Matisse® 9.0.8 Release Notes

February 2013

Matisse 9.0.8 Release Notes

PDF generated 18 February 2013

Contents

1 New Features in Matisse 9.0

1.1 Overview

The Matisse 9.0 release introduces new features and major enhancements in the Matisse product line:

- The *Matisse Core Model* has been extended to support namespaces for schema objects.
- The *Matisse Enterprise Manager* has been extended to integrate the support for namespaces.
- *Matisse Database Modeler* has been enhanced to integrate the support for namespaces.
- Matisse tools and utilities (ODL, SQL, XML and DTS) have been extended to support the management of namespaces.
- The Java binding has been upgraded to support Java 7.
- The Eiffel binding has been upgraded to support Eiffel 7 for both 32-bit and 64-bit.
- All the language bindings (.NET, Java, C++, Python, PHP and Eiffel) have been extended to integrate the management of database schema object namespaces.
- The maximum capacity of the datafiles handled by the database server has been extended.
- The Matisse DBMS products are available on MacOS X.
- The new Objective-C binding has been added to the product line.

1.2 Matisse Core Model

The *Matisse Core Model* has been extended to support namespaces for schema objects. This new feature provides developers with more a powerful modeling capability and a seamless integration with the programming languages supported by Matisse.

Namespaces

Matisse Namespaces are identical in nature to namespaces provided by modern programming languages. They provide a seamless mechanism to define modules in large and complex data models.

Schema names

Schema names follow the naming conventions defined for identifiers or symbols in the supported programming languages (i.e. `MyClass`). Schema names can no longer include all ASCII characters (i.e. "`My Class's - [2011/June/18 @ 09:15]`").

Array datatypes

The Array datatypes (`MT_ARRAY_*`) which were not supported by most of the language bindings are no longer public types. The List datatypes are the datatypes for implementing Multi-value properties.

1.3 Enterprise Manager Tool

The *Matisse Enterprise Manager* has been enhanced to improve the database administrators and developers experience.

Schema Viewer

A `Namespaces` Node is added to the `Schema` and `Meta-Schema` nodes of an online database. The `Namespaces` Node lists all the namespaces in a table and the hierarchy of namespaces is displayed under the `Namespaces` Node. Each Namespace node details the information about the object.

The Schema object nodes under `Schema` and `Meta-Schema` nodes lists their respective schema objects in the namespace hierarchy.

Query Editors

The Object Browser Editor and SQL Query Analyzer Editor provide a pull-down menu to select the current namespace into which the query will be executed.

Export Schema

The Export ODL and DDL script dialogs provide an option to only export the database schema objects defined inside a namespace hierarchy.

Import/Export Data

The Import and Export CSV data dialogs provide an option to select a class defined inside a namespace.

Import XML

The Import XML data dialog has been redesigned to add an Advanced Options panel. It also provides an option to remap the data from a source namespace in the XML file into a destination namespace in the database.

Export XML

The Export XML data dialog has been redesigned to add an Advanced Options panel. It also provides an option to only export the objects from classes defined inside a namespace hierarchy.

Scheduled Tasks

The Scheduled Tasks dialog adds the option the option to schedule the recycling of the server log file as well as to schedule the execution of a user-defined script located in scripts/task directory in `MATISSE_HOME`.

Audit Log

The Enterprise Manager Lite now provides access to the log file in the Audit tab of the Host node.

Log and Resource files

The log and resource files for Enterprise Manager Lite are specific to this version and are named respectively `mtemgrlite_<login name>.log` and `mtemgrliterc`.

Manager License

The Manage License dialog enables the user to check and update the product license key.

Graphics and L&F

The Enterprise Manager images have been redesigned providing a better integration on the supported platforms.

Help

The Tutorial and Online Documentation menu-items in the Help menu have been added to improve the developer's experience.

1.4 Database Modeler

Matisse Database Modeler has been extended to support the management of schema object namespaces.

1.5 Matisse ODL

Matisse Object Definition Language (ODL) has been extended to support the management of namespaces.

Define a Namespace

The `module` keyword in the ODL syntax defines a namespace hierarchy.

module *Namespace_name* {...};

Associated sub-namespaces, classes, indexes and entry-point dictionaries are declared within the brackets.

