Matisse® Data Transformation Services

January 2017

Matisse Data Transformation Services

PDF generated 7 January 2017

Contents

1 Introduction

1.1 Scope of this document

This document is intended to help Matisse Database developers and administrators learn the command lines utilities and programming interfaces that can be used to extract, transform and load data from a relational source into a Matisse database.

This document also covers the migration process for converting relational data into a richer Matisse data model. The regeneration of the semantic links between data elements, that have been lost between the schema Entity-Relationship diagramming and the relational denormalization process, is also described in great details.

If there is anything you would like to see added, or if you have any questions about or corrections to this document, please e-mail us at support@matisse.com.

1.2 Before Reading this document

Throughout this document, we presume that you know the basics of defining a Matisse schema and that you are familiar with the manipulation of Comma Separated Values (CSV) files.

2 Migrating Relational Data into Matisse

To streamline the transition between a relational database and Matisse, we describe a simple 3-step data migration process.

2.1 Step 1: Exporting from a Relational Database

This first step consists in exporting the application schema from your relational database as well as exporting the table contents into CSV format files.

In most cases, exporting your application database schema consists in generating a SQL DDL script. This script file should include all the database objects (tables, index, stored procedures, etc.) that you want to migrate into Matisse.

Then, to export all the data, you will in most cases export each table into a single CSV file. You may also split the table content into multiple CSV files, a typical example would be if you decide to convert a single table into a class hierarchy.

2.2 Step 2: Importing into Matisse

The second step consists in 'converting' the SQL DDL script that you exported from your relational application, loading it into your Matisse database to create your schema, and then loading the CSV files into Matisse.

To convert between a relational SQL DDL script and a Matisse SQL DDL script, you will verify that the datatypes are supported by Matisse, and if not substitute to the closest match. This task is similar to migrating between different relational products as all implementations have variations on data types like Timestamp or Blobs types. The correspondence between data types is described in *Chapter 9 Data Types Conversion*.

The SQL DDL script can be loaded into Matisse through the Matisse Enterprise Manager or by running the following command:

```
> mt_sdl -d <db> import --ddl -f <ddl_script.sql>
```

The CSV files can be loaded into Matisse through the Matisse Enterprise Manager or by running the following command for each file:

```
> mt_dts -d <db> import -f <file.csv> -c <class_name>
```

2.3 Step 3: Establishing Links between Entities

To truly benefit from Matisse modeling capabilities, you need to recreate the semantic links between entities that existed in your original E-R diagram. All the relationships between tables, that have been translated into primary-key foreign-key constraints through the normalization process, need to be recreated.

Consider the case of the associations IsServedBy/IsInChargeOf between the Presidency and Person classes, which are described in the relational model by the Primary-key/Foreign-key Pid/Pid.

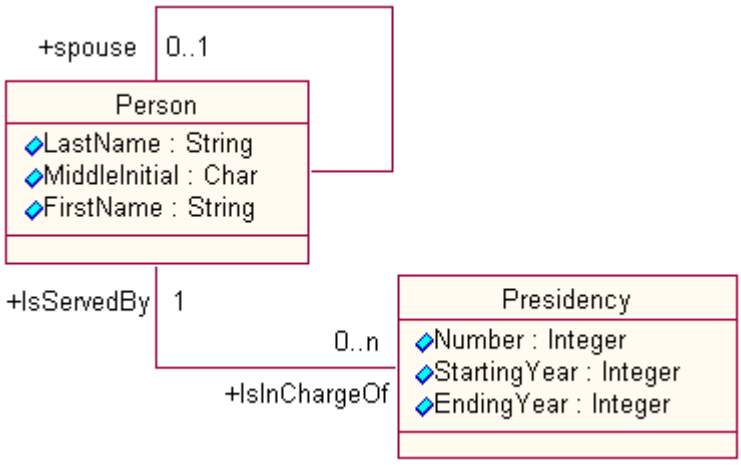

Figure 2.3.1 Database Schema in Matisse

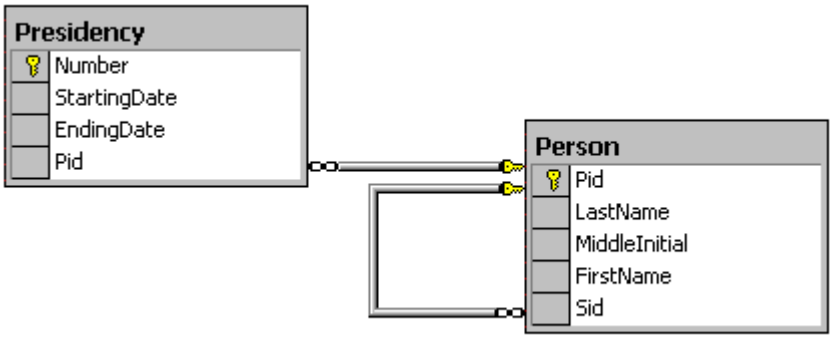

Figure 2.3.2 Original Relational Schema

The links of this one-to-many association can be recreated by going though each instance of Presidency to associate the appropriate instance of Person.

