 11 support**, avoiding the error "p.proisagg does not exist. perhaps you meant to reference the column p.prolang"

Version 3.3.1 (Jan-2019)
- **Improved:** Arguments for OnClassGenerated event now includes references to Table and Sequence attribute. Used in the example Adding Schema Name to Table Attribute.
- **Fixed:** Performing project checking in ElevateDB projects now reports wrong field size for char/binary fields with size higher than 1024.

Version 3.3 (Dec-2018)

- **New: "Scripting" button in Tools tab in ribbon, opens a full scripting IDE for low-level and advanced manipulation of the existing data dictionary.**

Version 3.2.6 (Nov-2018)
- **Improved:** List of tables in the Generate Script dialog is now sorted.
- **Fixed**: Fields in primary keys were being nullable when using nullable domains. It broke backward compatibility, fields in primary keys should always be not null.
- **Fixed:** PostgreSQL database importer was not correctly retrieving foreign keys with same name in different tables.

Version 3.2.5 (Nov-2018)
- **Improved:** There is now a check box "Specific" for the "Not null" field. When the field has a domain associated to it, this new check box allows explicitly control if the NOT NULL flag should come from domain or you would want to ignore the domain setting and use a specific value for the field.

Version 3.2.4 (Nov-2018)
- **Fixed:** Foreign keys with multiple fields (relationships with composite keys) not being correctly imported from Oracle databases.

Version 3.2.3 (Oct-2018)
- **Fixed:** Null/NotNull checkbox was disabled for fields associated with logical domains.
- **Fixed:** Project window maximizing automatically in some situations.

Version 3.2.2 (Oct-2018)
- **Fixed:** SQL Server ALTER TABLE ADD COLUMN statement was not including constraint name for default values.

Version 3.2.1 (Sep-2018)
- **Fixed:** Firebird behavior with domains: size, not null and constraint was not being retrieved from domain.

Version 3.2 (Sep-2018)

- **New:** Support for Interbase 2017.

- **Fixed:** Error when importing MySQL 8 databases (Table 'mysql.proc' doesn't exist)

- **Fixed**: Firebird/Interbase connection was not working in some situations.

Version 3.1.2 (Jul-2018)

- **Fixed:** Data Modeler version information in VCL Subscription Manager showing incorrect after automatic update.

· Version 3.1.1 (Jul-2018)

- **Fixed:** SQLite database import was wrongly considering SQLite "TEXT" datatype as a (char) blob. Now it's considered as regular string (VARCHAR).

- **Fixed:** Importing SQLite tables with field names starting with number was raising error.

Version 3.1 (May-2018)

- **New: Database metadata objects available in customization script events**. One usage example is adding ForeignKey attribute to associations to force Aurelius create FK using the existing database foreign key (relationship) name.

Version 3.0.3 (May-2018)

- **Fixed:** Connecting to MS SQL Server LocalDB was raising an error "SQL Server does not exist or access is denied"

· Version 3.0.2 (Mar-2018)

- **Fixed:** Exporting descriptions with single quotes to Aurelius classes was generating invalid Pascal code. The single quotes are now duplicated to form valid Pascal strings.

· Version 3.0.1 (Nov-2017)

- **New:** OnUnitGenerated event in TMS Aurelius Export customization script.

- **Improved:** Table list in Mappings Tab of TMS Aurelius Export Dialog is now sorted in alphabetical order.

- **Improved:** Added OID type for PostgreSQL databases.

- **Fixed:** When adding a property in a customization script, if property type was empty the generated code was adding a colon after property name: "*property Name: ;*"

· Version 3.0 (Oct-2017)

- **New: Customization scripts in TMS Aurelius export.** You can now customize the source code generated by TMS Aurelius export feature using a customization script that be be written in the Script Tab of export dialog. This adds lots of flexibility for you to add specific attributes, fine tune the generated code, and even add new fields, properties and methods to the classes.

- **New: Option to export TMS Aurelius classes to several different units.** You now have the option to specify the name of the unit in which a class will be created, allowing you to better organize your classes into several units. If there is a cyclical reference between units, Data Modeler will warn you in advance giving you the chance to fix the issues.

- **New: Model names in TMS Aurelius export.** Now you have an option to specify the models where an export class will belong to. This will create [Model] attributes in the class. The model names can also be automatically set based on the existing diagrams in the project (each diagram is considered to be a model).

- **New: Source code preview in Aurelius export.** In the Aurelius export dialog, you can new preview the source code of the unit(s) that will be created. It makes it easier to change settings and see how they affect final source code without needing to generate the actual files in directory.

- **New: Modern User Interface.** Ribbon user interface was updated to a new color theme and some ribbon elements were updated. The look and feel is now closer to Office 2017 style and provides a more modern and refreshing feeling. Splash screen has also been updated.

- **Improved**: Documentation has received a significant review. Contains now more detailed and organized topics, especially on TMS Aurelius export. The look and feel has also been improved in both CHM and PDF formats. A new online help is now available for browsing.

- **Fixed**: Checkboxes for defining specific default and check constraint were not being enabled correctly.

- **Fixed**: "Access Violation" error when clicking Exit or double click menu button.

Version 2.8.1 (Dec-2016)
- Fixed: SQL Server primary key constraint ignoring field order (asc/desc)

Version 2.8 (Dec-2016)
- Improved: Installers for trial and registered versions now signed to minimize Windows warnings and false antivirus alerts
- Improved: Register entities option checked by default (Aurelius Export)
- Improved: AllButRemove is default option for association cascade type (Aurelius Export)
- Fixed: Data Modeler not appearing as "installed" in TMS Subscription Manager

Version 2.7 (Nov-2016)

- New: Support for Firebird 3 database

Version 2.6.1 (Oct-2016)
- Fixed: When generating Aurelius classes, constructor implementation is now calling inherited constructor.

Version 2.6 (Jun-2016)
- Improved: PostgreSQL support for data types JSON and JSONB

Version 2.5.1 (Nov-2015)
- Fixed: Firebird database importer was failing with DOMAIN_NULL_FLAG error

Version 2.5 (Jan-2016)
- New: Option to explicitly define the type of the field/property when exporting model to Aurelius classes

Version 2.4.10 (Nov-2015)
- Fixed: Firebird stored procedures now being created with header only to avoid errors with procedure dependencies
- Fixed: Firebird CREATE DOMAIN statement incorrectly generated when using CHECK constraints
- Fixed: Visual glitch when displaying comments in stored procedure editor

Version 2.4.9 (Nov-2015)
- Improved: Firebird configuration allows providing the CHARSET for the connection. Default is UTF8.

Version 2.4.8 (Aug-2015)
- Fixed: PostgreSQL TIMESTAMP fields were wrongly being exported to Aurelius as Double properties, instead of TDateTime.

Version 2.4.7 (Aug-2015)
- Improved: When exporting memo fields to Aurelius classes, a [DBTypeMemo] attribute is added for better Aurelius handling of memo types

Version 2.4.6 (Jul-2015)
- Fixed: Text block not displaying text in workflow instance if text was edited directly in the block (not using editor)
- New: MySQL script for workflow tables

Version 2.4.5 (May-2015)
- Improved: Aurelius Export Tool now supports composite keys in inheritance relationships

Version 2.4.4 (Apr-2015)
- Improved: CascadeTypeAll included as a cascade option for Associations when exporting to Aurelius

Version 2.4.3 (Apr-2015)
- Fixed: Import of SQLite databases with field names using different characters like "&" or "."
- Fixed: Import ot SQLite databases with default value using identifiers

Version 2.4.2 (Nov-2014)

- New: Aurelius export includes an option register all entities using RegisterEntity procedure (to avoid linker to remove unused classes)
- New: Firebird connection dialog includes "Vendor Lib" option allowing specify driver (for example, gds32.dll as alternative to fbclient.dll)

Version 2.4.1 (Oct-2014)
- Fixed: Firebird import now retrieves descriptions using correct character set

Version 2.4 (Sep-2014)
- New: PostgreSQL 9 support
- Fixed: Indexes marked as "Unique Key" not being generate as exclusive indexes (Indexes marked as "Exclusive" were working correctly) (2.3.3 - Aug/2014)
- Improved: Aurelius classes exported sorted by name (2.3.2 - Jul/2014)
- Fixed: Aurelius classes exported in wrong order in some complex inheritance hierarchy (2.3.2 - Jul/2014)
- Fixed: wrong Inheritance attribute value when exporting to Aurelius classes (2.3.1 - Jun/2014)

Version 2.3 (Feb-2014)
- New: Advanced connection options for ElevateDB databases
- New: Zoom tab in ribbon makes it easier to define diagram zoom, zoom to 100% or zoom to fit all
- New: Export to Aurelius allows defining cascade types for associations
- Improved: Trigger editor now expands to the whole window area making it easier to write trigger code
- Fixed: Several errors reported by automatic error report (mostly Access Violations in some specific situations)

Version 2.2 (May-2013)
- New: Diagram navigator allows overview of entire diagram and easy navigation/zooming using navigation cursor
- Improved: Automatic selection of last used connection when importing database structure
- Fixed: Timestamp fields correctly imported from Oracle databases
- Fixed: MySQL reverse engineering was not correctly importing multi-column foreign keys
- Fixed: Oracle reverse engineering was retrieving wrong size for NVarchar2 fields
- Fixed: ElevateDB reverse engineering was retriving foreign keys incorrectly when composed of more than one field
- Fixed: Import SQLite tables with /*..*/ comments in DDL command
- Fixed: Rare AV when generating Aurelius classes unit

Version 2.1 (Feb-2013)
- New: Support for ElevateDB Unicode Server (in addition to existing support to ANSI servers)
- Improved: ElevateDB server name configuration can accept both IP address and host name
- Improved: Support for Windows 8 (2.0.2)
- Improved: Connection settings in NexusDB reverse engineering now displays available servers automatically (2.0.1)
- Fixed: Incorrect NexusDB reverse engineering when connecting to 3.10 and 3.11 databases using internal server (2.0.1)

Version 2.0 (Feb-2013)
- New: TMS Aurelius Export: option to define Sequence attribute for each entity class
- New: "Duplicate table" feature available in diagram toolbar button and popup menu in table list
- New: Find toolbar button makes easy to find a table in diagram by name
- New: Licensing tool which provides a much better, easier way to register Data Modeler
- Improved: Better error message when exporting to Aurelius source code fails due to file system error
- Improved: Id properties exported to TMS Aurelius are now read-write instead of read-only.
- Note: 1.9.1 version was released at the same time as 2.0 version, including several other improvements and all bugs fixed since version 1.9.

Version 1.9.1 (Feb-2013)
- New: option to display relationship description/caption in diagram
- New: support for TGuid properties and Guid/Uuid generators in Aurelius export
- Improved: SQLite reverse engineering sets a default size for varchar and decimal data types, when not specified
- Improved: better detection of Firebird/Interbase database server version
- Improved: Clearer message when updating Data Modeler to avoid confusion with Windows/Data Modeler restart
- Improved: Firebird now differentiates unique indexes from unique constraints
- Improved: Better error message when trying to use unsupported versions of MySQL server
- Fixed: issues with SQLite reverse engineering (incorrect SQL parsing)
- Fixed: Access Violation in Aurelius source code generator (automatic report)
- Fixed: When adding new trigger, syntax was incorrect for Firebird database
- Fixed: Uncommon error when importing Firebird databases (RDB$DESCRIPTION column unknown)
- Fixed: Aurelius source generator now generates Int64 properties for database fields of type BIGINT and equivalents
- Fixed: Aurelius exporting foreign key fields that were part of primary key was incorrect when the property name was being changed manually by the user
- Fixed: Multi-line descriptions now handled correctly when exporting to Aurelius classes
- Fixed: Script viewer window now being presented correctly in multi-monitor desktops
- Fixed: Wrong Alter Table statement in Firebird databases (add non-nullable column)
- Fixed: Importing SQLite databases with spaces in table Name
- Fixed: All reproduceable automatic error reports up to release date

Version 1.9 (May-2012)
- New: Support for SQLite 3.7 databases
- New: Option for specifying dynamic properties in class when exporting to TMS Aurelius
- New: Aurelius export now has an option to singularize table names (convert plural table names into singular class names)
- Improved: Project check now checks if fields in a relationship have the same size, besides being of same type
- Improved: Hint for recent files in menu makes it easier to see full file name

- Improved: Better saving of form position and state (maximized/minimized)
- Fixed: issue with ribbon menu when windows is set to use big-sized fonts
- Fixed: Parent and child fields in relationship could have different data types in some cases (varchar with different sizes)
- Fixed: duplicated drop constraints in sql server when both default value and constraint default name were changed
- Fixed: internal issue with having duplicated id's for objects
- Fixed: MySQL drop index and drop foreign key statements had incorrect syntax
- Fixed: double data types were being imported incorrectly in MySQL
- Fixed: 0000748 [SQL Script] Check creation order of objects in SQL Script (firebird)
- Fixed: 0001084 [Comparer] Firebird-Merge Function-Stored Procedure-No input parameters generate a header with an integer type but no variable name
- Fixed: 0001083 [Comparer] Firebird - Using Merge Function - Stored Procedure scripts Terminator missing
- Fixed: Automatic reports 0001187, 0001162, 0001238

Version 1.8 (Jan-2012)
- 0001145: New: Export classes to TMS Aurelius framework.
- 0001158: [Interface] New window to find a table in diagram using Ctrl+F.
- 0001152: [Interface] New: "Find field..." option to search for fields in field grid in table editor. Can also use Ctrl+Shift+F.
- 0001151: [Interface] New: "Find in Diagram" popup menu option in table list makes it easy to find a table in the diagram.
- 0001149: [Reverse engineering] New: Import field descriptions from SQL Server.
- 0001159: [Interface] Automatically selects a child field when creating a new relationship if same field name and type as parent.
- 0001146: [Interface] New: New diagram popup menu options "All Keys and Indexes" in diagram context menu.
- 0001156: [Interface] Diagram popup menu option to create relationship using selected table
- 0001150: [Interface] Fixed: Dragging a table to a scrolled or zoomed diagram puts the table at wrong position.
- 0001148: [Interface] Improved: Deleting a field causes grid flickering and scrolling.
- 0001147: [Interface] Improved: Put close button on tabs.
- Other bug fixes. Tracker items: 0001142, 0001044, 0001087, 0001119, 0001109, 0001160, 0001161

Version 1.7.1 (Feb-2011)
- 0000902: Object comments imported from Firebird databases in reverse engineering.
- 0000903: Script generation and comparison for object comments (Firebird).
- Other minor bug fixes and small improvements. Tracker items: 0000505, 0000506, 0001031, 0001033, 0001034, 0001042.

Version 1.7 (Dec-2010)
- 0000871: Included support for ElevateDB.
- Bug fixes. Tracker items: 0000945, 0000956.

Version 1.6 (Nov-2010)
- 0000923: Included support for Microsoft SQL Azure.

- 0000954: [Reverse engineering] Database server version check before perform reverse engineering.
- 0000933: Fixed: [Interface] Window state not restored when main form is maximized.
- Other minor bug fixes. Tracker items: 0000886, 0000895, 0000910.