For example, defining `Person` and `Company` classes in `MyCompany.MyApp` namespace and `Employee` and `Manager` in sub-namespace `HR`:

```
module MyCompany
{
  module MyApp
  {
    interface Person : persistent
    {
       attribute String<16> name;
    };
    interface Company : persistent
    {
       attribute String<32> name;
```

```
        };
        module HR
        {
          interface Employee: MyCompany.MyApp.Person: persistent
          {
          };
          interface Manager: Employee: persistent
          {
          };
        };
      };
    };
```

1.6 Matisse SQL

Matisse SQL has been extended to support the management of database schema object namespaces.

create namespace

To create a namespace in the database, you can use the `CREATE NAMESPACE` statement. The following statements create the `com.matisse.example` namespace hierarchy:

```
CREATE NAMESPACE com;
CREATE NAMESPACE com.matisse;
CREATE NAMESPACE com.matisse.example;
```

alter namespace

To rename an existing namespace, you can use `ALTER NAMESPACE RENAME` statement. For example, the following statement modifies the example sub-namespace name:

```
ALTER NAMESPACE com.matisse.example RENAME TO examples;
```

drop namespace

To remove a namespace from the database, you can use the `DROP NAMESPACE` statement. The following statement removes the examples sub-namespace:

```
DROP NAMESPACE com.matisse.examples;
```

set current_namespace

This option sets the default namespace where to find schema objects unless their names are fully qualified. `DEFAULT` refers to the root namespace. For example, the following statement sets the default namespace `com.matisse.example.media` to where schema objects can be manipulated without their fully qualified name:

```
SET CURRENT_NAMESPACE com.matisse.example.media;
CREATE CLASS movie ( ... );
SET CURRENT_NAMESPACE DEFAULT;
```

Pseudo Attributes

`MtFullClassName` returns the fully qualified class name of an object as string. For example,

```
SELECT LastName, MtFullClassName FROM Artist;
```

```
LastName           MtFullClassName
------------------ -------------------
Hanks              examples.media.Artist
Foster             examples.media.Artist
Spielberg          examples.media.MovieDirector
```

`MtFullName` returns the fully qualified class name of a schema object as string. For example:

```
SELECT MtName, MtFullName FROM MtClass;
MtName              MtFullName
------------------- -------------------
Artist              examples.media.Artist
Movie               examples.media.Movie
MovieDirector       examples.media.MovieDirector
```

Pseudo Relationships

Matisse SQL provides several pseudo relationships `MtAllAttributes`, `MtAllRelationships`, `MtAllInverseRelationships`, `MtAllSuperclasses`, `MtAllSubclasses`, `MtAllMethods`, `MtAllEntryPointDictionaries`, `MtAllIndexes` defined on `MtClass` to ease the description of class objects with inheritance. Each pseudo relationship navigates through the class hierarchy to aggregate the result produced in each individual level of the hierarchy.

`MtAllAttributes` returns the aggregation of the `MtAttributes` relationship value produced in each individual level of the class hierarchy. For example:

```
SELECT MtName,MtAttributes.MtName,MtAllAttributes.MtName FROM
    MtClass WHERE MtName = 'Manager';
MtName                MtName                MtName
--------------------- --------------------- ----------------
Manager               Expertise             EmpId
Manager               NULL                  LastName
Manager               NULL                  Expertise
```

`MtAllRelationships` returns the aggregation of the `MtRelationships` relationship value produced in each individual level of the class hierarchy. For example:

```
SELECT MtName,MtRelationships.MtName, MtAllRelationships.MtName FROM
    MtClass WHERE MtName = 'Manager';
MtName                MtName                MtName
--------------------- --------------------- ----------------
Manager               DirectReports         Address
Manager               ManageProjects        Accruals
Manager               NULL                  Department
Manager               NULL                  DirectReports
Manager               NULL                  ManageProjects
```

1.7 Schema Manager

Matisse Schema Manager has been enhanced to support the management of database schema object namespaces.