Matisse DTS services provide a feature to reestablish the links between entities. The links to be established are described in an XML Relationship Definition (XRD) file. For example, the relationship to be reestablished between Presidency and Person is described as follows:

```
<?xml version="1.0"?>
<DTSlinks>
```

```
  <OneToMany>
    <PrimaryKey Class="Person"
                Relationship="IsInchargeOf"
               DeleteAfter="True">Pid</PrimaryKey>
    <ForeignKey Class="Presidency"
                Relationship="IsServedBy"
                DeleteAfter="True">Pid</ForeignKey>
  </OneToMany>
  <OneToMany>
    <PrimaryKey Class="Person"
                Relationship="Spouse"
               DeleteAfter="True">Pid</PrimaryKey>
    <ForeignKey Class="Person"
                Relationship="Spouse"
                DeleteAfter="True">Sid</ForeignKey>
  </OneToMany>
</DTSlinks>
```

You must first create the missing relationships in your schema. For instance, executing the following SQL DDL statements will create the relationships that are described in this XRD example:

```
ALTER CLASS Person ADD RELATIONSHIP Spouse
    REFERENCES (Person)
    CARDINALITY (0, 1)
    INVERSE Person.Spouse;

ALTER CLASS Person ADD RELATIONSHIP IsInChargeOf
    REFERENCES SET (Presidency)
    CARDINALITY (0, -1)
    INVERSE Presidency.IsServedBy;

ALTER CLASS Presidency ADD RELATIONSHIP IsServedBy
    REFERENCES (Person)
    CARDINALITY (0, 1)
    INVERSE Person.IsInChargeOf;
```

The XRD file can then be executed from the Matisse Enterprise Manager or by running the following command:

```
> mt_dts -d <db> link -f <file.xrd>
```

You can also reestablish a relationship between entities where the primary key and foreign key are composed of multiple columns. The primary/foreign keys can hold up to 4 columns separated by a comma (,). For example, the relationship to be reestablished between Account and Person is described as follows:

```
<?xml version="1.0"?>
<DTSlinks>
  <OneToMany>
    <PrimaryKey Class="Account"
                Relationship="Onwer"
               DeleteAfter="True">BankId,AccountNumber</PrimaryKey>
    <ForeignKey Class="Person"
                Relationship="OnwedBy"
                DeleteAfter="True">BankId,AccountNumber</ForeignKey>
  </OneToMany>
</DTSlinks>
```

3 Consolidating Matisse Data into a Legacy System

3.1 Step 1: Exporting Data from Matisse

While Matisse provides you with a richer data model, multi-dimensional data stored into Matisse can be easily flattened to fit into a table format.

Flatten a class hierarchy

Consider the case of the ProjectMember and ProjectManager hierarchy defined in Matisse as follows:

```
CREATE CLASS ProjectMember (
    EmpId           INT,
    FirstName       VARCHAR(32),
    LastName        VARCHAR(32),
    Rate            NUMERIC(19,2)
) ;
CREATE CLASS ProjectManager UNDER ProjectMember (
    SignatureLevel  INT,
    BudgetAuthority INT
) ;
```

We assume that it is implemented in the relational database as follows:

```
CREATE TABLE ProjectMembers (
    EmpId           INT,
    EmpType         VARCHAR(32),
    FirstName       VARCHAR(32),
    LastName        VARCHAR(32),
    Rate            NUMERIC(19,2),
    SignatureLevel  INT,
    BudgetAuthority INT
) ;
```

The following two queries are generating data that matches the relational model:

```
SELECT EmpId, CLASS_NAME AS EmpType , FirstName, LastName, Rate,
    SignatureLevel, BudgetAuthority FROM ProjectManager;

SELECT EmpId, CLASS_NAME AS EmpType , FirstName, LastName, Rate, FROM
    ONLY ProjectMember;
```

Alternatively if your model defines EmpType as in INT, you will have to run the following queries:

```
SELECT EmpId, 2 AS EmpType , FirstName, LastName, Rate,
    SignatureLevel, BudgetAuthority FROM ProjectManager;
```

```
SELECT EmpId, 1 AS EmpType , FirstName, LastName, Rate, FROM ONLY
    ProjectMember;
```

Create Primary Key / Foreign Key

Consider the case of the Person and Presidency classes defined in *Figure 2.3.1* where there is no user-defined Primary Key defined on the Person class.

The following queries are generating data that matches the relational model defined in *Figure 2.3.2*.

```
SELECT OID AS Pid, LastName, MiddleInitial, FirstName, Spouse.OID AS
    Sid FROM Person;

SELECT Number, StartingYear, EndingYear, IsInChargeOf.OID AS Pid FROM
    Presidency;
```

3.2 Step 2: Importing into a Relational Database

At this stage, you should have created a set of CSV files in a format that matches your relational database schema.

To import the data into your relational application you will use the import/export utilities of your target relational product.

4 Using DTS Utilities

4.1 The mt_dts Utility

The mt_dts utility can be used to perform import and export of data and to link data objects.

To import a CSV file input.csv into a database example on host localhost to populate the class MyClass, use the following command:

```
> mt_dts -d example@localhost import -f input.csv -c MyClass
```

and to update the class MyClass, use the following command:

```
> mt_dts -d example@localhost import -f input.csv -c MyClass -update
```

To export objects specified by the SQL statement "SELECT ..." from the database example on host localhost to the file output.csv, use the following command:

```
> mt_dts -d example@localhost export -f output.csv --sql "SELECT ..."
```

To establish links between data objects in the database example on host localhost, use the following command:

```
> mt_dts -d example@localhost link -f myfile.xrd
```

You can get a status report of the number of objects imported/exported. The status report is written to the standard error. The -h option provides a full description of the command line options.

Return Status

The mt_dts utility can return a status value as listed below.

Table 4.1.1 mt_dts status values

Status	Code	Description
SUCCESS	0	Successful. The whole CSV file has been stored into the database as new data objects.
PSUCCESS	1	Successful. However, some elements in the CSV file were not imported in the database, since they already existed in the database.
MATISSE_ERROR	2	Error regarding Matisse (for example, class not found).
SYNTAX_ERROR	3	Error regarding CSV file format.
NOSUCHFILE	4	The CSV file or option file specified in the command line is not found.
INVALIDARGS	5	Arguments in the command line are invalid.
INVALIDOPTIONS	6	Options in the option file are invalid.
FIELDNOTFOUND	7	A field in the filed does not correspond to any property in the class.