Version 1.5.1 (Oct-2010)
- 0000392: [Interface] Persistence of size and position of windows.
- 0000402: [Interface] New keyboard shortcuts and context options on interface.
- 0000414: [Project checking] Option to update the version files after saving a project as a different name.
- 0000489: [Interface] New option "Select all" in diagram context menu.
- 0000680: [Interface] New option "Close all except this" in tabs context menu.
- 0000681: [Interface] New option to set text colour on notes.
- 0000852: [Interface] Improved resizing of controls in table editor.
- 0000857: [Reverse engineering] Issues in reverse engineering from MySQL databases.
- 0000828: [SQL Script] Issue with expression of check constraints retrieved from Firebird databases.
- 0000823: [Reverse engineering] Issue importing UTF8 defined fields from Firebird databases.
- 0000746: [SQL Script] Fixed use of NULL/NOT NULL constraints in SQL script for computed columns (SQL Server).
- 0000602: [Interface] Fixed not null option in child field created automatically for a non-identifying relationship.
- 0000601: [Interface] Issues with suggestion of fields in Child Table when creating a relationship.
- 0000569: [Interface] Issues with editing of check constraint fields.
- 0000482: [Interface] Issues with editing of domain fields.
- Other bug fixes and small improvements. Tracker items: 0000243, 0000559, 0000563, 0000649, 0000656, 0000729, 0000735, 0000785, 0000792, 0000798, 0000802, 0000807, 0000817, 0000841, 0000854.

Version 1.5 (Aug-2010)
- 0000694: [SQL Script] Support for MySQL 5.1 databases.
- 0000734: [Interface] Option for inserting any object in popup menu of the explorer.

Version 1.4.0.1 (Mar-2010)
- 0000702: [Internal] Error loading project file (reading TGDAODiagram.DiagramString).

Version 1.4 (Mar-2010)
- 0000693: [SQL Script] Support to Oracle 10g databases.
- 0000650: [Reverse engineering] Allow "Server:Port" syntax in NexusDB server name connection.
- Other bug fixes and small improvements. Tracker items: 0000696, 0000688, 0000659, 0000658, 0000664, 0000667, 0000668, 0000686, 0000687.

Version 1.3.1 (Aug-2009)
- 0000648: [Interface] Use default tray icon balloon hint for update notification. Allow update notification for trial versions.

- 0000647: [Interface] Allow setting font name, size and shape color for diagram blocks.
- 0000645: [SQL Script] Invalid column name in alter script.
- Other bug fixes and small improvements. Tracker items: 0000644, 0000604.

Version 1.3 (Jul-2009)
- 0000622: [SQL Script] Support for NexusDB database.
- Other bug fixes and small improvements. Tracker items: 0000626, 0000636, 0000623, 0000552, 0000617, 0000608.

Version 1.2 (Jun-2009)
- 0000613: [SQL Script] Support for SQL Server 2008.
- 0000615: [Interface] Allow setting paper size, orientation and measurement units for diagram printing.
- 0000614: [Reverse engineering] Char size and domain issues with SQL Server 2000 reverse engineering.
- Other bug fixes and small improvements. Tracker items: 0000609, 0000600, 0000606, 0000605, 0000597, 0000598, 0000534, 0000492.

Version 1.1 (May-2009)
- 0000405: [Internal] Support for Absolute Database.
- 0000589: [Interface] Diagram Print Preview feature included.
- Other bug fixes and small improvements. Tracker items: 0000577, 0000580.

Version 1.0 (Apr-2009)
- First official 1.0 release.

1.3   Copyright Notice

Licensing Information

Trial version of this product is free for use in non-commercial applications, that is any software that is not being sold in one or another way or that does not generate income in any way by the use/distribution of the application.

For registered version of this product, three types of licenses apply:

Single Developer License

Main Copyright

1.4   Supported Databases

The following database systems are supported:

- MySQL
- Oracle
- Microsoft SQL Server
- PostgreSQL
- Firebird
- SQLite
- Microsoft SQL Azure
- NexusDB
- ElevateDB
- Absolute Database

1.5   Concepts

All database objects are listed under the appropriate data dictionary object button on the Project explorer. This chapter lists and gives a brief explanation about the main objects on a project.

Tables
Tables are the basic elements of a project. They can be imported from an existing database or be easily created in a TMS Data Modeler project and later implemented in the database.

All tables in the project are listed under *Tables* on the Project eEplorer, and can also be visualized in a diagram.

Double-clicking on a table from the list opens the *Table editor*, where you can view and update all of its properties including fields and indexes.

Using TMS Data Modeler you may also create Constraints, which are restrictions/rules applied to a table, and Triggers, which are procedures executed every time an update / insert / delete is executed in a given table.

Relationships
Relationships are connections between tables. A relationship is shown in diagrams as a line between the child and parent table, or around a single table in the case of a self-relationship. TMS Data Modeler uses a key from the parent table to manipulate relationships, therefore previous creation of a key, such as a primary key or other exclusive index, is mandatory. By selecting a field on the parent key you are able to select all compatible fields on the child table.

The supported relationship types are:

Non-identifying relationship: it's the most usual type of relationship. It represents a weak connection, a relationship between parent and child table that does not involve a key field, such as the relationship between tables "Products" and "Categories", for example.

Identifying relationship: a relationship where the related fields from the child table are part of the key which identifies a unique record in this table. It indicates a strong connection, such as the relationship between tables "Order details" and "Products", for example. There are no "Order details" without a related "Product", and the "Product" is part of the key of the "Order details" table.

All relationships present in the project are listed under Relationships on the Project explorer, and can also be visualized in a diagram.

Double-clicking on a relationship from the list opens the Relationship editor, where you can view and update all of its properties and settings.

Diagrams
In TMS Data Modeler a diagram acts as a view from the system or part of the system, unlike most database tools where diagrams are the main element of the project, defining all tables and structures.

Several different diagrams can be created, each of them with selected tables so you can better examine and edit their properties and relationships. Diagrams show all relationships between the inserted tables automatically, and allows full edition of the database structure. It is possible to update and remove objects, create new tables and relationships and so on.

All diagrams created in the project are listed under Diagrams on the Project explorer. Double-clicking on a diagram opens the Diagrams editor where you can view and update all of its tables and settings.

Extra objects
Besides the main elements tables and relationships, you may also create and edit extra objects: generic database objects that do not present a strong connection with the tables in a TMS Data Modeler project. Note that not all target DBMS support all project objects, such as procedures, generators and views.

Procedures
Procedures (or "stored procedures") are functions programmed into the database, like triggers. Through TMS Data modeler, you can edit and create procedures as shortcuts to regularly executed actions, and store them for future use.

All procedures created in the project are listed under Procedures on the Project explorer. Double-clicking on a procedure opens the Procedures editor, where you can view and update all of its properties and settings.

Generators
Generators are a resource to generate number sequences. You are able to specify an initial number and the desired increment, and each time a value is selected in this Generator, a new number is generated. It is an useful tool to fill sequential fields, automatically numbering them. Some DBMS such as SQL Server have automatic auto-increment options (available on the fields editor when selecting Int(identity) as logic type).

All Generators created in the project are listed under Generators on the Project explorer. Double-clicking on a Generator opens the Generators editor, where you can view and update all of its properties and settings.

Views

Views are a stored queries into databases. By creating a view, you are able to select frequently used query processes and save them for future use. For example, you can create a view called AlphabeticalList using the structure:

```
create view "AlphabeticalList" AS
SELECT Products.*, Categories.CategoryName
FROM Categories INNER JOIN Products ON Categories.CategoryID =
Products.CategoryID
WHERE (((Products.Discontinued)=0))
```

This will list all products on the database in alphabetical order.

All views created in the project are listed under Views on the Project explorer. Double-clicking on a view the Views editor, where you can view and update all of its properties and settings.

Domains

Domains are resources that allow you to define a special type of field and reuse efficiently these definitions in various fields in the database. Selecting Domains on the Create Tab on the Home Ribbon, it is possible to create a domain called "ZIPCODE" for example and define its properties in the Domains editor, such as: data type text, size 5, default value "00000" and so on.

Afterwards, when creating a zip code field in a table, you are able to select the domain ZIPCODE and import all of these settings. Specifying all field properties again becomes unnecessary, which is extremely useful not only to avoid reworking when creating fields but also for maintenance and consistency. If it ever becomes necessary to increase the field size, for example, instead of making this change field by field, table by table, you just need to adjust a single domain.

The Project Files

TMS Data Modeler works with a model that represents the physical database. Through this model, scripts can be generated to implement changes in an existing physical database or to create a physical database from scratch.

Creating a project will generate the following files:

- A single .dgp file is created for each project, containing project settings and the structure of all data dictionary objects. You may select the file directory and name when saving your project.

- Conversion maps created by users are saved in the Application Data folder, inside TMS Software\Data Modeler\Conversions subdirectory. Data Modeler's default conversion maps are located in the application directory.

- Connections are saved on the file UsrConnections.bin, located in the Application Data folder inside TMS Software\Data modeler subdirectory.

- Different project versions are saved by default in the subdirectory \versions inside the project directory (%projectdir%). This working directory can be changed on the General settings dialog. To open this dialog, select Settings on the Project tab on the Tools ribbon, and click on the Version control tab, as shown below.

Each new version file is saved under the project file name with its extension corresponding to the version number, e.g. demo.1, demo.2 etc.

Chapter

II
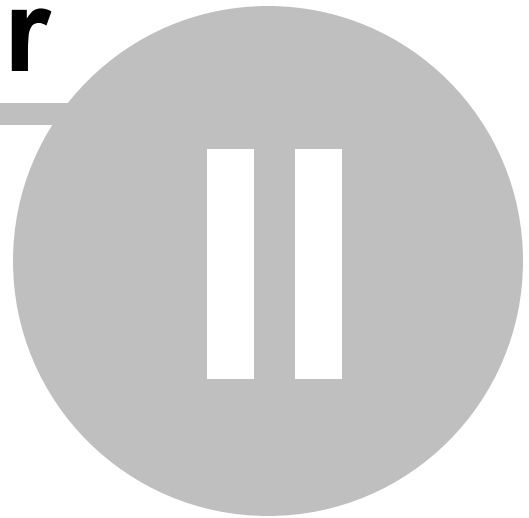
The User Interface

2   The User Interface

TMS Data Modeler offers a clear and intuitive user interface using Ribbons and tabs to organize and edit all database information.

UI Overview
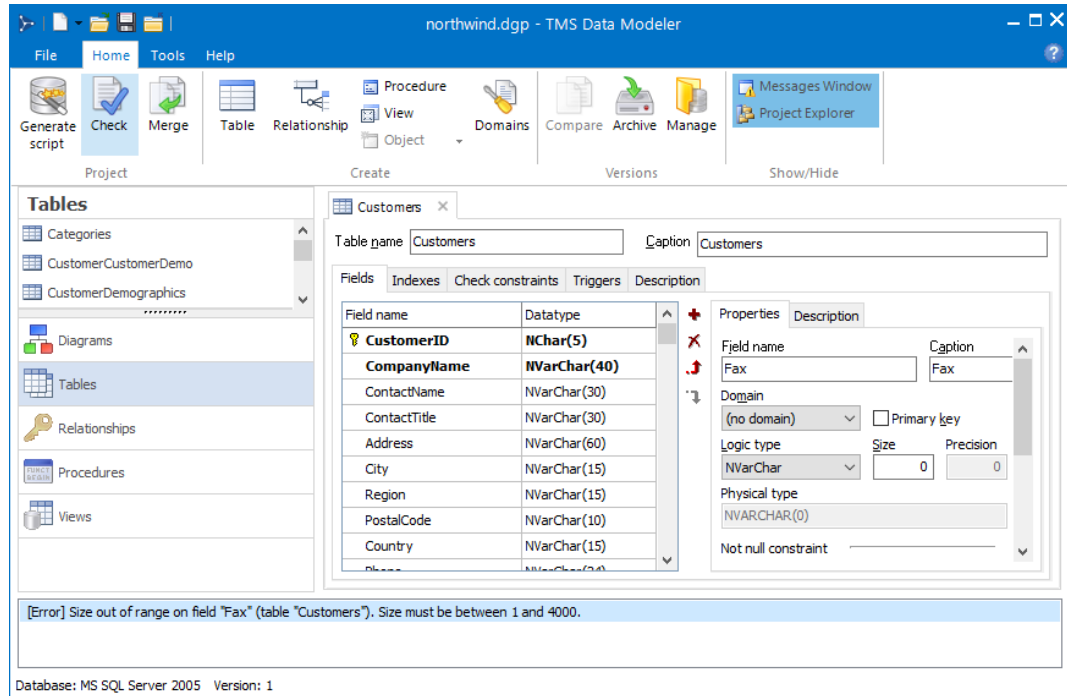Ribbons
Project Explorer
Workspace
Messages Window

2.1   UI Overview

TMS Data Modeler Home organizes application options into Ribbons, keeping the most used tools at hand when visualizing or editing any part of the project. Contextual tabs are enabled when specific objects are selected, allowing improved usability.

The Project explorer on the left allows you to visualize your whole project easily, listing object types that when selected show all items in a clear structure. To edit or view details of any of the project items, you just have to double-click.

When selecting an item by double-clicking, the item tab will open in the Workspace, on the right. The workspace may contain one or more tabs, and each tab is subdivided into sections to offer a clear and efficient view of the selected object.

The Message window shows notifications when validating a project, such as warning and error messages.
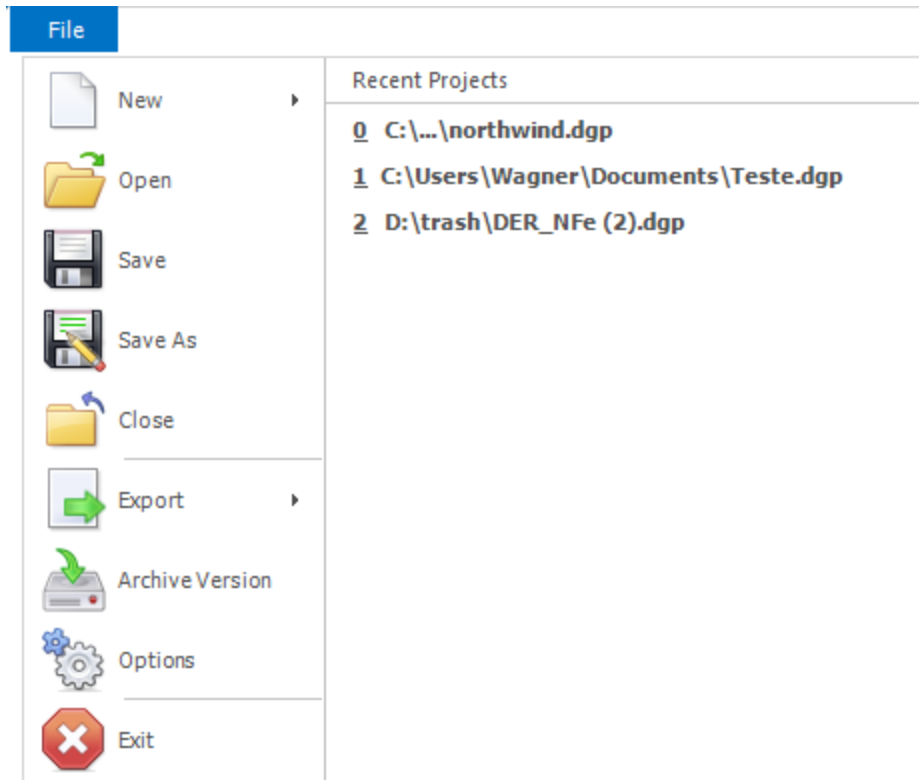
2.2   Ribbons

TMS Data Modeler's interface is organized into Ribbons, which group all related features in one place, improving usability and making those features extremely accessible.