mt_sdl export

The `mt_sdl` utility with the `export` command allows you to export the database schema in ODL or SQL DDL format. The new `-n <namespace>` option exports the database schema defined into the provided namespace.

```
$ mt_sdl export -h
MATISSE Schema Definition Language x64 Version 9.0.0.0 (64-bit
    Edition) - Jan 30 2012.
(c) Copyright 1992-2012 Matisse Software Inc. All rights reserved.

Usage:
  mt_sdl [OPTIONS] export {-odl | -ddl} <schema file> [-n <namespace>]
    [-h]
    -odl  Generate the ODL class definitions from a database schema.
    -ddl  Generate the SQL DDL script from a database schema.
    -n    Export only the schema objects under the provided namespace
    -h    Display this help and exit.
```

mt_sdl stubgen

The `mt_sdl` utility with the `stubgen` command allows you to generate source code from the database schema classes defined in the ODL file. The new `-sn <namespace>` and `-ln <namespace>` options define the mapping between the schema class namespace and the language class namespace.

```
]$ mt_sdl stubgen -h
MATISSE Schema Definition Language x64 Version 9.0.0.0 (64-bit
    Edition) - Jan 30 2012.
(c) Copyright 1992-2012 Matisse Software Inc. All rights reserved.

Usage:
  mt_sdl stubgen {-lang cxx [-sn <namespace>] [-ln <namespace>] | -
    lang java [-sn <namespace>] [-ln <package>] | -lang php [-sn
    <namespace>] [-ln <namespace>] | -lang python [-sn <namespace>] |
    -lang eiffel [-sn <namespace>] } <ODL file> [-h]
    -lang cxx     Create C++ files from the ODL class definitions.
    -lang java    Create Java files from the ODL class definitions.
    -lang php     Create PHP files from the ODL class definitions.
    -lang python  Create Python files from the ODL class definitions.
    -lang eiffel  Create Eiffel files from the ODL class definitions.
    -sn           Specify the schema class namespace that is mapped to a
                language class namespace if any and if language supports
                  namespaces.
    -ln          Specify the language class namespace for the generated
    proxi
                 classes. when the -sn and -ln options are omitted, each
                  class is generated in a namespace matching the schema
                  class namespace.
    -psm          Generate methods mapping SQL method calls.
    -h            Display this help and exit.
```

1.8 Data Transformation Services

Matisse Data Transformation Services utility (`mt_dts`) has been updated to be consistent with other data management command line. The `import`, `export` and `link` commands require the `-f` option to specify the `CSV` or `XRD` file.

mt_dts import

The mt_dts utility with the import command allows you to load data in a CSV format into the database server.The -f option is required to specify the CSV file. The class name must be the fully qualified name if the class is defined in a namespace.

```
$ mt_dts import -h
MATISSE Data Transformation Services x64 Version 9.0.0.0 (64-bit
    Edition) - Jan 30 2012.
(c) Copyright 1992-2012 Matisse Software Inc. All rights reserved.

Usage:
mt_dts [OPTIONS] import -f <CSV file> [-c <class name>] [-update] [-
    noname] [-media File|Column]
    -f       Specify the CSV file to be loaded.
    -c       Specify the class name where the data will be loaded if
    different
            from the CSV filename.
    -update  When specified with the first columns of the CVS file
    composing
            the primary key, values of existing objects are updated.
   -noname  When specified there is no field name on the first row in
    the
            CVS file.
    -media   When specified with 'File', the media data is in an
    external file.
            With 'Column' the media data is in the CVS file.
```

mt_dts export

The mt_dts utility with the export command allows you to extract data in a CSV format from the database server.The -f option is required to specify the CSV file. The class name must be the fully qualified name if the class is defined in a namespace.

```
$ mt_dts export -h
MATISSE Data Transformation Services x64 Version 9.0.0.0 (64-bit
    Edition) - Jan 30 2012.
(c) Copyright 1992-2012 Matisse Software Inc. All rights reserved.

Usage:
  mt_dts [OPTIONS] export -f <CSV file> [-sql "<SQL select>" | -c
    <class name>] [-noname] [-media File|Column]
    -f       Specify the CSV file to be generated.
   -sql     Specify the SQL select statement which filters data to be
    exported.
    -c       Specify the class containing the data to be exported.
   -noname  When specified there is no field name on the first row in
    the
            CVS file.
    -media   When specified with 'File', the media data is exported
    into an
          external file. With 'Column' the media data is exported into
            the CVS file.
```

mt_dts link

The mt_dts utility with the link command allows you to extract data in a XRD format from the database server.The -f option is required to specify the XRD file.

```
$ mt_dts link -h
```

```
MATISSE Data Transformation Services x64 Version 9.0.0.0 (64-bit
    Edition) - Jan 30 2012.
(c) Copyright 1992-2012 Matisse Software Inc. All rights reserved.

Usage:
  mt_dts [OPTIONS] link -f <XRD file>
    -f  Specify the XML Relationship Descriptors file describing how to
        establish relationship between entities.
```