Location

`mt_dts` is located in $MATISSE_HOME/bin.

4.2 Connection options

Tag Definitions

Table 4.2.1 Connection option tags

Tag	Values	Default Value	Description
memoryTransport	YES | NO	YES	
transportBufferSize	64, 128, 256, 512	128	
objectsPerTransaction	between 1 and 100,000	5120	
discardInvalidRows	YES | NO	YES	
accessForUpdate	YES | NO	YES	
parseOnly	YES | NO	NO	Parse the file then create objects but rollback all changes.

XML Format

Example:

```
<DTSconnection>
   <memoryTransport>YES</memoryTransport>
   <transportBufferSize>128</transportBufferSize>
   <objectsPerTransaction>1024</objectsPerTransaction>
   <discardInvalidRows>YES</discardInvalidRows>
   <accessForUpdate>YES</accessForUpdate>
    <parseOnly>NO</parseOnly>
</DTSconnection>
```

4.3 Import options

Tag Definitions

Table 4.3.1 Import option tags

Tag	Values	Default Value	Description
className	A class name	None	The filename is selected for class name if the tag is not defined in the options.
fieldName	YES | NO	YES	Include field name on the first row.
fieldDelimiter	‘,’ | ‘;’ | {tab}	,	
textQualifier	“ | ‘	“	
bytesQualifier	a character string	0x	a prefix to qualify a media type (MT_BYTES, MT_IMAGE, MT_AUDIO, MT_VIDEO) represented in hexadecimal coded-ascii characters
allowUpdates	YES | NO	NO	allow primary key-based updates. The primary key must be in column 1. For a multi-column key, the columns (1 through 4) must match the order defined in the primary key index.

Table 4.3.1 Import option tags

Tag	Values	Default Value	Description
columnFromFile	a Column name	None	Contains the large data filename (image, audio, video, bytes or text). The 'Directory' tag can define the files location (i.e. Directory="photos"). If the 'Directory' tag is omitted, the files must be in the same directory as the CSV file.
dateOrder	MDY | YMD | DMY	YMD	
yearDigits	2 | 4	4	
dateDelimiter	/ | -	-	
timeDelimiter	:	:	
decimalSymbol	. | ,	.	
mediaData	File | Column	File	Import/export large binary data (`MT_BYTES, MT_IMAGE, MT_AUDIO, MT_VIDEO`) and large text data (`MT_TEXT`) embedded as field values in the CSV file or externalized in files with a filename referred in the field

XML Format

Example:

```
<DTSimport>
  <className>Person</className>
  <fieldName>YES</fieldName>
  <fieldDelimiter>,</fieldDelimiter>
  <textQualifier>”</textQualifier>
  <bytesQualifier>0x</bytesQualifier>
  <dateOrder>YMD</dateOrder>
  <yearDigits>4</yearDigits>
  <dateDelimiter>-</dateDelimiter>
  <timeDelimiter>:</timeDelimiter>
  <decimalSymbol>.</decimalSymbol>
  <mediaData>File</mediaData>
</DTSimport>
```

XML Format for large data

Example of importing images from the directory photo:

```
<DTSimport>
  <columnFromFile Directory="photos">Photo</columnFromFile>
</DTSimport>
```

NOTE: Large binary data (MT_BYTES) and large text data (MT_TEXT) are imported as field values. But using the ColumnFromFile option tag allow to import the data from a file.

4.4 Export options

Tag Definitions

Table 4.4.1 Export option tags

Tag	Values	Default Value	Description
className	A class name	None	
selectStatement	A SQL select statement	None	
skipOIDColumn	YES | NO	YES	Skip to export the OID column.
fieldName	YES | NO	YES	Include field name on the first row.
fieldDelimiter	‘,’ | ‘;’ | {tab}	,	
textQualifier	“ | ‘	“	
bytesQualifier	a character string	0x	a prefix to qualify a media type (MT_BYTES, MT_IMAGE, MT_AUDIO, MT_VIDEO) represented in hexadecimal coded-ascii characters
columnToFile	a column name	None	Contains the large data filename (image, audio, video, bytes or text) exported into a external file. The ‘Directory’ tag can define the files location (i.e. Directory="C:\photos"). By default, if the ‘Directory’ tag is omitted, the files are created in the same directory as the CSV file. The ‘FilenameFormat’ tag defines the file name format (see below for more details). If the ‘FilenameFormat’ tag is omitted, the filename format is as follows: {ClassName}_{AttributeName}_{RowId}.{data type}
dateOrder	MDY | YMD | DMY	YMD	
yearDigits	2 | 4	4	
dateDelimiter	/ | -	-	
timeDelimiter	:	:	
decimalSymbol	. | ,	.	
booleanSymbol	1/0 | True/False | Yes/No	True	
nullSymbol	a character string	NULL	export `NULL` values with the keyword of your choice or with an empty field
mediaData	File | Column	File	Import/export large binary data (`MT_BYTES, MT_IMAGE, MT_AUDIO, MT_VIDEO`) and large text data (`MT_TEXT`) embedded as field values in the CSV file or externalized in files with a filename referred in the field