File Menu
A simple menu offers general project options, while the panel on the right lists projects that have been recently worked on for selection.

Menu Option	Description
New	Offers the options of creating a new blank project or importing the structure of a new project from an existing database.
Open	Opens an existing project.
Save	Saves the current project.
Save As	Prompts a new path and file name before saving the current project.
Close	Closes the current project after prompting if changes should be saved.
Export	Exports the current model to TMS Aurelius classes.
Archive version	Archives the current version of the project and starts a subsequent version. Version files are saved by default in the sub-directory \versions in the project directory.
Options	Prompts for general environment settings.
Exit	Closes the current project after prompting if changes should be saved and terminates the application.

Home Ribbon
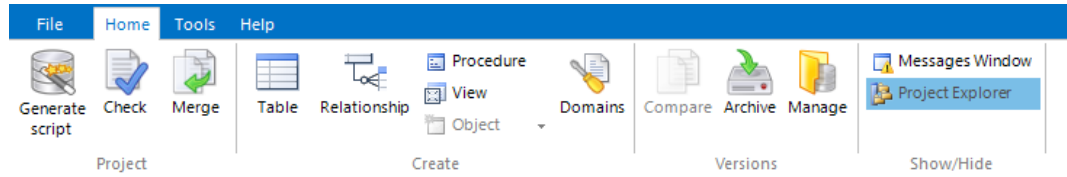
"Project" Tab Option	Description
Generate script	Prompts for information to generate a script to create a database according to the current project.
Check	Checks project information, validating all related data such as table fields and keys. Warnings and other messages are shown on the Messages window, on the bottom of the workspace. To enable this window, select Messages window on the Show/Hide tab.
Merge	Compares and allows merging of two different projects into one, by generating an impact script.

"Create" Tab Option	Description
Table	Creates a new table. After selecting it, a blank table tab will open on the table editor, where you will define all of its properties.
Relationship	Creates a new relationship. After selecting it, the New relationship dialog will open, where you will define all of its properties.
Procedure	Creates a new procedure on the Project.
View	Creates a new view on the Project.
Object	Drops down a menu to create extra objects as available per DBMS. You will find "Generators" available on Firebird, for example.
Domains	Shows existing domains information and allows editing and additions.

"Versions" Tab Option	Description
Compare	Compares versions of the current project after selection, allows generation of an impact script.
Archive	Archives the current version of the project and starts a subsequent version. Version files are saved by default

	in the subdirectory \versions in the project directory.
Manage	Allows version management and comparisons.

"Show/Hide" Tab Option	Description
Message Window	Shows or hides the message window, at the bottom of the workspace. This window lists warning and error messages when the validation tool is used.
Project Explorer	Shows or hides the Project explorer, on the left of the workspace.

Design Ribbon
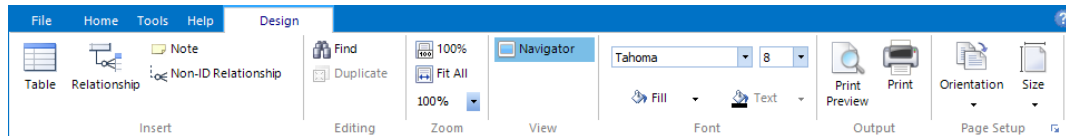
The items on this Ribbon are only enabled when a Diagram tab is selected on the workspace.

"Insert" Tab Option	Description
Table	Creates a new Table. After selecting it, you must drag and drop the table on the diagram. The Table editor will open for you to adjust table settings.
Relationship	Creates a new identifying relationship. After selecting it, you must drag the relationship line connecting parent and child tables. The Add relationship dialog will open for you to adjust relationship settings.
Note	Creates a new note after releasing it on the diagram tab.
Non-ID Relationship	Creates a new non-identifying relationship. After selecting it, you must drag the relationship line connecting parent and child tables. The Add relationship dialog will open for you to adjust relationship settings.

"Editing" Tab Option	Description

| Find | Show/hide the Find panel at the bottom of diagram, allowing to search for tables in diagram by table name |
| Duplicate | Allows duplicating the selected table. |

"Zoom" Tab Option	**Description**
100%	Sets the current diagram zoom to 100%.
Fit to All	Sets the current diagram zoom to a value that allows a view of the whole diagram in the current window.
Zoom Combo	Allows setting a specific zoom value for the current diagram.

"View" Tab Option	**Description**
Navigator	Show/Hide the Diagram Navigator Panel.

"Font" Tab Option	**Description**
Font	Allows for changing font settings for the currently select table(s).

"Output" Tab Option	**Description**
Print Preview	Opens a dialog that displays a preview for printing the diagram.
Print	Opens the print dialog for printing the diagram.

"Page Setup" Tab Option	**Description**
Orientation	Chooses between Landscape/Portrait orientation of diagram
Size	Chooses paper size for diagram printing/paging.

Tools Ribbon
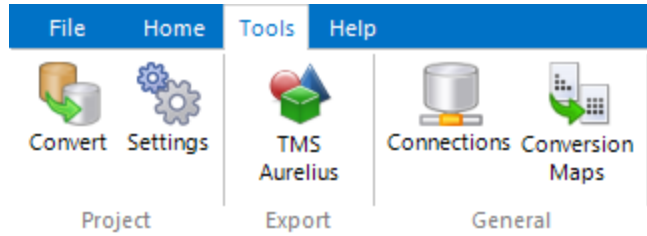
"Project" Tab Option	Description
Convert	Converts current project into another DBMS.
Settings	Shows and allows edits on current project settings.

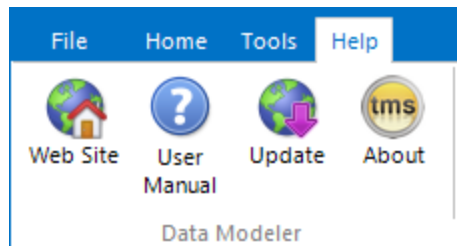| "Export" Tab Option	Description
TMS Aurelius	Opens a dialog to export existing database schema to Aurelius classes

"General" Tab Option	Description
Connections	Shows existing database connections and allows adding, removing or editing them.
Conversion Maps	Allows you to create a conversion map with specific data type equivalences to use when converting a project from a DBMS to another.

Help Ribbon

"Data Modeler" Tab Option	Description
Web Site	Opens TMS website on the standard web browser.
User Manual	Opens the user manual as help file.
Update	Checks if a new Data Modeler version is available and if true, prompts for updating the software.

About	Opens About windows.

2.3　Project Explorer

The project explorer is on the left side of the screen, and lists all items of the project. By clicking any of the data dictionary buttons on the bottom, you will find all objects, such as Tables, Relationships, Procedures, and Diagrams. You can hide it or show it by selecting Project explorer on the Show/Hide tab on the Home Ribbon.

By double-clicking in any of the available objects, its specific editor will open in the workspace, allowing visualization and editing. If the selected object has already been opened on the workspace, a single click will focus on its existing tab. By right-clicking in any point of the project explorer, a context menu will open with options such as creation of new objects and removal of selected objects
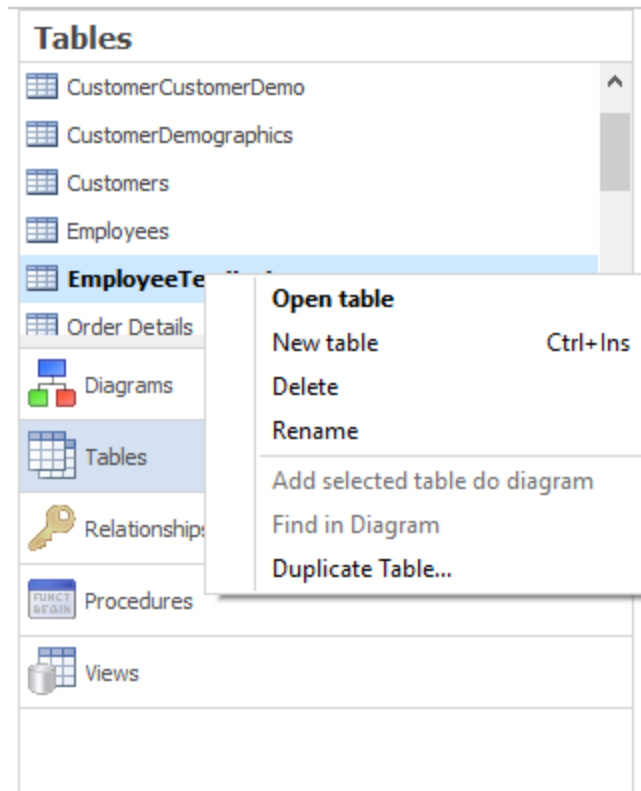
2.4　Workspace

The project workspace is the area on the right that organizes the project information into tabs. When double-clicking any object from the project explorer, its specific editor will open on the workspace, on a new tab. After editing, closing any tab will automatically save changes.

Here is a sample view of the Table editor open on the Workspace. Other tabs already open are diagram and relationship editors.
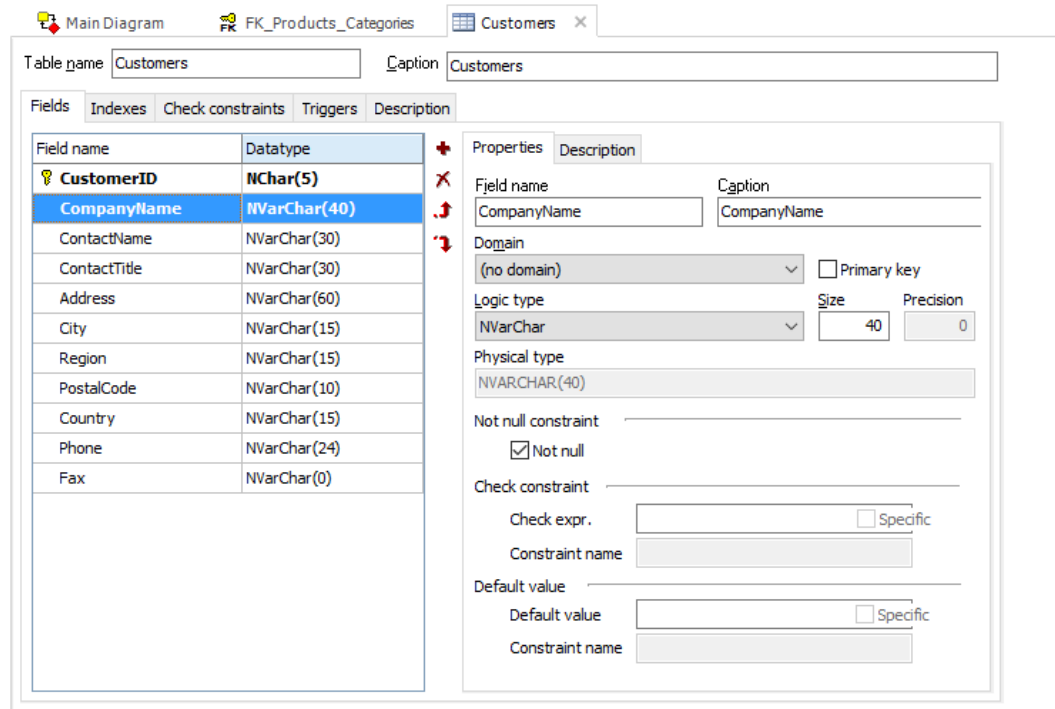
2.5   Messages Window

The Message window is a text box at the bottom of the application which shows notifications about the project. To enable this tool, select Messages window on the Show/Hide tab on the Home ribbon. To validate a project and list messages about any problems found, select Check on the Project tab on the Home ribbon.

Chapter

III
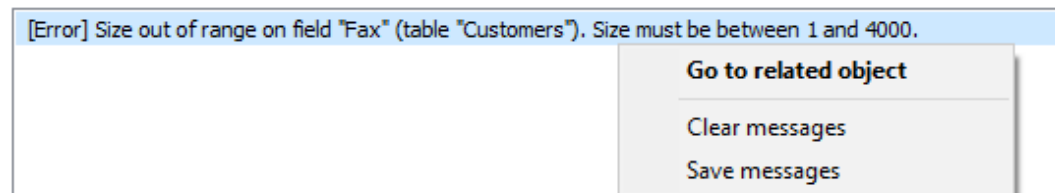
Basic Operations

3   Basic Operations

The following topics describe the most common used operations in TMS Data Modeler.

Creating a New Project
Creating Objects in Database Model
Comparing projects
Generating Database Creation Scripts
Project Validation
General Project Settings
Version Control
Reverse Engineering
Convert Project to Different Database

3.1   Creating a New Project

To create a new blank project

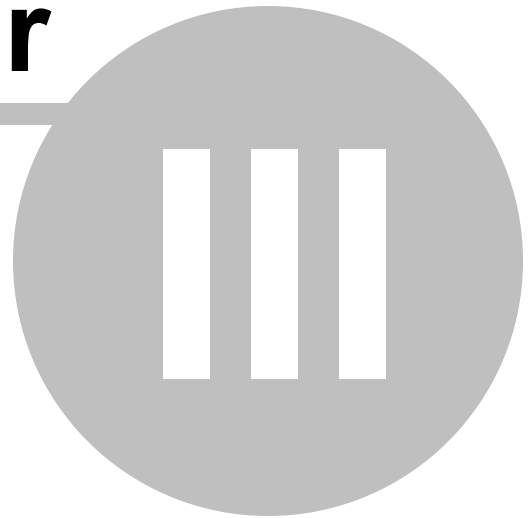1. Select New / New project on the File Menu or select New blank project after clicking on the New icon in the quick access toolbar.

2. On the New Project dialog, select the desired target database and click Ok.

You can also use the shortcut Ctrl+N or click on the icon of the project toolbar to open the New Project dialog.

To create a new project from existing database

The reverse engineering tool from TMS Data Modeler creates a project from an existing database, so you can manage, view and edit all current information easily. Check the desired DBMS step-by-step process to create your project:

Select New / Import from database from the File Menu or select New project from database after clicking the New icon in quick access toolbar. On the Data dictionary import dialog, you can either select an existing connection or a new connection.

If you select Existing connection:

1. Choose the Connection name from the list and click Next.

2. The Data dictionary import dialog will open, showing a progress bar so you can follow the import process. When it's finished, the message Done will appear and the Next button will be enabled again. Click to continue. This will enable the button Finish. After clicking, the process will be complete.

If you select New connection:

1. Choose the Database type from the list and click Next.

2. Different DBMS will ask you for different information to be filled in on Data dictionary import dialog. It is necessary to complete this screen before proceeding.

3. Fill in the connection settings then click on "Test connection". If there are no connection problems, click Next.

4. The Data dictionary import dialog will show a progress bar so you can follow the import process. When it's finished, the message Done will appear and the Next button will be enabled again. Click to continue.

5. The last screen of the Data dictionary import dialog will allow you to save the current connection settings for future use. Select the checkbox and type the connection name if you wish to do so. Otherwise, uncheck the option and click Finish.