1.9 XML Manager

Matisse XML Manager utility (`mt_xml`) has been extended to load in parallel XML documents produced with the export parallel option. The XML utility has also been enhanced to support the management of schema object namespaces.

mt_xml import

The `mt_xml` utility with the `import` command provides the `-parallel <n>` option to load in parallel XML documents. It also provides the `-fn` and `-dn` options to remap the data from a source namespace in the XML file into a destination namespace in the database.

```
$ mt_xml import -h
MATISSE XML Manager x64 Version 9.0.4.0 (64-bit Edition) - Sep 13
    2012.
(c) Copyright 1992-2012 Matisse Software Inc. All rights reserved.

Usage:
  mt_xml [OPTIONS] import {-f <xmlfile> | -in} [-fn <nsname>] [-dn
    <nsname>] [-parallel <n>] [-parse <n>] [-update] [-commit <n> | -
    scommit]
    -f         Specify the XML data file to be loaded into the database.
    -in        Read the XML data from standard input and import it into
                the database.
    -fn         Specify the namespace from which the objects are
    imported.
    -dn         Specify the namespace into which the objects are
    imported.
              When the -fn and -dn options are ommitted, each object is
                imported in a namespace matching the schema class
    namespace.
    -parallel  Import data with <n> tasks running in parallel. The XML
    data is
                imported from a multi-segment XML file. The number of
    tasks is
                limited by the number of XML file segments.
    -parse      Specify the number of objects to be parsed in one data
    sequence.
              The default value is 256. The values range between 1 and
    512.
    -update     When specified with MtPrimaryKey attribute, values of
    existing
                objects are updated.
    -commit     Commit transaction for every <n> objects created. (by
    default
                commit occurs every 20480 objects created)
    -scommit   Forces to import the XML document in a single
    transaction
               (require enough memory to parse the XML document and to
    create
               all the objects in memory)
```

parallel import

The following example imports the XML documents in the database with 6 tasks running in parallel:

```
$ mt_xml -d example import -f outp6/exampleP6.xml -parallel 6
```

namespace mapping import

The following example imports the XML document in the database remapping the `proj1.app1` namespace into the database `app.client1` namespace:

```
$ mt_xml -d example import -f example.xml -fn proj1.app1 -dn
    app.client1
```

mt_xml export

The `mt_xml` utility with the `export` command provides the `-n` option to only export the objects from classes defined inside a namespace hierarchy.

```
]$ mt_xml export -h
MATISSE XML Manager x64 Version 9.0.4.0 (64-bit Edition) - Sep 13
    2012.
(c) Copyright 1992-2012 Matisse Software Inc. All rights reserved.

Usage:
  mt_xml [OPTIONS] export {-f <xmlfile> [-s <size>[M|G]] | -out} [-
    emedia] [-foid] [-parallel <n>] [-prefetch <n>] [-n <nsname>] {-
    full | -sql <stmt> | -oid <oid> ...}
   -f        Specify the XML data file storing XML data extracted from
    the
               database.
    -s        Specify the XML data file max size therefore splitting
    XML data
              into multiple XML files named <xmlfile>_xds_<docid>.xml.
    -out      Write XML data to the standard output.
    -emedia    Export media data in the XML document instead of
    exporting
               media data into external files.
    -foid      Export data in a format with OIDs in the xml tags to
    enable
               Primary Key recovery. (The -full option always exports
    in this
               format)
   -task     Export data with <n> tasks running in parallel. The XML
    data is
               exported into multiple XML files named
    <xmlfile>_xds_<docid>.xml.
   -parallel  Export data with <n> tasks running in parallel. The XML
    data is
               exported into multiple XML files named
    <xmlfile>_xds_a<docid>.xml
               and <xmlfile>_xds_r<docid>.xml. This is the XML format
    to import
               a multi-segment XML file in parallel.
```

```
 -prefetch  Specify the number of objects to be prefetched when
 exporting
           data. The default value is 128. The values range between
 1 and
            128.
 -full      Export all non-schema data into one or multiple XML
 files.
-sql       Specify the SQL SELECT statement retrieving the objects
 to be
            exported.
 -oid       Specify the list of object OIDs to be exported. Both
 decimal
            and hexadecimal oid formats are accepted.
-n        Specify the namespace from which the objects are exported.
```