XML Format

Example:

```
<DTSexport>
```

```
  <className>Person</className>
  <selectStatement>SELECT * FROM Person</selectStatement>
  <skipOIDColumn>YES</skipOIDColumn>
  <fieldName>YES</fieldName>
  <fieldDelimiter>,</fieldDelimiter>
  <textQualifier>”</textQualifier>
  <bytesQualifier>0x</bytesQualifier>
  <dateOrder>YMD</dateOrder>
  <yearDigits>4</yearDigits>
  <dateDelimiter>-</dateDelimiter>
  <timeDelimiter>:</timeDelimiter>
  <decimalSymbol>.</decimalSymbol>
  <booleanSymbol>True</booleanSymbol>
  <nullSymbol>NULL</nullSymbol>
  <mediaData>File</mediaData>
</DTSexport>
```

XML Format for large data

Using the ColumnToFile tag allows to customize the media filename to be exported.

```
<columnToFile Directory="C:\photos"
              FilenameFormat="{MediaName}">Photo</columnToFile>
```

The ‘Directory’ tag can define the files location (i.e. Directory="C:\photos"). By default, if the ‘Directory’ tag is omitted, the media files are created in the same directory as the CSV file.

The ‘FilenameFormat’ tag defines the file name format. It can be composed of text and parameters. A parameter can be an column name, RowId, or OID. a Parameter name is defined inside curly braces ({}).

If the ‘FilenameFormat’ tag is omitted, the filename format is as follows:

```
{ClassName}_{ColumnName}_{RowId}.{data type}
```

This default format for attribute Photo of class Employee will generate filenames such as Employee_Photo_1.img, Employee_Photo_2.img, Employee_Photo_3.img, etc.

For example, exporting images using the OID parameter as defined below will produce media filename such as image_5334.jpg, image_5338.jpg, image_5359.jpg, etc.:

```
<DTSexport>
  <selectStatement>SELECT c.MediaName,c.Photo FROM PhotoShot
    c</selectStatement>
  <columnToFile Directory="C:\Export\photos"
    FilenameFormat="image_{OID}.jpg">Photo</columnToFile>
</DTSexport>
```

For example, exporting images using a column name as parameter as defined below will produce media filename equal to the value of MediaName:

```
<DTSexport>
```

```
  <selectStatement>SELECT c.MediaName,c.Photo FROM PhotoShot
    c</selectStatement>
  <columnToFile Directory="C:\Export\photos"
    FilenameFormat="{MediaName}">Photo</columnToFile>
</DTSexport>
```

NOTE: Large binary data (MT_BYTES) and large text data (MT_TEXT) are exported as field values. But using the ColumnToFile option tag allow to export the data into a file.

4.5 Link options

Tag Definitions

Table 4.5.1 Link option tags

Tag	Values	Default Value	Description
fieldName	YES | NO	YES	Include field name on the first row.
fieldDelimiter	‘,’ | ‘;’ | {tab}	,	
textQualifier	“ | ‘	“	
dateOrder	MDY | YMD | DMY	YMD	
yearDigits	2 | 4	4	
dateDelimiter	/ | -	-	
timeDelimiter	:	:	
decimalSymbol	. | ,	.	

XML Format

Example:

```
<DTSlink>
   <fieldName>YES</fieldName>
   <fieldDelimiter>,</fieldDelimiter>
   <textQualifier>”</textQualifier>
   <dateOrder>YMD</dateOrder>
   <yearDigits>4</yearDigits>
   <dateDelimiter>-</dateDelimiter>
   <timeDelimiter>:</timeDelimiter>
   <decimalSymbol>.</decimalSymbol>
</DTSlink>
```

The associations between classes or tables are divided into 2 broad types: one-to-many and many-to-many.

One-to-Many

The one-to-many type describes associations with cardinality constraint of [0..1] on one side and [0..n] or [0..1] on the other side.

Typically, you will use one-to-many to describe the relationship Address between a class Person and a class Location or the relationship Spouse on the class Person.

Table 4.5.2 One-to-Many Relationship Descriptor tags

Tag	Option	Values	Default Value	Description
OneToMany				Object describing the elements to rebuild a one-to-many relationship between 2 classes.
	Name	A descriptor name		Defines the relationship descriptor name to help identify the descriptor in a file containing multiple descriptors.
	PreserveOrder	TRUE|FALSE	FALSE	To indicate that the order in which the are defined in the intermediate table must be preserved
PrimaryKey		A column name		Column name defining the primary key element of the association.
	Class	A class name		Defines the class name that supports the primary key.
	Relationship	A relationship name		Defines the relationship name used to materialize the one-to-many relationship.
	DeleteAfter	True | False	False	To indicate that the column will be deleted after the links are established.
ForeignKey		A column name		Column name defining the foreign key element of the association.
	Class	A class name		Defined the class name that support the foreign key.
	Relationship	a Relationship name		Defined the relationship name used to materialize the one-to-many relationship.
	DeleteAfter	True | False	False	To indicate that the column will be deleted after the links are established.
IntermediateTable				Object describing the intermediate table.
	DeleteAfter	True | False	False	To indicate that the intermediate table will be deleted after the links are established.
TableName		A table name		Name of the intermediate table.
IntermediateKey		A column name		Column name defining the foreign key element of the association.
	AssociatedClass	A class name		Defined the class name that support the associate primary key.
	AssociatedPrimaryKey	An attribute name		Defined the associated primary key.
IntermediateFile				Object describing the CSV file containing the link. IntermediateTable and IntermediateFile are exclusive
FileName		A file name		Name of the CSV file.
IntermediateColumn		A column name		Column name defining the foreign key element of the association.