3.2 Creating Objects in Database Model

Creating a new Table

1. Select Table on the Create tab on the Home ribbon or Right-click on the Project explorer and select Add table . You may also use the shortcut Ctrl+T to create a new table.

2. The Table editor will open on the Workspace. You can add or edit all table data by selecting the different internal tabs: Fields, Indexes, Check constraints, Triggers and Comments.

To create fields in a table

1. Open the desired table by selecting Tables on the Project explorer and double clicking on the table's name on the list.

2. The Tables editor opens with the Fields tab selected by default. This tab is divided into three sections: Fields list, Field properties tab and Description tab.

3. On the Fields list, click on the Add button or right-click on the list background and select Add field. You may also create a new field by using the shortcut Ctrl+F.

4. Enter the field data on the Field properties section. To get detailed information, take a look at the Fields editor topic.

To create a new relationship

1. Select Relationship on the Create tab on the Home ribbon or right-click on the Project explorer and select New relationship.

2. On the New relationship dialog, select the relationship type by clicking on one of the buttons:

Non-identifying relationship: it's the most usual type of relationship. It represents a weak connection, a relationship between parent and child table that does not involve a key field, such as the relationship between tables "Products" and "Categories", for example.

Identifying relationship: a relationship where the related fields from the child table are part of the key which identifies a unique record in this table. It indicates a strong connection, such as the relationship between tables "Order details" and "Products", for example. There are no "Order details" without a related "Product", and the "Product" is part of the key of the "Order details" table.

3. Select the Parent table and the Child table from the lists, and click Ok.

4. The Add relationship dialog will open, allowing you to specify all relationship data:

In the Relationship properties section, you may visualize or set the relationship name and its description, for future reference. A default relationship name is created automatically when adding a new relationship to the project, combining parent and child table names. The relationship name may be edited at any time.

In the Relationship keys section you may visualize or edit the relationship keys by selecting them from the available list.

- Parent key: Lists all available keys (primary keys and indexes) from the parent table.
- Parent table: This column shows all fields on the parent table which are part of the selected parent key. It is not possible to modify this data.
- Child table: This column lists all fields on the child table that are compatible with the selected field on the parent table (same type). By default a new field listed as "field_name(new)" with the same name and type of the parent key is created and selected. It is possible to select any other available child field from the list.

In the Relationship options section, you are able to select the relationship behavior when a record on the parent table is deleted or updated:

- No action: no action is taken when there are changes on the parent table. This is the default option.
- Cascade: automatically deletes all records on the child table when a parent record is deleted. Example: Customer/Contact, when deleting a customer, all of its contacts are also deleted.
- Set null: when a parent record is deleted, all child records related to it get this field set to null. Example: on the Person/Gender deletion, all people related to this record would be set to null.
- Set default: similar to "set null" option, but instead of setting the field to null, it is filled with its default value.

5. The created relationship will be listed under the Relationship branch of the Data dictionary tree. To edit any of the relationship data, double-click on the relationship on the Project explorer and the Relationship editor will open.

You may also create a relationship visually, by using a Diagram. It will open the same New relationship dialog to allow properties setting.

3.3   Comparing projects

To compare two different projects, you may use the Merge projects tool. It will allow you to visualize their differences and to generate an impact script to update your database with information on both projects.

1. Select Merge on the Project tab on the Home ribbon.

2. On the Compare projects dialog, select the path and the file of the project you want to merge with the current one and click next.

3. The Compare projects dialog will now list each project's objects. All selected objects are organized hierarchically in two synchronized project trees. You are able to edit the information below:

- Hide unchanged items: when selected, this option hides all items that are equivalent in both projects. The only items shown are the ones that differ or are present in only one of the projects. By default, this option is unchecked and all objects are shown, with the differences highlighted in bold.
- Filter the desired objects: selects which objects are compared on the project trees. The options are Tables, Indexes, Relationships, Triggers, Domains, Procedures and Views. By default all objects are checked.
- Action: defines which action will be taken regarding the differences found after project comparison. By default, the option Generate database script is selected, and results in the generation of an impact script to update the database with all the selected differences after clicking Generate.
- Project trees: the objects that differ between the projects are in bold.
    - The middle column allows you to select changes to be included on the impact script. Selected changes are marked with a fingerpoint icon.
    - By expanding the bold objects you will get to the exact item where the difference is. For example, if a field exists in one project but does not exist on the other, the corresponding tables are in bold. When expanding the table, the item Fields is in bold, and when expanding Fields, only the different fields remain in bold.
    - When clicking on the bold object, the object creation scripts are compared side by side highlighting the differences on the text boxes below.
    - When an object is non-existent in one of the projects, it is signalized on this project tree with the text '(not exists)' besides the equivalent field on the other project. In this case, the impact script will generate commands to add or remove the item, as shown below.

4. By clicking Generate, an impact script will be created and shown to allow you to update your current database with the merged projects' settings. Selecting Back will take you to the dialog to select other projects to be compared. Selecting Close will close the dialog without saving any changes.

3.4   Generating Database Creation Scripts

TMS Data Modeler's database generation tool allows you to plan and design your database in a single project, creating a script that generates your database automatically. You may also create impact scripts to update a database that has already been created before, by comparing versions or projects.

To generate a database creation script

1. Select Generate script on the Project tab on the Home ribbon or press F9.

2. The Script generation dialog will open with the following information:

- Show the script: selecting this option will allow you to edit any further details by opening a dialog with the script after clicking Generate.
- Save script to file: selecting this option will allow to choose the file path and file name and save it for later use without opening it after clicking Generate .
- Workspace: an item tree where you can check the desired items to be deployed. You may check one or more objects.

3. On the Tables tab, all project tables are listed by name. All tables are selected by default. You may choose which tables will be created through the script.

4. After selecting all desired items and tables, click Generate. The tab Process will appear, listing all steps of the script generation.

3.5   Project Validation

TMS Data Modeler offers a tool to validate all project items, checking consistency and settings of objects.

To validate your project, select Check on the Project tab on the Home ribbon. All project messages are then shown on the Messages window, which is at the bottom of the application, under the workspace.

- **Error message**: refers to an error on the settings of a project object that will cause problems when generating a database, for example a relationship between two tables without defined fields/keys.
- **Warning message**: refers to a possible consistency problem that may cause problems when generating a database, for example a relationship between two tables where the selected fields/keys are not compatible.

Right-clicking on any message opens a context menu will allowing you to Go to the related object, Clear messages or Save messages. You can also go to the related object by double-clicking on the message.

Below follows a list with all possible validation messages. Messages about [object] are regarding extra objects, such as procedures, views and generators.

Error Messages
Constraint has no name on table [table]
Duplicate constraint name [constraint] on table [table]
Duplicate domain name [domain]
Duplicate field name [field] on table [table]
Duplicate index name [index] on table [table] (all DBMS except Firebird): a table contains more than one index with the same name. Index names must be unique on the table.
Duplicate index name [index] (table: [table]) (only on Firebird): the project contains more than one index with the same name. Index names must be unique on the database.
Duplicate [object] name [name]
Duplicate relationship name [relationship]
Duplicate table name [table]
Duplicate trigger name [trigger] on table [table]
Empty expression on computed field. Field: [field] on table [table]
Field has no name on table [table]
Identity field cannot have a default value. Field: [field] on table [table]
Increment value cannot be zero on identity fields. Field: [field] on table [table]
Index has no name on table [table]
Index [index] on table [table] has no linked fields
Invalid constraint name [constraint] on table [table]
Invalid field name [field] on table [table]
Invalid index name [index] on table [table]
Invalid [object] name [name]
Invalid relationship name [relationship]
Invalid table name [table]
Invalid trigger name [trigger] on table [table]
Missing child field on relationship [relationship] (link #[N])
Missing parent field on relationship [relationship] (link #[N])
[Object] has no name
Relationship has no name
Relationship [relationship] has no linked fields
Relationship [relationship] is self-referencing and can only accept ON DELETE NO ACTION and ON UPDATE NO ACTION methods. (only on SQL Server)
Size out of range on field [field] (table [table]). Size must be between [min] and [max].
Table has no name
Table [table] contains two identity fields
Trigger has no name on table [table]

Warning Messages
Constraint name [constraint] on table [table] is a reserved word
Field name [field] on table [table] is a reserved word
Incompatible data types ([parent field]/[child field]) on relationship [relationship]
Index name [index] on table [table] is a reserved word
Missing expression on check constraint [constraint] (table [table])
Name too long for constraint [constraint] on table [table]. Maximum size is [size] characters.

Name too long for field [field] on table [table]. Maximum size is [size]
characters.
Name too long for index [index] on table [table]. Maximum size is [size]
characters.
Name too long for [object] [name]. Maximum size is [size] characters.
Name too long for relationship [relationship]. Maximum size is [size] characters.
Name too long for table [table]. Maximum size is [size] characters.
Name too long for trigger [trigger] on table [table]. Maximum size is [size]
characters.
[Object] name [name] is a reserved word
Object] [name] has no create implementation
Parent index [index] of relationship [relationship] is not unique
Relationship name [relationship] is a reserved word
Size was not specified on field [field] (table [table])
Table name [table] is a reserved word
Table [table] has no fields
Table [table] has no primary key
Trigger name [trigger] on table [table] is a reserved word
Trigger [trigger] on table [table] has no implementation

Open.bmp Related Topics

3.6   General Project Settings

To edit general project settings

1. Select Settings on the Project tab on the Tools ribbon

2. The Settings window will open

To edit general settings click on the tab Information. You are able to edit the
Project name, Author and Description.

To select the working directory for your project versions, click on the tab
Version control. The working directory where project versions are saved is
shown. You can use the default \versions directory or select a different
directory by clicking on the folder icon.

3.7   Version Control

TMS Data Modeler allows you to version control your database model. This
means you can archive (snapshot) your existing model into versions and then
later compare versions to check differences between them and generate SQL
update scripts to "upgrade" the database schema from one version to another.

Creating (archiving) versions

To archive the version you are currently working on and start the next one,
select Archive on the Versions tab in the Home ribbon or click on the File Menu
 button and select Archive version.

The dialog will allow you to add any relevant information to identify this version later. By clicking Archive you will automatically close this version and start the subsequent one.

Version management window

To open the version management window, select Manage on the Versions tab in the Home ribbon.

The Project versions window will list all saved versions, the last date/time of the alterations and the complete file name.

Compare versions

To perform a comparison between versions:

1. Select Compare in the Versions tab of the Home ribbon.

2. You will be prompted to select the Base version and the Compare to version. Any existing version can be chosen.

3. The Compare versions window will list each version's objects. All selected objects are organized hierarchically in two synchronized project trees. You are able to edit the information below:

- **Hide unchanged items**: When selected, this option hides all items that are identical in both versions, unchanged by update/insert/delete. The only items shown are the ones that differ. By default, this option is unchecked and all objects are shown, with the differences highlighted in bold.

- **Filter the desired objects**: select which objects are compared on the project trees. The options are Tables, Indexes, Relationships, Triggers, Domains, Procedures and Views. By default all objects are checked.

- **Action**: defines which action will be taken regarding the differences found after version comparison. By default, the option Generate database script is selected, and results in the generation of an impact script to update the database with all the selected differences after clicking Generate.

- **Project trees**: The objects that are differ between the versions are in bold.

   o The middle column allows you to select what changes will be included on the impact script. Selected changes are marked with a icon_checkbox.jpg checkbox icon.

   o By expanding the bold objects you will get to the exact item where the difference is. For example, if a field exists in one version but does not exist on the other, the corresponding tables are in bold. When expanding the table, the item Fields is in bold, and when expanding Fields, only the different fields remain in bold.

   o When clicking on the bold object, the object creation scripts are compared side by side highlighting the differences on the text boxes below.

o When an object is non-existent in one of the versions, it is signalized on this project tree with the text '(not exists)' besides the equivalent field on the other project. In this case, the impact script will generate commands to add or remove the item.

4. By selecting Generate, an impact script will be created and shown to allow you to update your current version with the compared versions' settings. Selecting Back will take you to the dialog to select other versions to be compared. Selecting Close will close the dialog without saving any changes.

3.8 Reverse Engineering

The reverse engineering tool from TMS Data Modeler creates a project from an existing database, so you can manage, view and edit all current information easily.

To create a new project from an existing database, select New / Import from database from the File Menu or select New project from database after clicking the New icon in Quick Access Toolbar. In the Data dictionary import dialog, you can either select an existing connection or a new connection.

If you select existing connection:

1. Choose the Connection name from the list and click Next.

2. The Data dictionary import dialog will open, showing a progress bar so you can follow the import process. When it's finished, the message Done will appear and the Next button will be enabled again. Click to continue. This will enable the button Finish. After clicking, the process will be complete.

If you select New connection:

1. Choose the Database type from the list and click Next.

2. Different RDBMS will ask you for different information to be filled in on Data dictionary import dialog. It is necessary to complete this screen before proceeding. Fill in the specific connection settings according to the RDBMS you have chosen.

3. Click on Test connection to make sure your settings are correct. If there are no connection problems, click Next.

4. The Data dictionary import dialog will show a progress bar so you can follow the import process. When it's finished, the message Done will appear and the Next button will be enabled again. Click to continue.

5. The last screen of the Data dictionary import dialog will allow you to save the current connection settings for future use. Select the checkbox and type the connection name if you wish to do so. Otherwise, uncheck the option and click Finish.

3.9   Convert Project to Different Database

Each Data Modeler project has a specified target database. You can change the target database which will perform a conversion operation in the project, changing the table column types from one database to another. To help you in the process, you can use field mapping concept using conversion maps.

Conversion Maps

Field mapping is the process of selecting equivalences of data type between different RDBMS. Using these maps, a project can be converted from one RDBMS to another without data loss. To view all existing field/conversion maps open the Conversion maps dialog, by selecting Conversion maps in the General tab of the Tools ribbon.
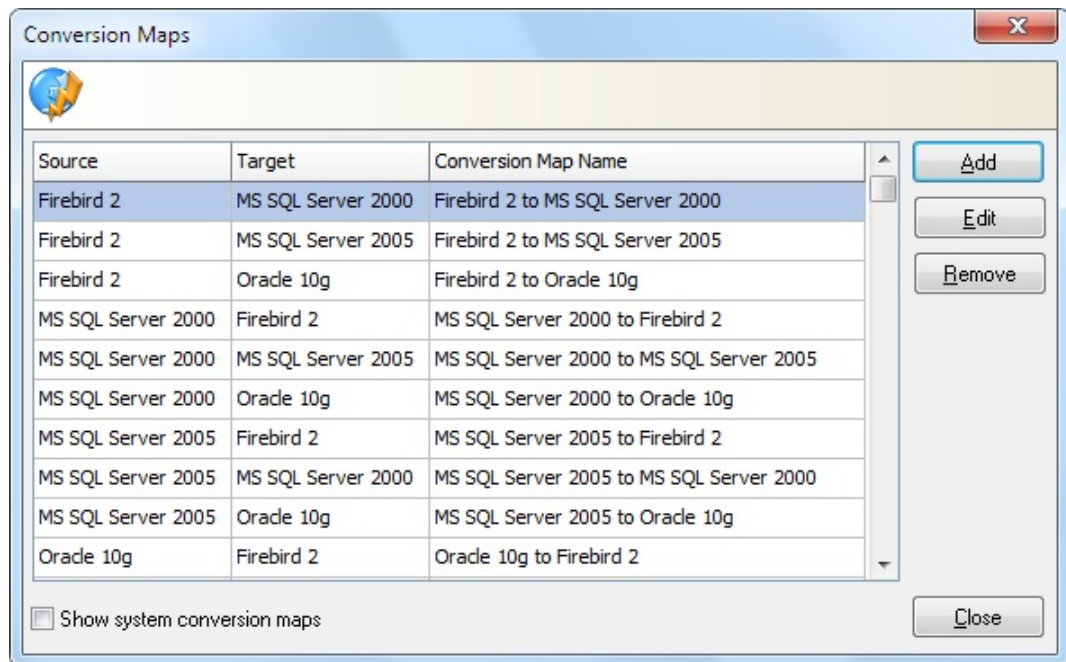
This dialog enables you to edit or remove existing conversion maps and to create new ones by clicking on the appropriate button. By default, only conversion maps created using TMS Data Modeler are displayed. To see all conversion maps available in your system, select the checkbox Show system conversion maps at the bottom of the screen.

By clicking Edit, the Data type conversion map dialog will open, allowing you to visualize and edit all conversion map info:

- **Source database**.

- **Target database**.

- **Name**: Name of this particular conversion map.

- **Conversion map**:

    o Source type: shows all field types available on the Source database.

   o Target type: shows data types supported by the Target database that are compatible to the corresponding Source type. This information will be used when converting projects between databases.

    o Size/length and Precision: allows setting of some data types, such as numeric fields, which is then used for this data type in conversions to the target database. The option Keep (default) uses the same property value from the source database on the target database.
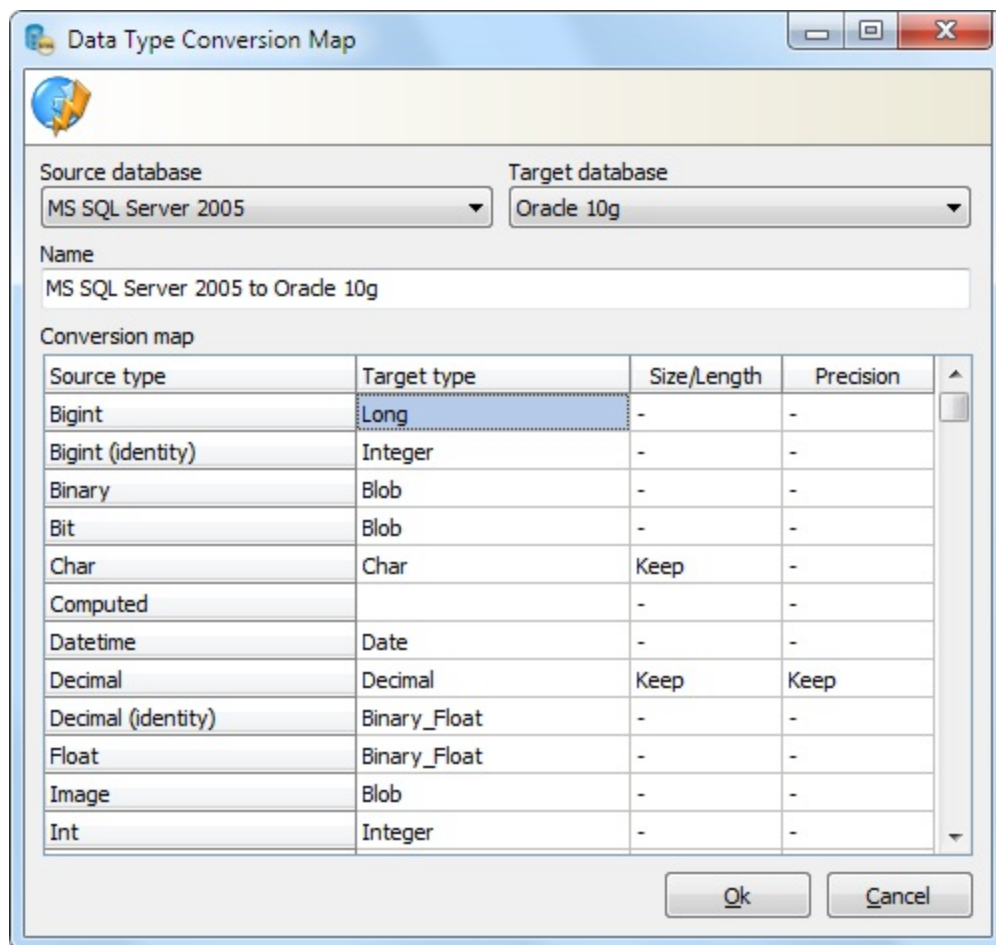
Database conversion

Before converting your project between databases, make sure you have a field map for them. Having created the field map, select Convert in the Project tab of the Tools ribbon. The Convert database dialog will open, enabling you to select your target database. Only mapped databases will appear on the list. After selection, click Ok.
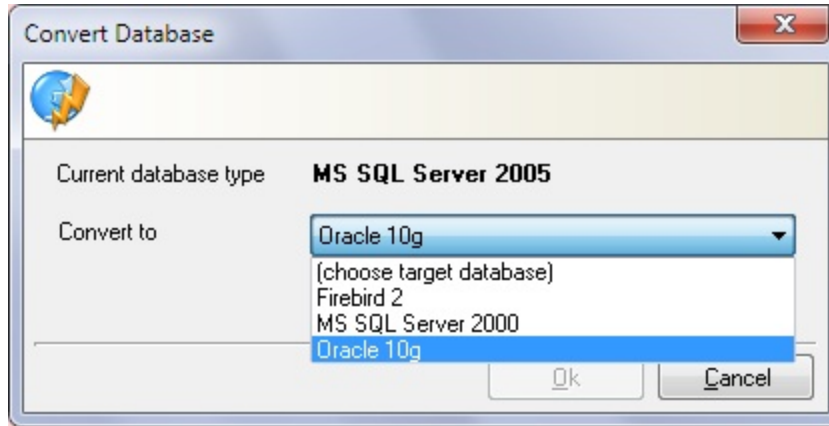
Chapter

IV

Editors

4   Editors

The following topics present the different TMS Data Modeler editors.

Table Editor
Domains
Diagrams
Relationship Editor
Procedures and Functions
Generators/Sequences
Views

4.1   Table Editor

The table editor allows you to view and edit all table settings, such as its Fields, Indexes, Constraints, Triggers and Comments. Each of these items has all settings organized inside tabs on the Table editor, to improve visualization.

To access the table editor, double-click on the desired table on the Project explorer or on a Diagram. When opening a table by double-clicking on a diagram, the Back to diagram button is enabled, so you can easily return to the original diagram.

Fields Tab

The fields tab inside the table editor is divided into three sections: Fields list, Field properties tab and Description tab.

The fields list shows all field names and types. You may add, remove or reorder all fields using the icons on the top left of the list. Right-clicking on any part of the list will open a context menu with these same functions, plus a duplicate field option and the option to copy field to clipboard. Copying to clipboard offers three options: List of fields, which will create a comma-separated list of all fields in this table, INSERT command, which will create an insert script to generate a record with all the fields, and UPDATE command, which will create an update script.

All required fields are shown in bold. The vertical key icon identifies fields which are part of the table's primary key. The horizontal key icon/FK text identifies a foreign key related to a parent table.

Selecting the desired field, all of its properties can be edited in the Field properties tab:

- **Field name**.

- **Domain**: the domain that will be used to define settings such as type and size of the selected field. All settings automatically loaded from the domain will be filled and their respective editors will be disabled.

- **Primary key**: defines if the selected field is part of the table's primary key. These keys may be set by selecting the checkbox or through the Indexes tab where specification and ordering of these fields can be done.

- **Logic type**: defines the concept type of the selected field, usually corresponding to the physical type in the database. All data types supported by the DBMS in use are available for selection. Depending on your choice of logic type, different options will be enabled in the editor. Selecting any Identity type, such as Int (identity) on SQL Server, the Identity section will be enabled, allowing you to set automatic increments on the field.
  - **Seed**: defines the initial number of an auto-increment field.
  - **Increment**: defines the increment value of an auto-increment field.

- **Size**: this editor will be enabled when applicable, as for alphanumeric types.

- **Precision**: decimal and numeric types will enable this editor. It defines precision for these data types.

- **Physical type**: non-editable. It displays the settings of the physical type applied in the field when the database is generated, based on specifications of logic type, size etc. This editor shows the exact definition of this field on the generated script.

- **Not null constraint**: it will require a not-null value for the field when inserting or updating a record, making this a required field.

- **Check constraint**:
  - **Check expr.**: a formula for validation / condition that must always be true. For example, the field age could have a check constraint of "Age > 18". It will not be possible to insert a record in this table that does not satisfy this condition. This editor is automatically filled and disabled for changes if an existing domain with a check constraint is selected on the Domain list.
  - **Specific**: is only enabled when an existing domain with a check constraint is selected on the Domain list. If checked, it allows changes on the expression only applied to this specific field. It's unchecked by default.

- **Constraint name**: optional information, enabled when an expression is entered on the Check expr. editor.

- **Default value**:
  - **Default value**: the field is automatically filled with a specific value when inserting a new record in the table. This editor is automatically filled and disabled for changes if an existing domain with a default value is selected on the Domain list.
  - **Specific**: is only enabled when an existing domain with a default value is selected on the Domain list. If checked, it allows changes on the value only applied to this specific field. It's unchecked by default.

- **Constraint name**: optional information, enabled when a value is entered on the Default value editor.

Index Tab

The indexes tab inside the table editor is divided in three sections: Indexes list, Index properties and Index fields.

The Indexes list shows all indexes present on a table. You may add or remove indexes by using the buttons on the top right of the indexes list or by right-clicking anywhere on its area. A key field is identified by the key symbol.

In the Index properties section you may have two data editors:

- **Index name**: it is always enabled, allowing you to add or change the index name.

- **Index type**: is only enabled when a non-primary index is selected. Valid options are:
    - Non exclusive: Index does not enforce any validation, it's just used for performance
    - Exclusive: Index is exclusive (unique) which means it won't allow duplicated values for the index fields
    - Unique Key: Index is actually an Unique Key constraint. Exclusive and UniqueKey usually have the same effect, the difference is when index is "Unique Key", it will be created as unique constraints in table (usually the database will create an internal exclusive index to enforce the unique key.

The Index fields section, you may add or remove fields from the index. When clicking Add field to index, all available fields on the table will be shown for your selection. The fields in the index can be ordered by clicking on the heading of the column order, allowing Ascending (default) or Descending options.

Check Constraints Tab

The Check constraints tab inside the table editor is divided in two sections: Constraints list and Constraints properties.

In the Constraints editor you may view all constraints related to the selected table on the Constraints list. By selecting them, you can edit its name and expression in the Constraint properties section. You may add or remove constraints by using the buttons on the top right of the constraints list or by right-clicking anywhere on its area.

Triggers Tab

The Triggers tab inside the table editor is divided in tree sections: Triggers list, Trigger properties and Implementation.

In the Triggers list, all triggers related to the selected table are listed. Triggers are procedures executed every time an update / insert / delete is executed in a given table. It works as a code programmed in the DBMS language.

In Trigger properties, you may edit the name of this trigger and its description.

In Implementation, the complete command to generate the trigger in the database is shown for edition. When you create a trigger, it automatically implements "CREATE TRIGGER <%TriggerName%> ON <%TableName%>". These macros "<%...%>" are replaced by the trigger's and the table's names, so you

can rename both trigger and table without having to update this implementation.

4.2   Domains

To create domains

1. Select Domains on the Create tab on the Home ribbon.

2. On the Domains editor dialog, you may create a domain by right-clicking on the Domains lists or selecting of the Add buttons. This dialog allows creation of two types of domains, which are indicated at the bottom of the screen as 'Logical domain (not in database)' or 'Physical domain (kept in database)':

Physical domain: created as a 'domain' object and kept in the database (command 'CREATE domain'). A field using this type of domain is generated through a direct reference to this object.

Logical domain: used only on Data Modeler's project, not created physically in the database. A field using this type of domain is generated with the properties defined by this domain.

3. You are able to set all domain data on the General tab:

- **Domain name**.

- **Data type**: lists all available data types on the DBMS in use.

- **Physical type**: non-editable. It displays the settings of the physical type applied in the domain/field when the database is generated, based on specifications of logic type, size etc. This editor shows the exact type definition on the generated script.

- **Size**: this editor will be enabled when applicable, as for alphanumeric types.

- **Precision**: decimal and numeric types will enable this editor. It defines precision for these data types.

- **Seed**: identity types will enable this editor. It defines the initial number of an auto-increment field.

- **Increment**: identity types will enable this editor. It defines the increment value of an auto-increment field.

- **Default value**: the field is automatically filled with a specific value when inserting a new record in the table.

- **Constraint**: a formula for validation / condition that must always be true. For example, the field age could have a check constraint of "Age > 18". It will not be possible to insert a record in this table that does not satisfy this condition.

4. On the Information tab, you are able to add any documentation or description info to better identify the domain later. The Usage tab allows you to visualize all the domain's related fields on all project tables.

5. By clicking Close, all updates will be saved.

Using Domains

To associate a field with an existing domain, go to the Fields tab on the Table editor. On the Properties tab, select the desired domain from the list. All settings automatically loaded from the domain will be filled and editors related to these settings will be disabled.

You can manage all domain information in the Domain editor. To open the editor, select Domains on the Create tab on the Home ribbon.

By selecting a domain from the Domain list, you are able to view and edit its settings on the General tab, automatically adjusting all related fields. You can also update its description by editing the Information tab, or visualize its related fields on all project tables on the Usage tab.

4.3   Diagrams

To create new diagrams:

1. Right-click on the project explorer and select New diagram or press CTRL+D.

2. A new blank diagram will open in the workspace.

3. The Design ribbon is enabled. You may edit and add objects to the diagrams on the workspace:

- by selecting the desired option on the Insert tab on the Design ribbon and dropping the item on the diagram;
- or right-clicking on the existing diagram items or on the workspace.

To insert tables in a diagram:

To insert an existing table, drag and drop the desired table from the Project explorer to the Workspace.

To add all tables and their relationships, right-click the workspace and select Add all tables.

To add a new table, select Table on the Insert tab on the Design ribbon and release it into the Workspace.

You are able to edit the tables' properties by double-clicking on them, which opens a new Table editor tab on the Workspace.

Creating relationships

All existing relationships are automatically shown in a diagram when its related tables are inserted. To create new relationship between two tables the diagram:

1. Click Relationship or Non-ID Relationship on the Insert tab on the Design ribbon.

   o **Non-ID Relationship**: This is the most usual type of relationship. It represents a weak connection, a relationship between parent and child table that does not involve a key field, such as the relationship between tables "Products" and "Categories", for example.

   o **(ID) Relationship**: A relationship where the related fields from the child table are part of the key which identifies a unique record in this table. It indicates a strong connection, such as the relationship between tables "Order details" and "Products", for example. There are no "Order details" without a related "Product", and the "Product" is part of the key of the "Order details" table.