Table 4.5.2 One-to-Many Relationship Descriptor tags

Tag	Option	Values	Default Value	Description
	AssociatedName	A Primary Key name		Defines the associated primary key name
	AssociatedClass	A class name		Defines the class name that support the associate primary key. Optional when AssociatedName is set.
	AssociatedPrimaryKey	An attribute name		Defines the associated primary key value. Optional when AssociatedName is set.

XML Format

Example:

```
<OneToMany>
    <PrimaryKey Class=”ProjectManager”
                Relationship=”Manages”
                DeleteAfter=”True”>EmpId</PrimaryKey>
    <ForeignKey Class=”Project”
                Relationship=”ManagedBy”
                DeleteAfter=”True”>ManagerId</ForeignKey>
</OneToMany>
```

XML Format Preserving the Order

To establish a one-to-many relationaship which preseves the order of the elements in the association, you need to define an intermediate table that describes the association elements in the order they will be created. Example:

```
<OneToMany Name=”ManagedProjectsRel” PreserveOrder=”TRUE”>
    <PrimaryKey Class=”ProjectManager”
                Relationship=”Manages”
                DeleteAfter=”True”>EmpId</PrimaryKey>
    <PrimaryKey Class=”Project”
                Relationship=”ManagedBy”
                DeleteAfter=”True”>ProjectId</PrimaryKey>
     <IntermediateTable DeleteAfter=”True”>
       <TableName>ManagedProjects</TableName>
       <IntermediateKey
          AssociatedClass=”ProjectManager”
          AssociatedPrimaryKey=”EmpId”>EmpId</IntermediateKey>
       <IntermediateKey
          AssociatedClass=”Project”
         AssociatedPrimaryKey=”ProjectId”>ProjectId</IntermediateKey>
    </IntermediateTable>
</OneToMany>
```

One-To-Many Ordered With Intermediate File

You can also create an ordered one-to-many relationship from data stored in a CSV file. The use of a CSV file is described in XML with three new tags: `IntermediateFile`, `FileName` and `IntermediateColumn`. An ordered one-to-many relationship descriptor with a CSV file is defined as follows:

```
<DTSlinks>
 <OneToMany Name="BookAuthor" PreserveOrder="True">
  <PrimaryKey Name="PersonId" Class="Author"
     Relationship="RecentPublications">PersonId</PrimaryKey>
  <PrimaryKey Name="ISBN" Class="Book">ISBN10</PrimaryKey>
```

```
  <IntermediateFile>
    <FileName>AuthorBooks.csv</FileName>
    <IntermediateColumn
     AssociatedName="PersonId">PersonId</IntermediateColumn>
    <IntermediateColumn
     AssociatedName="ISBN">ISBN10</IntermediateColumn>
  </IntermediateFile>
 </OneToMany>
</DTSlinks>
```

Many-to-Many

The many-to-many type describes associations with cardinality constraints of [0..n] on both sides.

Many-to-many can be used to describe the relationships Parent and Child on a class Person.

Table 4.5.3 Many-to-Many Relationship Descriptor tags

Tag	Option	Values	Default Value	Description
ManyToMany				Object describing the elements to rebuild a Many-to-Many relationship between 2 classes.
	Name	A descriptor name		Defines the relationship descriptor name to help identify the descriptor in a file containing multiple descriptors.
PrimaryKey		A column name		Column name defining the primary key element of the association.
	Name	A Primary Key name		Defines the primary key name to help identify the primary key with its matching its associate Primary Key defined in intermediate table or intermediate file.
	Class	A class name		Defines the class name that supports the primary key.
	Relationship	A relationship name		Defines the relationship name used to materialize one side of the many-to-many association.
	DeleteAfter	True | False	False	To indicate that the column will be deleted after the links are established.
IntermediateTable				Object describing the intermediate table. IntermediateTable and IntermediateFile are exclusive
	DeleteAfter	True | False	False	To indicate that the intermediate table will be deleted after the links are established.
TableName		A table name		Name of the intermediate table.

Table 4.5.3 Many-to-Many Relationship Descriptor tags

Tag	Option	Values	Default Value	Description
IntermediateKey		A column name		Column name defining the foreign key element of the association.
	AssociatedName	A Primary Key name		Defines the associated primary key name
	AssociatedClass	A class name		Defines the class name that support the associate primary key. Optional when AssociatedName is set.
	AssociatedPrimaryKey	An attribute name		Defines the associated primary key value. Optional when AssociatedName is set.
IntermediateFile				Object describing the CSV file containing the link. IntermediateTable and IntermediateFile are exclusive
FileName		A file name		Name of the CSV file.
IntermediateColumn		A column name		Column name defining the foreign key element of the association.
	AssociatedName	A Primary Key name		Defines the associated primary key name
	AssociatedClass	A class name		Defines the class name that support the associate primary key. Optional when AssociatedName is set.
	AssociatedPrimaryKey	An attribute name		Defines the associated primary key value. Optional when AssociatedName is set.

XML Format using an Intermediate Table

Example of a many-to-many relationship:

```
<DTSlinks>
 <ManyToMany Name="ProjectTasks">
    <PrimaryKey Class=”ProjectMember”
                Relationship=”AssignedTo”
                DeleteAfter=”True”>EmpId</PrimaryKey>
    <PrimaryKey Class=”Task”
                Relationship=”Assignee”
                DeleteAfter=”True”>TaskId</PrimaryKey>
    <IntermediateTable DeleteAfter=”True”>
       <TableName>AssignedTasks</TableName>
       <IntermediateKey
          AssociatedClass=”ProjectMember”
          AssociatedPrimaryKey=”EmpId”>EmpId</IntermediateKey>
       <IntermediateKey
          AssociatedClass=Task
          AssociatedPrimaryKey=”TaskId”>TaskId</IntermediateKey>
    </IntermediateTable>
 </ManyToMany>
</DTSlinks>
```

XML Format using a CSV File

Example of a many-to-many relationship:

```
<DTSlinks>
 <ManyToMany Name="Friends">
  <PrimaryKey Name="Member" Class="MemberProfile"
    Relationship="Friends">MemberId</PrimaryKey>
  <PrimaryKey Name="Friends" Class="MemberProfile"
    Relationship="Friends">MemberId</PrimaryKey>
  <IntermediateFile>
    <FileName>Friends.csv</FileName>
    <IntermediateColumn
    AssociatedName="Member">MemberId</IntermediateColumn>
    <IntermediateColumn
    AssociatedName="Friends">FriendId</IntermediateColumn>
  </IntermediateFile>
 </ManyToMany>
</DTSlinks>

example of content of the associated CSV file Friends.csv:

"MemberId","FriendId"
1,196
1,410
1,350
2,322
2,253
2,90
```

4.6 Default Options file

The default DTS options are presented below. These are provided in the `DTSdefault.opt` options file, which is located in $MATISSE_HOME/options:

```
<?xml version="1.0"?>
<DTSoptions>
  <DTSconnection>
    <memoryTransport>YES</memoryTransport>
    <transportBufferSize>128</transportBufferSize>
    <objectsPerTransaction>1024</objectsPerTransaction>
    <accessForUpdate>YES</accessForUpdate>
    <discardInvalidRows>YES</discardInvalidRows>
    <parseOnly>NO</parseOnly>
  </DTSconnection>
  <DTSimport>
    <className>MtClass</className>
    <fieldName>YES</fieldName>
    <fieldDelimiter>,</fieldDelimiter>
    <textQualifier>"</textQualifier>
    <bytesQualifier>0x</bytesQualifier>
    <dateOrder>YMD</dateOrder>
    <yearDigits>4</yearDigits>
    <dateDelimiter>-</dateDelimiter>
    <timeDelimiter>:</timeDelimiter>
    <decimalSymbol>.</decimalSymbol>
    <mediaData>File</mediaData>
    <allowUpdates>YES</allowUpdates>
```

```
  </DTSimport>
  <DTSexport>
    <selectStatement>SELECT * FROM MtClass</selectStatement>
    <skipOIDColumn>YES</skipOIDColumn>
    <fieldName>YES</fieldName>
    <fieldDelimiter>,</fieldDelimiter>
    <textQualifier>"</textQualifier>
    <bytesQualifier>0x</bytesQualifier>
    <dateOrder>YMD</dateOrder>
    <yearDigits>4</yearDigits>
    <dateDelimiter>-</dateDelimiter>
    <timeDelimiter>:</timeDelimiter>
    <decimalSymbol>.</decimalSymbol>
    <booleanSymbol>True</booleanSymbol>
    <nullSymbol>NULL</nullSymbol>
    <mediaData>File</mediaData>
  </DTSexport>
  <DTSlink>
    <fieldName>YES</fieldName>
    <fieldDelimiter>,</fieldDelimiter>
    <textQualifier>"</textQualifier>
    <bytesQualifier>0x</bytesQualifier>
    <dateOrder>YMD</dateOrder>
    <yearDigits>4</yearDigits>
    <dateDelimiter>-</dateDelimiter>
    <timeDelimiter>:</timeDelimiter>
    <decimalSymbol>.</decimalSymbol>
  </DTSlink>
</DTSoptions>
```

5 Importing Data from a CSV File Format

The `mt_dts` utility adheres to the standard CSV data representation.

5.1 Field Values

This section explains the valid format for the most common data types.

Integer

This includes the types `SHORT`, `INTEGER`, and `LONG`. The valid format for integer is as follows:

```
[+|-]{0-9}*
```

Real Number

This includes the types `FLOAT` and `DOUBLE`. The valid format for real numbers is as follows:

```
[+|-][{0-9}*][.{0-9}*][{e|E}[+|-]{0-9}*]
```

The following examples are valid values for real numbers:

```
123
123.
-.123
+1.23e05
123.E-5
```

Boolean

The valid values for the type `BOOLEAN` are:

```
true
false
yes
no
1
0
```

Date

The valid format for the type `DATE` is:

```
YYYY-MM-DD
MM-DD-YYYY
DD-MM-YYYY
YY-MM-DD
MM-DD-YY
DD-MM-YY
```

where `YYYY` is year number on four digits, `MM` is month number, and `DD` is the day number in the month.

For example, the following is a valid date:

```
2004-02-29
```

Timestamp

The valid format for the type `TIMESTAMP` is:

```
YYYY-MM-DD HH:mm:SS[.uuuuuu]
```

where `YYYY` is year number, `MM` is month number, `DD` is the day number in the month, `HH` is hour number (24 hour system), `mm` is minute number, `SS` is seconds number and `uuuuuu` is the micro-second number. The time is stored as GMT (Greenwich Mean Time).

For example, the following is a valid timestamp:

```
2004-01-06 23:24:00
```

The next one is not valid, since `HH` must be between 0 and 23:

```
2004-01-06 24:24:00
```

Interval

The valid format for the type `INTERVAL` is:

```
[+|-]DD HH:MM:SS[.uuuuuu]
```

where `DD` is number of days, `HH` is hour number, `MM` is minute number, `SS` is seconds number and `uuuuuu` is the micro-second number.

For example, the following is a valid interval:

```
+10 23:00:00.00
```

List

The valid format for the type `LIST` is:

```
list elt #1
list elt #2
[...]
list elt #n
```

where one list element is listed per row.

NOTE: To avoid data duplication in your CSV files, we recommend you use one CSV file for each column of type list.