2. Drag a line from the Parent table to the Child table. Upon release, the Add relationship dialog will open.

3. On the Add relationship dialog, fill in with the relationship data:

In the Relationship properties section, you may visualize or set the relationship name and its description, for future reference. A default relationship name is created automatically when adding a new relationship to the project, combining parent and child table names. The relationship name may be edited at any time.

In the Relationship keys section you may visualize or edit the relationship keys by selecting them from the available list.

- **Parent key**: Lists all available keys (primary keys and indexes) from the parent table.

- **Parent table**: This column shows all fields on the parent table which are part of the selected parent key. It is not possible to modify this data.

- **Child table**: This column lists all fields on the child table that are compatible with the selected field on the parent table (same type). By default a new field listed as "field_name(new)" with the same name and type of the parent key is created and selected. It is possible to select any other available child field from the list.

In the Relationship options section, you are able to select the relationship behavior when a record on the parent table is deleted or updated:

- **No action**: no action is taken when there are changes on the parent table. This is the default option.

- **Cascade**: automatically deletes all records on the child table when a parent record is deleted. Example: Customer/Contact, when deleting a customer, all of its contacts are also deleted.

- **Set null**: when a parent record is deleted, all child records related to it get this field set to null. Example: on the Person/Gender deletion, all people related to this record would be set to null.

- **Set default**: similar to "set null" option, but instead of setting the field to null, it is filled with its default value.

4. The relationship will appear on the diagram, connecting both tables. Relationship lines allow easy visualization of different properties on the relationship, such as:

- Identifying relationships are represented by continuous lines, while non-identifying relationships are represented by dotted lines.

- A crow's foot at the end of the line indicates many records, while single line ends represent a single record.

- A red line represents a weakly defined relationship, missing specific fields, for example.

5. The created relationship will be listed under the Relationship branch of the Data dictionary tree. To edit any of the relationship data, double-click on the relationship on the Project explorer and the Relationship editor will open.

To add notes to a diagram

1. Select Note on the Insert tab on the Design ribbon and release the note on the desired point of the Workspace.

2. You may edit the note by right-clicking on it. The context menu will allow you to Edit text, change color or font and remove the note.

Customizing the diagram

By right-clicking on the diagram's background and objects, context menus allow you to customize many aspects of its visualization and organization.
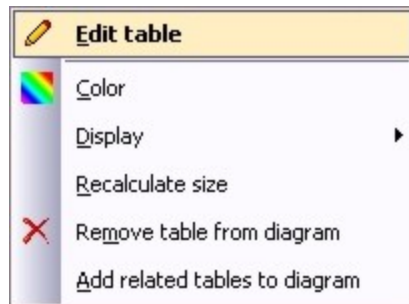
Background context menu
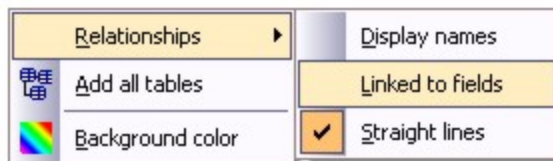
- Background color: changes the Diagram background color.
- Relationships:

- o Display names: shows each relationship's name next to the relationship line/visual connection.
  - o Linked to fields: moves the relationship line/visual connection so each end links to the exact related fields on each table. If there are relationships connecting tables through more than one field, the line then links the first field of the relationship, both for parent and for child tables.
  - o Straight lines: moves the relationship line/visual connection so each line will run straight between the tables, without breaks.
- Add all tables: adds all of the project's tables to the Diagram.

Table context menu

- Edit table: opens the Table editor.
- Color: changes the color on each table's background.
- Display: adjusts the visualization of tables by selecting options on the context menu:
  - o All fields: all fields are displayed (default).
  - o All keys: only fields on the primary key of the table or foreign key fields are displayed.
  - o All primary fields: only fields on the primary key of the table are displayed.
  - o Table name: only the table name is displayed, no field names.
  - o Show field types: shows or hides field types.
  - o Recalculate size: automatically adjusts the displayed table size.
- Remove table from diagram: removes a table from the diagram without deleting it from the project. You can also use the shortcut CTRL+R. By using DEL, you will delete the table from the project.
- Add related tables to diagram: adds to the diagram all tables related to the selected table.

Relationship context menu

- Edit relationship: opens the Relationship editor.
- Color: changes the color on each relationship line.

Note context menu

- Edit text: allows text edition.
- Color: changes the color on a note's background.
- Font: adjusts all font options (face, size etc.)

4.4   Relationship Editor

To open the relationship editor, you should double-click on the desired relationship on the Project explorer. This editor is divided in tree sections: Relationship properties, Relationship keys and Relationship options.

In the **Relationship properties** section, you may visualize or set the relationship name and its description, for future reference. A default relationship name is created automatically when adding a new relationship to the project, combining parent and child table names. The relationship name may be edited at any time.

In the **Relationship keys** section you may visualize or edit the relationship keys by selecting them from the available list.

- **Parent key**: Lists all available keys (primary keys and indexes) from the parent table.
- **Parent table**: This column shows all fields on the parent table which are part of the selected parent key. It is not possible to modify this data.
- **Child table**: This column lists all fields on the child table that are compatible with the selected field on the parent table (same type). By default a new field listed as "field_name(new)" with the same name and type of the parent key is created and selected. It is possible to select any other available child field from the list.

In the **Relationship options** section, you are able to select the relationship behavior when a record on the parent table is deleted or updated:

- **No action**: no action is taken when there are changes on the parent table. This is the default option.
- **Cascade**: automatically deletes all records on the child table when a parent record is deleted. Example: Customer/Contact, when deleting a customer, all of its contacts are also deleted.
- **Set null**: when a parent record is deleted, all child records related to it get this field set to null. Example: on the Person/Gender deletion, all people related to this record would be set to null.
- **Set default**: similar to "set null" option, but instead of setting the field to null, it is filled with its default value.

When opening a relationship by double-clicking on a diagram, the Back to diagram button is enabled, so you can easily return to the original diagram.

4.5   Procedures and Functions

To open the procedure editor, you should double-click on the desired procedure/function on the Project explorer.

Procedures (or "stored procedures"), like triggers, are functions programmed into the database. Through TMS Data modeler, you can edit and create procedures as shortcuts to regularly executed actions, and store them for future use. In this editor, you are able to set:

- Object name: procedure name

- Description.

- Create implementation code: full command to create the stored procedure in the database. When you create a procedure, it automatically implements "CREATE PROCEDURE <%ObjectName%>". This macro "<%...%>" is replaced by the procedure's name, so you can rename it without having to update this implementation.

4.6   Generators/Sequences

Generators/Sequences are supported by some databases like Firebird, Oracle, SQLite, etc.. To open the generator editor, you should double-click on the desired generator on the Project explorer.

Generators are a resource to generate number sequences. You can specify an initial number and the desired increment, and each time a value is selected in this Generator, a new number is generated. To create a new generator, select Object / Generator on the Create tab on the Home ribbon.

All Generators created in the project are listed under Generators section in the Project explorer. By double-clicking on the desired Generator, the Generators editor will open showing:

- Sequence name.

- Start with: initial value of the sequence generator (specified when creating the object).

4.7   Views

Views are queries stored into databases. By creating a view, you are able to select frequently used query processes and save them for future use. To open the views editor, you should double-click on the desired view on the Project explorer.

In this editor, you are able to edit the information below:

- Object name: view name.

- Description.

- Create implementation code: full command to create the view in the database. When you create a view automatically implements "CREATE VIEW <%ObjectName%> AS". This macro "<%...%>" is replaced by the view's name, so you can rename it without having to update this implementation.

Chapter

V
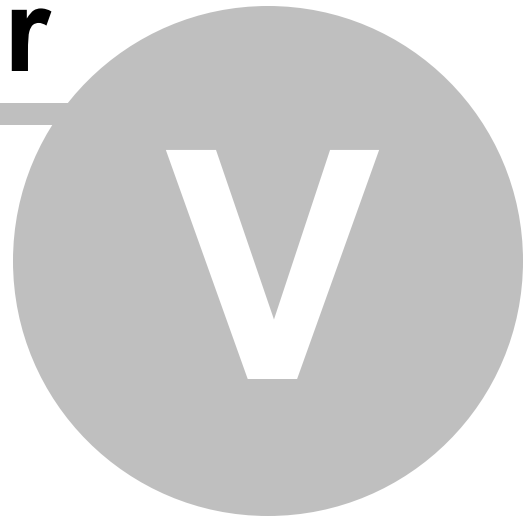
TMS Aurelius Export

5   TMS Aurelius Export

TMS Data Modeler has an option for integration with TMS Aurelius framework by generating classes based on the database structure.

Overview
Export Dialog
Mappings Tab
General Settings Tab
Script Tab
Customization Script

5.1   Overview

TMS Aurelius is a Delphi framework for Object-Relational Mapping (ORM), allowing you to save/load data to database using classes instead of directly using SQL Statements. For it to work, you must create (or use) your Delphi classes and then add attributes to it allowing the framework to know to which table and field each object and property will be saved.

TMS Data Modeler saves you time in the process of creating such classes, by generating the classes automatically based on tables and fields defined in the database. To export the current schema to classes, use the File Menu, option "Export | Delphi (TMS Aurelius)", or the "TMS Aurelius" button in Tools ribbon.
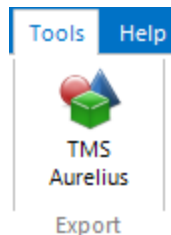
Each table in Data Modeler will become a class. Each table column will become a class field/property. Foreign keys will become associations (properties of an object type). So for example, this Orders table:
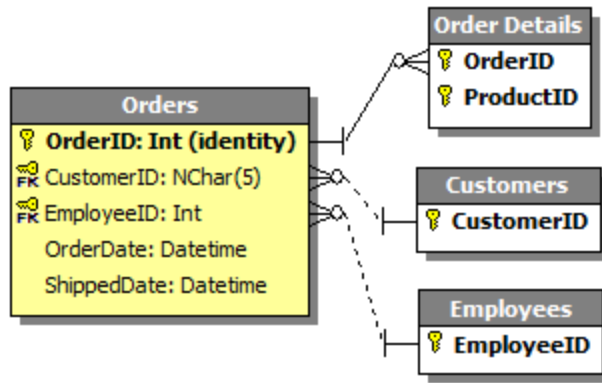
will become this class and mappings:

```
  [Entity]
  [Table('Orders')]
  [Id('FOrderID', TIdGenerator.IdentityOrSequence)]
  TOrder = class
  private
    [Column('OrderID', [TColumnProp.Required,
TColumnProp.NoInsert, TColumnProp.NoUpdate])]
    FOrderID: integer;
    [Column('OrderDate', [])]
    FOrderDate: Nullable<TDateTime>;
    [Column('ShippedDate', [])]
    FShippedDate: Nullable<TDateTime>;
    [Association]
    [JoinColumn('CustomerID', [], 'CustomerID')]
    FCustomer: Proxy<TCustomer>;
    [Association]
    [JoinColumn('EmployeeID', [], 'EmployeeID')]
    FEmployee: Proxy<TEmployee>;
    [ManyValuedAssociation([], [TCascadeType.SaveUpdate,
TCascadeType.Merge], 'FOrderID')]
    FOrderDetailsList: Proxy<TList<TOrderDetails>>;
    function GetCustomer: TCustomer;
    procedure SetCustomerID(const Value: TCustomers);
    function GetEmployee: TEmployee;
    procedure SetEmployee(const Value: TEmployee);
    function GetOrderDetailsList: TList<TOrderDetails>;
  public
    constructor Create;
    destructor Destroy; override;
    property OrderID: integer read FOrderID;
    property OrderDate: Nullable<TDateTime> read FOrderDate
write FOrderDate;
    property ShippedDate: Nullable<TDateTime> read FShippedDate
write FShippedDate;
    property Customer: TCustomer read GetCustomer write
SetCustomer;
    property Employee: TEmployee read GetEmployee write
SetEmployee;
    property OrderDetailsList: TList<TOrderDetails> read
GetOrderDetailsList;
  end;
```

5.2   Export Dialog

In the TMS Aurelius Export Dialog you have several settings to configure how the export will be performed.

The main option - and the only one you need to specify explicitly - is the **output directory**. This is where your units will be generated.

For all the other settings, you have a specific tab with several options you can configure:
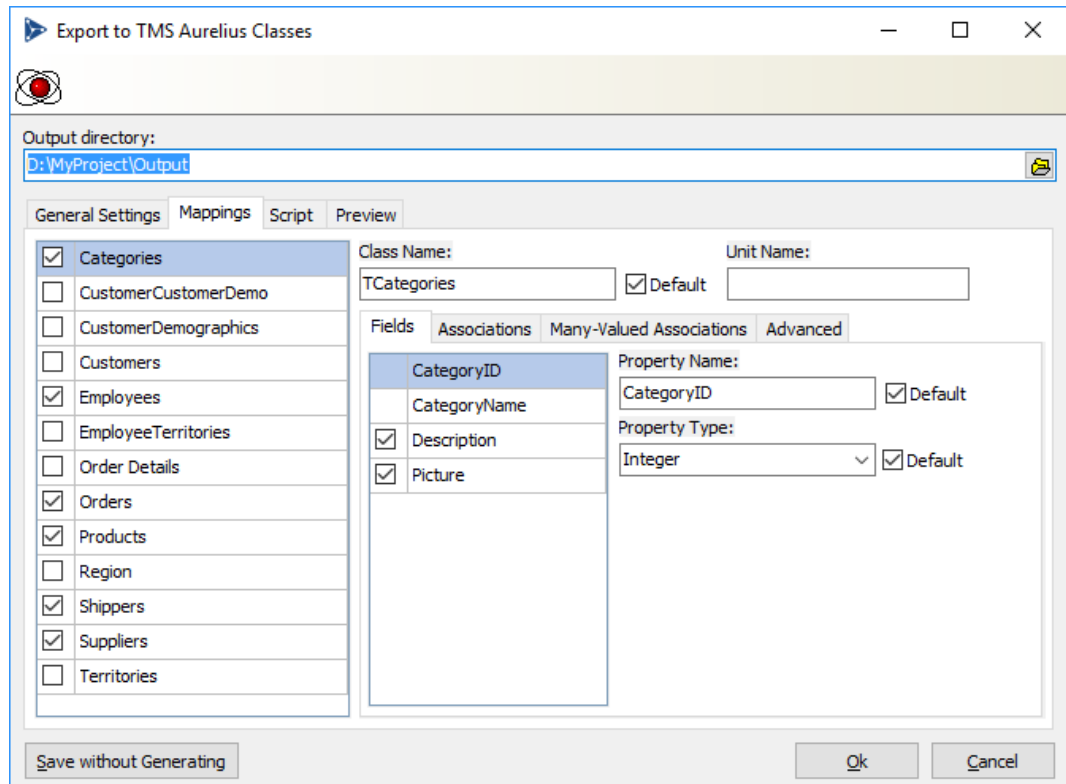
Mappings Tab
General Settings Tab
Script Tab

Finally, in the Preview Tab you have can see all the source code with unit names that will be generated in the output directory. This tab is useful to browse and check the source code before you actually generate it (and eventually replace existing files).

5.3　Mappings Tab

When exporting to TMS Aurelius, you are presented with a dialog with several options for configuring how the classes will be generated. Clicking "Ok" will update the export options in the project and generate the source files. The "Save without generating" button allows you to update the options in the project but with no generating source code - to avoid error messages, if any. Note that you must later save the Data Modeler project to effectively save the options to project file.

The Mappings tab contains a list of all classes, properties, associations and many-valued associations that will be generated by Data Modeler. It allows you to fine tune your exporting, by checking each individual class/property and overriding some default values. This tab is pre-filled with default values and all tables are selected for exporting - you don't need to change any settings here if you don't need to, it will export all tables with default settings.

The first list displayed at the left of Mappings tab is a list of available tables/classes that will be generated. When you select an item, the right part of the window is updated to reflect the settings for the currently selected table. By default all tables are selected (meaning all classes will be generated). You can uncheck an item to avoid that class to be generated.

You can right click the list of selected tables for options to select all, unselect all, and other operations in the list.

You can also perform a search in the list by pressing **Ctrl+Shift+F.** This will open a search box at the bottom of the list:
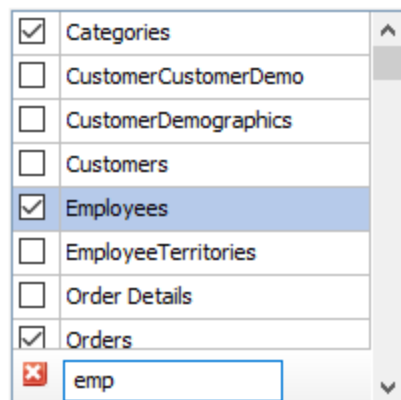
For each selected table in the list at the left, you have the following options you can configure:

Field	Description
Class Name	Defines the name of class to be generated. If Default is checked it will use the default naming rule for the class.
Unit Name	Specifies the name of the unit file where this class will be included. If empty (default) the class will be included in the main default unit, specified in the General Settings tab. You can then specify a different unit name here. TMS Data Modeler will automatically add needed uses clause in the unit (for example, if your class uses other classes defined in other units). However, you have to pay attention to cyclical references - for example, if class A references class B and class B references A, and you put each of those classes in different units, the resulting code will not compile. TMS Data Modeler will automatically inform you that you have cyclical references, but it's up to you to solve them by filling the correct unit names in this field. By default all classes are generated in a single unit, which will never cause cyclical references errors.

Fields Tab

Lists all fields/properties that will be generated for the selected class. You can uncheck items if you don't want them to be exported. Required fields cannot be unchecked. For each selected field you can configure the following options:

Field	Description
Property Name	Defines the name of the property to be generated. Class field names will have the same name of the property but will be prefixed with "F". If Default is checked it will use the default naming rule for properties.
Property Type	Defines the type of the property to be generated ("Integer", "Nullable<double>", etc.). The default value is automatically defined from the table column type. If Default is checked it will use the default type (based on column type). If unchecked, you must type the full name you want. It can be any type name, including Nullables.

Associations Tab

Lists all properties that will be generated as associations. You can uncheck items if you don't want them to be exported. For each selected association you can configure the following options:

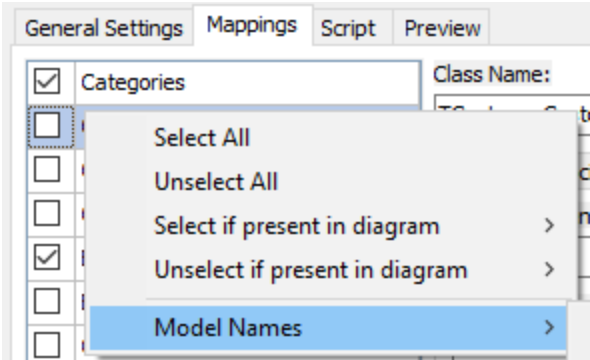| Field	Description
Association Property Name	Defines the name of property to be generated for the association. If Default is checked, the default naming rule specified in General Settings tab will be used.
Fetch Mode	Specifies how the association will be fetched (lazy, eager or default). If Default is selected, it will use the value specified in the General Settings tab.
Cascade	Specifies the cascade type to be used in association. Options are "Default", "None" (no cascade) and "All but Remove" (all cascade options like save, update, merge, except remove cascade).
Map this 1:1 relationship as	Visible for identity relationships. Specifies how the 1:1 association will be exported, if as association or treat as inheritance. If Default is selected, it will use the value specified in the General Settings tab.

Many-Valued Associations Tab

Lists all properties that will be generated as many-valued associations (collections). You can check items if you want them to be exported. By default collections are unchecked. For each many-valued association that can be generated you can configure the following options:

Field	Description
List Property Name	Defines the name of collection property to be generated for the many-valued association. If Default is checked, the default naming rule specified in General Settings tab will be used.
Fetch Mode	Specifies how the association will be fetched (lazy, eager or default). If default is specified, it will use the value specified in the General Settings tab.

Advanced Tab

Lists several other settings for the table/class being exported.

Field	Description
Sequence/Generator for ID	Specifies the name of the sequence to be generated as the Sequence attribute (Sequence['Sequence_Name']).

	Not specifying a value here might cause Data Modeler to raise an error when exporting, depending on the value of "Check for missing sequences" option in General Settings tab. If you don't want to specify a sequence and also don't want any error to be raised regardless of "Check for Missing Sequences" option, choose "(none)".
Dynamic Props Container Name	Specifies the name for the property that will be a container for dynamic properties. If empty, then by default no dynamic property container will be created in the class. If Default is checked, it will use the default name specified in the "Defaults" section in main tab.

Models (Comma Separated)	The Aurelius models where this class will belong to. This relates to multimodel design in Aurelius. For each model specified here, an attribute
	
	`[Model(`'ModelName'`)]`
	
	will be added to the class. If you want to specify more than one model, just separate the model names with commas. For example, "Default,Finance,Sales" will generate the following attributes:
	
	`[Model(`'Default'`)]`
	`[Model(`'Finance'`)]`
	`[Model(`'Sales'`)]`
	
	You can have this field to be automatically filled by right clicking the list of classes at the left and choosing "Model Names" menu option.
	
	You will have the following options:
	
	• Update From Diagrams: For each diagram containing the specified class, a model will be added with the same name of the diagram.
	• Update From Diagrams (include Default): Same as "Update from Diagrams", but "Default" model will also be added.
	• Clear All: Remove the models from all classes.

	Note that the operations above will be performed for **all tables checked** in the list, not for the currently selected only.

5.4   General Settings Tab

The General Settings tab contains some settings that apply for the exporting process as a whole, different from the Mapping tab that configures each class specifically. Here you will also configure some default behavior that apply to all classes.
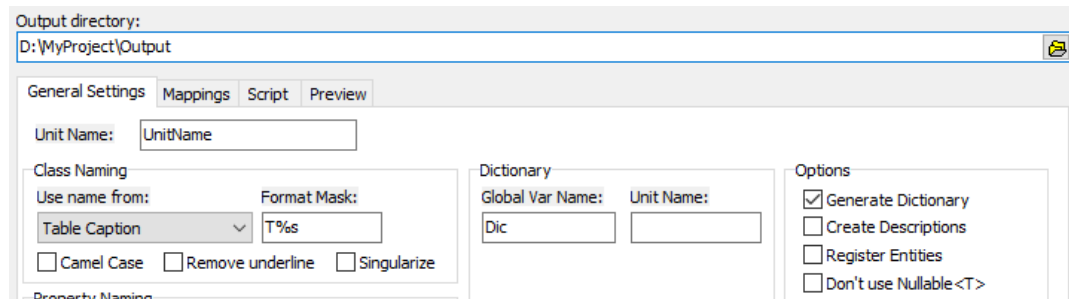
Naming options

You can define the default rule for naming classes, property/fields, associations and many-valued associations. Even though you can configure the name of each specific class, property, etc. in the Mappings tab, using this global configuration allows you to apply a default naming rule for all of them so you don't need to manually name them one by one.

Basically you have the "Use name from" fields which specifies what will be used for the "base name". For example, for class naming, you can use the base name from Table Caption in the model, or Table Name. From the base name, the Format Mask will be applied. The "%s" in the format mask will be replaced by the base name. For example, the defualt Format Mask for class naming is "T%s" which means the class name will be the base name (usually Table Caption) prefixed with "T".

Additionally, some naming options allow you to:

- Camel Case: The first character of the base name or any character followed by underling will become upper case, all the other will become lower case. For example, if the base name in model is "SOME_NAME", it will become Some_Name.
- Remove underline: All underlines will be removed. "SOME_NAME" becomes "SOMENAME". If combined with camel case, it will become "SomeName"
- Singularize: If the base name is in plural, it will become singular. "Customers" become "Customer", "Orders" become "Order". It also applies specified

singularization rules for English language (e.g., "People" becomes "Person", etc.).

Dictionary

Data Modeler can also generate a dictionary with metadata for the classes. This dictionary can be used in queries in TMS Aurelius.

- Global Var Name: Defines the name of Delphi global variable to be used to access the dictionary.
- Unit Name: Defines the unit name/file name where the dictionary will be created. If empty (default), the dictionary will be created in the same file/unit as the classes (specified in the "Unit Name" field)

Defaults

Defines some default behaviors when translating tables/fields into classes/properties. You can override this default behaviors individually for each class/property in the "Mappings" tab.

Field	Description
Association Fetch Mode	The default fetch mode used for associations. Default value is Lazy.
Association Cascade Definition	The default cascade definition for associations. Options are "None" (no cascade) and "All but Remove" (all cascade options like save, update, merge, except remove cascade). Default value is None.
Many-Valued Association Fetch Mode	The default fetch mode used for many-valued associations. Default is Lazy.
Map One-to-One Relationship As	Defines how 1:1 relationships will be converted by default. A 1:1 relationship can be converted as a regular association (property) or can be considered an inheritance between two classes. Default value is Association.
Ancestor Class	Specifies the name of the class to be used as base class for all entity classes generated by Data Modeler. Default value is empty, which means no ancestor (all classes will descend from TObject)
Dynamic Props Container Name	Specifies the default name for the property that will be a container for dynamic properties. If empty, then by default no property will be created in the class.
Check for Missing Sequences	Defines if Data Modeler must stop exporting (raise an error) if a sequence is not defined for a class. Options are:

|	• If supported by database: if database supports sequences/generators, then raise an error if a sequence is not defined (default) • Always: always raise an error if a sequence is not specified • Never: ignore any sequence check

Options

Defines some other general options for exporting.

Field	Description
Generate Dictionary	Defines if the dictionary will be generated.
Create Descriptions	If checked, an attribute [Description] will be generated for every class and field in the source code. The content of such attribute will come from the tables and fields descriptions in Data Modeler. This attribute is not used by TMS Aurelius and is useful only for the developer if he wants to retrieve such metadata at runtime.

Register Entities	When checked, the generated unit will have an initialization section with a call to RegisterEntity for each class declared in the script:
	
	``pascal
	initialization
	  RegisterEntity(TSomeClass);
	  RegisterEntity(TAnotherClass);
	``
	
	This will make sure that when using the generated unit, classes will not be removed from the final executable because they were not being used in the application. This option is useful when using the entity classes from a TMS XData server, for example.

Don't use Nullable<T>	By default, non-required columns will be generated as properties of type Nullable<T>. Check this option if you don't want to use Nullable, but instead use the primitive type directly (string, integer, etc.)

5.5   Script Tab

In the Script tab you can write the customization script that manipulates the source code generated by the TMS Aurelius Export feature.
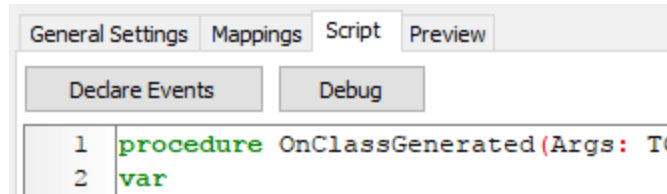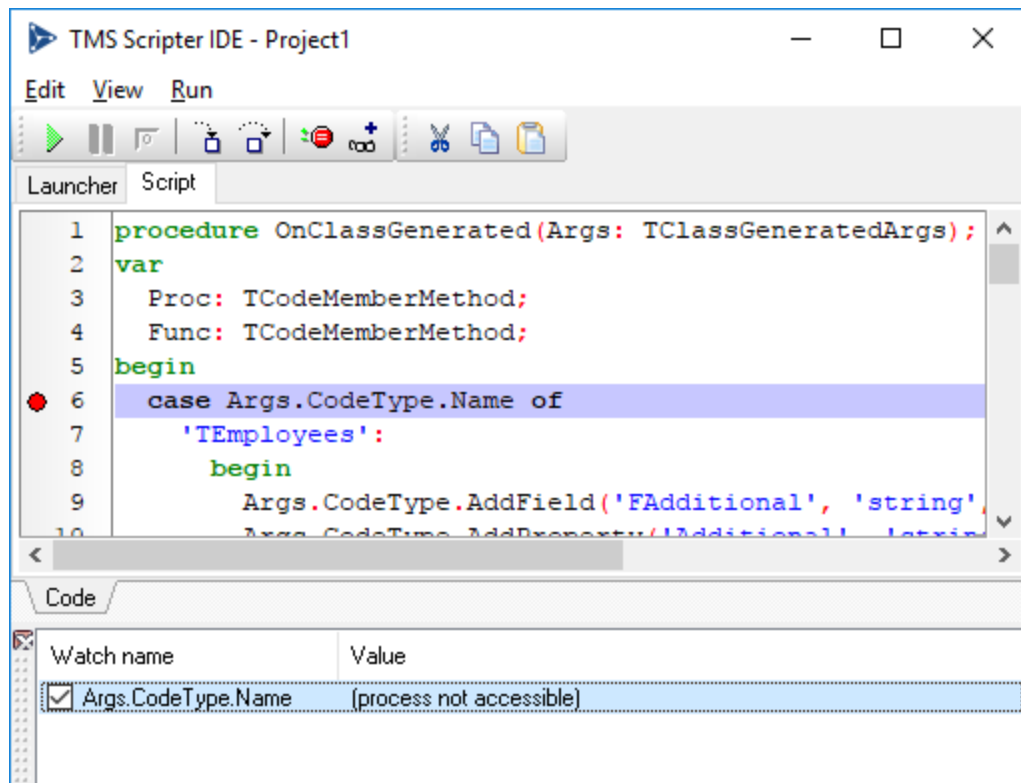
Just type the customization script in the editor. For more information about how to write the script itself, please refer to the Customization Script topic. If you want an initial help, you can click in the Declare Events button to have all the supported script events to be declared automatically in script.

You can also debug the script if needed, by clicking the Debug button. This will open a full script IDE window already prepared with the existing customization script, and you can set breakpoints, add watches, run the script, etc..

The IDE has two scripts available: Launcher and Script. The Launcher is just the main script used to run the source code generation process that makes the event handlers to be fired. The customization script is in the Script tab.

Any changes you make to the Script tab will later be updated to the customization script in the Script Tab, after you close the Debug IDE. You should never change the Launcher script - it's generated dynamically and any changes you make to it will be discarded.

5.6   Customization Script

You can fully customize the generated source code using the customization script, which you can write and debug from the Script Tab in the export dialog. Form information on how to edit and debug the script, please follow the Script Tab topic. This topic covers what code you can write from script and provide some code examples.