5.2 Importing Composed Objects

Composed objects can be imported from a single CSV file. Composed objects are objects described with “part-of” relationships of type Composition.

The first line in the CSV file must list the fields name. Each field of the object parts is described by its full property path name. For example assuming the `Order` class described as follows:

```
interface Order : persistent
{
  attribute Integer OrderID;

  relationship PostalAddress BillAddress;
  relationship PostalAddress ShipAddress;
};

interface PostalAddress : persistent
{
  attribute String<16> Nullable City;
  attribute String<16> PostalCode;
};
```

The path to reach the Postal Code in the Billing Address is as follows:

```
BillAddress.PostalCode
```

The CSV file may look like the following:

```
$ more orders.csv
OrderID,BillAddress.City,BillAddress.PostalCode,ShipAddress.City,Sh
    ipAddress.PostalCode
10248,Bern,3012,Geneve,1204
```

The following command is importing composed objects in the `Order` class:

```
$ mt_dts -d example import -f orders.csv -c Order
```

6 Exporting Data into a CSV File Format

You can export data objects into a CSV file format. The objects are selected by executing a SQL SELECT statement.

6.1 Export Using SQL

You can use an SQL statement to specify objects to be exported. For example, to export objects of the class ProjectMember, whose last name starts with S, you may type:

```
> mt_dts -d mydb@myhost export -f output.csv --sql "SELECT EmpId,
        1 AS EmpType , FirstName, LastName, Rate, FROM ONLY
    ProjectMember
        WHERE LastName LIKE 'S%'"
```

The double quotation marks surrounding the SQL statement are for escaping characters such as * (asterisk) or ' (single quotation). The `mt_dts` utility reads all strings following -sql until the end of the command line.

For more information about SQL, refer to the *Matisse SQL Programmer's Guide*.

NOTE: Exporting LIST type values generates one row per element in the list and repeating other columns value. To avoid data duplication in your CSV files, we recommend you use one CSV file for each column of type list.

6.2 Exporting Composed Objects

Exporting composed objects from the `Order` class just require to execute a navigational SQL query such as:

```
SELECT OrderID,BillAddress.City AS
    "BillAddress.City",BillAddress.PostalCode AS
    "BillAddress.PostalCode",ShipAddress.City AS
    "ShipAddress.City",ShipAddress.PostalCode AS
    "ShipAddress.PostalCode" FROM "Order"
```

The following command is exporting all the composed objects from the `Order` class:

```
$ mt_dts -d example export -f orders.csv --sql "SELECT
    OrderID,BillAddress.City AS
    \"BillAddress.City\",BillAddress.PostalCode AS
    \"BillAddress.PostalCode\",ShipAddress.City AS
    \"ShipAddress.City\",ShipAddress.PostalCode AS
    \"ShipAddress.PostalCode\" FROM \"Order\""
```

The CSV file produced is as the following:

```
$ more orders.csv
```

```
"OrderID","BillAddress.City","BillAddress.PostalCode","ShipAddress.
    City","ShipAddress.PostalCode"
10248,"Bern","3012","Geneve","1204"
```

7 Programming with the DTS C API

If you need to manage data in CSV format with Matisse from within an application, you can use the Matisse DTS C Programming API.

7.1 Environment

Your program needs to include the C header file matisseDTS.h which is located in the directory $MATISSE_HOME/include. The shared library matisseDTS is located $MATISSE_HOME/lib.

7.2 API References

All the C API functions begin with the prefix MtDTS.

All of the APIs are listed below:

ImportDataFile

Synopsis

```
#include "matisseDTS.h"
MtString MtDTSImportDataFile
            (MtString host,
            MtString dbname,
            MtString username,
            MtString passwd,
            MtString csvFile,
            MtString optionsFile,
            MtString className)
```

Purpose This function reads a CSV file and store its content in a database.

Arguments

```
host      INPUT
```

The host where the database server is located.

```
dbName    INPUT
```

The database into which the data will be loaded.

```
username  INPUT
```

The user name of a database account. It can be set to NULL, in which case the system account is used if the database server enforces access control.

```
passwrd   INTPUT
```

The password associated with the user name. It can be set to NULL if the access control is not enforced.

```
csvFile    INPUT
```

A file containing the data in a CSV format.

```
optionsFile INPUT
```

A file containing the connection and import options. It can be set to NULL in which case the default options are used.

```
className INPUT
```

The class that will receive the data.

Result

A formatted character string containing (1) an error message, (2) statistic information when the loading is completed successfully or null is the verbose mode is turned off.

Description

This function reads the option file. It connects to the database to check that the database schema matches with the class name provided as well as the column names provided in the CSV file. Then for each valid row that is read in the CSV file, a new instance is created.

This function manages its own connection to the database.

ExportDataFile

Synopsis

```
#include "matisseDTS.h"
MtString MtDTSExportDataFile
            (MtString host,
            MtString dbname,
            MtString username,
            MtString passwd,
            MtString csvFile,
            MtString optionsFile,
            MtString sqlSelectStmt)
```

Purpose

This function executes the select statement and write the result set to a CSV format file.

Arguments

```
host       INPUT
```

The host where the database server is located.

```
dbName     INPUT
```

The database from which the data are exported.

```
username   INPUT
```

The user name of a database account. It can be set to NULL, in which case the system account is used if the database server enforces access control.

```
passwrd    INTPUT
```

The password associated with the user name. It can be set to NULL if the access control is not enforced.

```
csvFile    INPUT
```

A file receiving the data in a CSV format.

```
optionsFile INPUT
```

A file containing the connection and export options. It can be set to NULL in which case the default options are used.

```
sqlSelectStmt INPUT
```

The class that will receive the data.