Event Handlers

The script is based on the concept of event handlers. When Data Modeler is creating the meta-information of the code, it can fire many events after each piece of code being generated. For example, when a database column is processed and a combination of class field/property is created, the event OnColumnGenerated is created. To handle such event, just declare a procedure in the script code with the same name, receiving a single param of type TColumnGeneratedArgs. From that parameter you can retrieve relevant context information and change/adapt the generate code for your own need.

Here is a list of the events fired by the source code generator:

Event	Description
OnColumnGenerated	Fired whenever a table column in processed and becomes a class field and property of a primitive type, and Column attribute is added to the class field.
OnAssociationGenerated	Fired whenever an association is processed, i.e., when a database foreign key becomes a class field and property of type Proxy<T> and T, respectively, and attributes Association and JoinColumn are added to the class field.
OnManyValuedAssociationGenerated	Fired whenever a many-valued association is processed and becomes a class field/property of type TList<T> and ManyValuedAssociation attribute is added.
OnClassGenerated	Fired after a class (entity) type has been fully generated.
OnUnitGenerated	Fired after the unit has been fully generated.

Code Samples

Some script samples for common customization tasks:
Adding OrderBy Attribute to Many-Valued Association
Adding Version Attribute to Class Fields
Creating a New Property in a Class
Creating a New Method Procedure in a Class
Creating a New Method Function in a Class
Adding a Unit Name to the Uses Clases
Changing Cascade of Many-Valued Association

5.6.1   OnColumnGenerated Event

This event is fired whenever a table column is processed and becomes a class field and property of a primitive type, and Column attribute is added to the class field. Should be declared as following:

```
procedure OnColumnGenerated(Args: TColumnGeneratedArgs);
begin
end;
```

Type TColumnGeneratedArgs and has the following properties:

Property	Description
Prop: TCodeMemberProperty	The public property of type TList<T>.
Field: TCodeMemberField	The private field of type Proxy<TList<T>>.
CodeUnit: TCodeUnit	The Pascal unit where this class is declared.
CodeType: TCodeTypeDeclaration	The type declaration of the class containing the property/field
Getter: TCodeMemberMethod	The private getter method used as the reader of the public property.
AssociationAttr: TCodeAttributeDeclaration	The [Association] custom attribute added to the private field.
ConstructorMethod: TCodeMemberConstructor	The constructor Create method of the class, used to add the TList<T>.Create statement that instantiates the list object.
DestructorMethod: TCodeMemberDestructor	The destructor Destroy method of the class, used to add the TList<T>.Destroy statement that destroys the list object.
DBField: TGDAOField	Metadata for the table column in database.

5.6.2 OnAssociationGenerated Event

This event is fired whenever an association is processed. This means a foreign key in the database becomes a class field of type Proxy<T> and a property of type T where T is an object, and attributes Association and JoinColumn are added to the class field. Should be declared as following:

```
procedure OnAssociationGenerated(Args:
TAssociationGeneratedArgs);
begin
end;
```

Type TAssociationGeneratedArgs has the following roperties.

Property	Description
Prop: TCodeMemberProperty	The public property of type T.
Field: TCodeMemberField	The private field of type Proxy<T>.
CodeUnit: TCodeUnit	The Pascal unit where this class is declared.
CodeType: TCodeTypeDeclaration	The type declaration of the class containing the property/field
Getter: TCodeMemberMethod	The private getter method used as the reader of the public property. It might be nil if property is not lazy-loaded.
Setter: TCodeMemberMethod	The private setter method used as the writer of the public property. It might be nil if property is not lay-loaded.
AssociationAttr: TCodeAttributeDeclaration	The [Association] custom attribute added to the private field.
DBRelationship: TGDAORelationship	Metadata for the database relationship (foreign key) in database.

5.6.3 OnManyValuedAssociationGenerated Event

This event is fired whenever a many-valued association is processed. This means a foreign key in the database becomes a class field and property of type TList<T> and attributes ManyValuedAssociation and ForeignJoinColumn are added to the class field. Should be declared as following:

```pascal
procedure OnManyValuedAssociationGenerated(Args:
TManyValuedAssociationGeneratedArgs);
begin
end;
```

Args is of type TManyValuedAssociationGeneratedArgs and has the following properties:

Property	Description
Prop: TCodeMemberProperty	The public property of type TList<T>.
Field: TCodeMemberField	The private field of type Proxy<TList<T>>.
CodeUnit: TCodeUnit	The Pascal unit where this class is declared.
CodeType: TCodeTypeDeclaration	The type declaration of the class containing the property/field
Getter: TCodeMemberMethod	The private getter method used as the reader of the public property.
AssociationAttr: TCodeAttributeDeclaration	The [ManyValuedAssociation] custom attribute added to the private field.
ConstructorMethod: TCodeMemberConstructor	The constructor Create method of the class, used to add the TList<T>.Create statement that instantiates the list object.
DestructorMethod: TCodeMemberDestructor	The destructor Destroy method of the class, used to add the TList<T>.Destroy statement that destroys the list object.
DBRelationship: TGDAORelationship	Metadata for the database relationship (foreign key) in database.

5.6.4   OnClassGenerated Event

This event is fired whenever a class (entity) type is fully generated. You can use this event to do some customization to the class after all its properties are added. Should be declared as following:

```
procedure OnClassGenerated(Args: TClassGeneratedArgs);
begin
end;
```

Type TClassGeneratedArgs and has the following properties:

Property	Description
CodeUnit: TCodeUnit	The Pascal unit where this class is declared.
CodeType: TCodeTypeDeclaration	The type declaration of the class containing the property/field
DBTable: TGDAOTable	Metadata for the table in database.

5.6.5   OnUnitGeneratedEvent

This event is fired whenever an unit is fully generated. You can use this event to do some customization to the unit after it has been generated, like adding an unit to the uses clause, add code to initialization section, etc.. Should be declared as following:

```
procedure OnUnitGenerated(Args: TUnitGeneratedArgs);
begin
end;
```

Type TUnitGeneratedArgs and has the following properties:

Property	Description
CodeUnit: TCodeUnit	The Pascal unit that has been generated.

5.6.6   Adding OrderBy Attribute to Many-Valued Association

This example adds the attribute [OrderBy] to the many-valued association. It identifies the many-valued association by the name of the generated private field (FEmployeesList) in a specific class (TEmployees):

```
procedure OnManyValuedAssociationGenerated(Args:
TManyValuedAssociationGeneratedArgs);
begin
  case Args.CodeType.Name of
    'TEmployees':
        case Args.Field.Name of
          'FEmployeesList':
            Args.Field.AddAttribute('OrderBy').AddRawArgument('''
LAST_NAME,FIRST_NAME''');
        end;
    end;
end;
```

Suppose the original generated source code was this one:

```
    [ManyValuedAssociation([TAssociationProp.Lazy],
[TCascadeType.SaveUpdate, TCascadeType.Merge], 'FReportsTo')]
    FEmployeesList: Proxy<TList<TEmployees>>;
```

With the script above, it will become this:

```
    [ManyValuedAssociation([TAssociationProp.Lazy],
[TCascadeType.SaveUpdate, TCascadeType.Merge], 'FReportsTo')]
    [OrderBy('LAST_NAME,FIRST_NAME')]
    FEmployeesList: Proxy<TList<TEmployees>>;
```

5.6.7 Adding Version Attribute to Class Fields

This example adds the attribute [Version] to any class field named "FVersion", regardless of the class it appears.

```
procedure OnColumnGenerated(Args: TColumnGeneratedArgs);
begin
  if Args.Field.Name = 'FVersion' then
    Args.Field.AddAttribute('Version');
end;
```

Suppose the original generated source code was this one:

```
    [Column('VERSION', [TColumnProp.Required])]
    FVersion: Integer;
```

With the script above, it will become this:

```
    [Column('VERSION', [TColumnProp.Required])]
    [Version]
    FVersion: Integer;
```

5.6.8 Creating a New Property in a Class

This example creates a private string field FAdditional and public property Additional (which getter and setter refers to private field) to the class TEmployees:

```
procedure OnClassGenerated(Args: TClassGeneratedArgs);
begin
  case Args.CodeType.Name of
    'TEmployees':
      begin
        Args.CodeType.AddField('FAdditional', 'string',
mvPrivate);
        Args.CodeType.AddProperty('Additional', 'string',
          'FAdditional', 'FAdditional', mvPublic);
      end;
  end;
end;
```

This will create the following private field and public property:

```
  private
    FAdditional: string;
  public
    property Additional: string read FAdditional write
FAdditional;
```

5.6.9 Creating a New Method Procedure in a Class

This example creates a public method Increase in class TCategories:

```
procedure OnClassGenerated(Args: TClassGeneratedArgs);
var
  Proc: TCodeMemberMethod;
begin
  case Args.CodeType.Name of
    'TCategories':
      begin
        Proc := Args.CodeType.AddProcedure('Increase',
mvPublic);
        Proc.AddParameter('Value', 'Integer').Modifier := pmVar;
        Proc.AddParameter('Increment', 'Integer');
        Proc.AddSnippet('Value := Value + Increment;');
      end;
  end;
end;
```

It will create the following method declaration and implementation:

```
    procedure Increase(var Value: Integer; Increment: Integer);
```

```
procedure TCategories.Increase(var Value: Integer; Increment:
Integer);
begin
  Value := Value + Increment;
end;
```

5.6.10   Creating a New Method Function in a Class

This example creates a public method Triple in class TEmployees:

```
procedure OnClassGenerated(Args: TClassGeneratedArgs);
var
  Func: TCodeMemberMethod;
begin
  case Args.CodeType.Name of
    'TEmployees':
      begin
        Func := Args.CodeType.AddFunction('Triple', 'double',
mvPublic);
        Func.AddParameter('Value', 'double');
        Func.AddSnippet('Result := Value * 3;');
      end;
  end;
end;
```

It will create the following method declaration and implementation:

```
    function Triple(Value: double): double;
```

```
function TEmployees.Triple(Value: double): double;
begin
  Result := Value * 3;
end;
```

5.6.11   Adding a Unit Name to the Uses Clases

This example adds a unit name "MyUnit" to the uses causes (interface section) of the unit "UnitName":

```
procedure OnUnitGenerated(Args: TUnitGeneratedArgs);
begin
  if Args.CodeUnit.Name = 'UnitName' then
    Args.CodeUnit.InterfaceUnits.Add(TCodeUsedUnit.Create('MyUni
t'));
end;
```

It will add the following uses clause do the unit:

```
unit UnitName;

interface

uses
  SysUtils,
  Generics.Collections,
  Aurelius.Mapping.Attributes,
  Aurelius.Types.Blob,
  Aurelius.Types.DynamicProperties,
  Aurelius.Types.Nullable,
  Aurelius.Types.Proxy,
  MyUnit,
  Aurelius.Criteria.Dictionary;
```

5.6.12 Changing Cascade of Many-Valued Association

This example changes the cascade parameter of many-valued association attribute. It identifies the many-valued association by the name of the generated private field (FEmployeesList) in a specific class (TEmployees):

```
procedure OnManyValuedAssociationGenerated(Args:
TManyValuedAssociationGeneratedArgs);
begin
  case Args.CodeType.Name of
    'TEmployees':
      case Args.Field.Name of
        'FEmployeesList':
            TCodeSnippetExpression(Args.AssociationAttr.Argumen
ts[1].Value).Value := 'CascadeTypeRemoveOrphan';
      end;
  end;
end;
```

Suppose the original generated source code was this one:

```
    [ManyValuedAssociation([TAssociationProp.Lazy],
[TCascadeType.SaveUpdate, TCascadeType.Merge], 'FReportsTo')]
    FEmployeesList: Proxy<TList<TEmployees>>;
```

With the script above, it will become this:

```
    [ManyValuedAssociation([TAssociationProp.Lazy],
CascadeTypeRemoveOrphan, 'FReportsTo')]
    FEmployeesList: Proxy<TList<TEmployees>>;
```

5.6.13 Adding ForeignKey Attribute to Associations

By default Aurelius defines the name of foreign keys automatically. You can force a name for the foreign key using ForeignKey attribute. Since Data Modeler already holds the name of all foreign keys (relationships) in the

database, you can include a ForeignKey attribute to force all foreign keys to have the existing name in database.

The following example does that:

```
procedure OnAssociationGenerated(Args:
TAssociationGeneratedArgs);
begin
  Args.Field.AddAttribute('ForeignKey').AddRawArgument(
    '''' + Args.DBRelationship.RelationshipName + '''');

end;
```

Suppose the original generated source code was this one:

```
    [Association([TAssociationProp.Lazy,
TAssociationProp.Required], [])]
    [JoinColumn('ProductID', [TColumnProp.Required],
'ProductID')]    [ForeignKey('FK_Order_Details_Products')]
    FProduct: Proxy<TProduct>;
```

With the script above, it will become this:

```
    [Association([TAssociationProp.Lazy,
TAssociationProp.Required], [])]
    [JoinColumn('ProductID', [TColumnProp.Required],
'ProductID')]
    [ForeignKey('FK_Order_Details_Products')]
    FProduct: Proxy<TProduct>;
```

5.6.14 Adding DBIndex Attributes From Table Indexes

TMS Data Modeler doesn't generate DBIndex attributes from existing table indexes. But if you need it, you can do it using the following code snippet:

```
procedure OnClassGenerated(Args: TClassGeneratedArgs);
var
  I, J: Integer;
  Idx: TGDAOIndex;
  Fields: string;
  Attr: TCodeAttributeDeclaration;
begin
  for I := 0 to Args.DBTable.Indexes.Count - 1 do
  begin
    Idx := Args.DBTable.Indexes[I];
    Fields := '';
    for J := 0 to Idx.IFields.Count - 1 do
    begin
      if Fields <> '' then Fields := Fields + ',';
      Fields := Fields + Idx.IFields[J].FieldName;
    end;
    Attr := Args.CodeType.AddAttribute('DBIndex');
    Attr.AddRawArgument('''' + Idx.IndexName + '''');
    Attr.AddRawArgument('''' + Fields + '''');
  end;
end;
```

For example, originally the class would be generated like this:

```
[Entity]
[Table('Categories')]
[Id('FCategoryID', TIdGenerator.IdentityOrSequence)]
TCategories = class
```

With the following script a DBIndex attribute will be added for each table index:

```
[Entity]
[Table('Categories')]
[Id('FCategoryID', TIdGenerator.IdentityOrSequence)]
[DBIndex('CategoryName', 'CategoryName')]
TCategories = class
```

5.6.15 Adding Schema Name to Table Attribute

Data Modeler doesn't hold schema information for each table, but when exporting to TMS Aurelius classes you can use customization script to add the schema name to the table attribute. The following script adds the schema name *dbo* to the table *category*:

```
procedure OnClassGenerated(Args: TClassGeneratedArgs);
begin
  if Args.DBTable.TableName = 'category' then
    Args.TableAttr.AddRawArgument('''dbo''');
end;
```

For example, originally the class would be generated like this:

```
[Entity]
[Table('Category')]
[Id('FCategoryID', TIdGenerator.IdentityOrSequence)]
TCategories = class
```

With the following script the Table attribute will be like this:

```
[Entity]
[Table('Category', 'dbo')]
[Id('FCategoryID', TIdGenerator.IdentityOrSequence)]
TCategories = class
```