Result — A formatted character string containing (1) an error message, (2) statistic information when the data export is completed successfully or null is the verbose mode is turned off.

Description — This function reads the option file. It connects to the database to execute the SQL select statement. Then the result set is exported into a file using the CSV options that are specified.

This function manages its own connection to the database.

EstablishRelationshipsFile

Synopsis

```
#include "matisseDTS.h"
MtString MtDTSEstablishRelationshipsFile
            (MtString host,
            MtString dbname,
            MtString username,
            MtString passwd,
            MtString xrdFile,
            MtString optionsFile)
```

Purpose — This function reads the XRD file and then it establishes the links between the data objects for each relationship description.

Arguments

```
host       INPUT
```

The host where the database server is located.

```
dbName     INPUT
```

The database into which the data will be loaded.

```
username   INPUT
```

The user name of a database account. It can be set to NULL, in which case the system account is used if the database server enforces access control.

```
passwrd    INTPUT
```

The password associated with the user name. It can be set to NULL if the access control is not enforced.

```
xrdFile    INPUT
```

An XRD file containing the description of the relationship to be established.

```
optionsFile INPUT
```

A file containing the connection and link options. It can be set to NULL in which case the default options are used.

Result

A formatted character string containing (1) an error message, (2) statistic information when the loading is completed successfully or null is the verbose mode is turned off

Description

This function reads the option file. It connects to the database to check that the database schema matches with the relationships described in the XRD file. Then it establishes the links between the data objects for each relationship defined in the XRD file.

This function manages its own connection to the database.

8 Table Conversion

8.1 Splitting a Table into a Class Hierarchy

When converting a relational model into Matisse, you may want to consider extending your application by splitting a single table into multiple tables which compose a class hierarchy.

For example splitting the ProjectMembers table into two classes ProjectMember and ProjectManager.

The following relational table becomes after split a two-class hierarchy in Matisse as presented below.

Before:

```
CREATE TABLE ProjectMembers (
      EmpId             INT,
      EmpType           INT,
      FirstName         VARCHAR(32),
      LastName          VARCHAR(32),
      Rate              NUMERIC(19,2),
      SignatureLevel    INT,
      BudgetAuthority   INT
);
```

After:

```
CREATE CLASS ProjectMember (
    EmpId           INT,
    FirstName       VARCHAR(32),
    LastName        VARCHAR(32),
    Rate            NUMERIC(19,2)
) ;
CREATE CLASS ProjectManager UNDER ProjectMember (
    SignatureLevel  INT,
    BudgetAuthority INT
) ;
```

Note that the EmpType field defined in the relational table to represent the type of employees is not carried over in the class hierarchy.

9 Data Types Conversion

9.1 SQL Server into Matisse

Table 9.1.1 Sql Server into Matisse data type conversion table

SQL Server Data type	Matisse SQL Data type	Matisse ODL Data type
bigint	LONG	Long
binary	BYTES | BLOB	List<Byte>
bit	BOOLEAN	Boolean
char	STRING | VARCHAR	String
datetime	TIMESTAMP	Timestamp
decimal	NUMERIC	Numeric
float	DOUBLE	Double
image	IMAGE	Image
int	INT | INTEGER	Integer
money	NUMERIC	Numeric
nchar	NVARCHAR	String UTF16
ntext	TEXT CHARACTER SET UTF16	Text UTF16
numeric	NUMERIC	Numeric
nvarchar	NVARCHAR	String UTF16
real	FLOAT	Float
small_datetime	TIMESTAMP	Timestamp
smallint	SHORT	Short
smallmoney	NUMERIC	Numeric
sql_variant	ANY	Any
text	TEXT	Text
Timestamp | rowversion	BYTES(8)	List<Byte, 8>
tinyint	BYTE	Byte
uniqueidentifier	VARCHAR(38)	String<38>
varbinary	BYTES | BLOB	List<Byte>
varchar	VARCHAR | STRING	String

9.2 Matisse into SQL Server

Table 9.2.1 Sql Server into Matisse data type conversion table

Matisse SQL Data type	Matisse ODL Data type	SQL Server Data type
ANY	Any	sql_variant
BOOLEAN	Boolean	bit
BYTE	Byte	tinyint
CHAR	Char	char(1)
DOUBLE	Double	float
FLOAT	Float	real
INT | INTEGER	Integer	int
INTERVAL	Interval	
LONG	Long	bigint
NUMERIC	Numeric	numeric
SHORT	Short	smallint
VARCHAR | STRING	String	varchar
NVARCHAR | VARCHAR CHARACTER SET UTF16	String UTF16	nvarchar
DATE	Date	datetime
TIMESTAMP	Timestamp	datetime
AUDIO	Audio	varbinary
BYTES | BLOB	List<Byte>	varbinary
IMAGE	Image	image
TEXT | CLOB	Text	text
VIDEO	Video	varbinary
NULL	NULL	
LIST(BOOLEAN)	List<Boolean>	
LIST(DATE)	List<Date>	
LIST(DOUBLE)	List<Double>	
LIST(FLOAT)	List<Float>	
LIST(INTEGER)	List<Integer>	
LIST(INTERVAL)	List<Interval>	
LIST(LONG)	List<Long>	
LIST(NUMERIC)	List<Numeric>	
LIST(SHORT)	List<Short>	
LIST(STRING) |LIST(VARCHAR)	List<String>	
LIST(NVARCHAR) | LIST(VARCHAR CHARACTER SET UTF16)	List<String UTF16>	
LIST(TIMESTAMP)	List<Timestamp>	