parallel export

The following example exports the database with 12 tasks running in parallel into the multiple XML documents of 1gigabytes each:

```
$ mt_xml -d example export -f out/exampleP12.xml -parallel 12 -s 1G -
    full
```

namespace export

The following example exports in a XML document all the objects from the classes defined in the `app.client1` namespace:

```
$ mt_xml -d example export -f example.xml -n app.client1 -full
```

1.10 Database Utility Commands

mt_server create

The `mt_server create` command allows you to create the configuration file for a new database.

```
$ mt_server create -h
Usage: mt_server [OPTIONS] create [-p <size>[K]] [-c <size>[GM]] [-s]
    [-m] [-r] [-f <size>[GM]] [-h]

  -p, --pagesiz   Server page size (in Kbytes)
  -c, --cachesiz  Initial memory cache size (default in Mbytes)
  -s, --security  Access control enabled
  -m, --memory    In-memory database enabled
  -r, --mirror    Mirrored datafile
  -f, --filesiz   Initial datafile capacity (default in Mbytes)
  -h, --help      Display this help and exit
```

1.11 Java Binding

The Java binding has been upgraded to support Java 7 and extended to support the management of database schema object namespaces.

Java 7

Matisse Java binding has been upgraded to support Java 7.

Generating Stub Classes

The `mt_sdl` utility with the `stubgen` command allows you to generate Java source code from the database schema classes defined in the ODL file. The new `-sn <namespace>` and `-ln <namespace>` options define the mapping between the schema class namespace and the language class namespace.

For example, to generate the Java classes in the root package from the database schema classes defined in the `java_examples.chap_3` namespace.

```
mt_sdl stubgen -lang java -sn java_examples.chap_3 examples.odl
```

To generate the Java classes in the `com.corp.myapp` package from the database schema classes defined in the root namespace.

```
mt_sdl stubgen -lang java -ln com.corp.myapp example.odl
```

To generate the Java classes in the `com.corp.myapp` package from the database schema classes defined in the `Examples.Sample03` namespace.

```
mt_sdl stubgen -lang java -sn Examples.Sample03 -ln com.corp.myapp
    example.odl
```

To generate the Java classes in the `com.corp.myapp` package from the database schema classes defined in the `com.corp.myapp` namespace.

```
mt_sdl stubgen -lang java example.odl
```

To generate the Java classes in the root package from the database schema classes defined in the root namespace.

```
mt_sdl stubgen -lang java example.odl
```

Object Factory

The object factory classes handle the namespace mapping between the Java classes and the schema classes. The `getDatabaseClass()` method was added to the `MtObjectFactory` interface to handle the bidirectional namespace mapping.

The `MtPackageObjectFactory` class constructor takes the Java package and the Schema Namespace as additional argument.

```
// The third argument is given so that the connection object can find
// the mapping between the Java class Person, which is defined in the
// "com.mycomp.myapp" package and the schema class Person defined in
// the "examples.java_examples.chap_3" namespace.
MtDatabase db = new MtDatabase(hostname, dbname, new
    MtPackageObjectFactory("com.mycomp.myapp","examples.java_exampl
    es.chap_3"));
```

As an alternative you can use the `MtExplicitObjectFactory` class with the generated `examplesSchemaMap.txt` file that defines a direct class mapping between the Java classes and the database schema classes.

```
MtDatabase db = new MtDatabase(hostname, dbname, new
    MtExplicitObjectFactory("examplesSchemaMap.txt"));
```

SQL Execution

The new `setSqlCurrentNamespace()` method defined on the `MtDatabase` class sets the SQL `CURRENT_NAMESPACE`. The `getSqlCurrentNamespace()` returns the `CURRENT_NAMESPACE` value.

```
MtDatabase db = new MtDatabase(hostname, dbname);
db.startVersionAccess();
// Set the SQL CURRENT_NAMESPACE to 'examples.java_examples.jdbc'
db.setSqlCurrentNamespace("examples.java_examples.jdbc");
```

Schema Names

The new `getMtFullName()` method defined on the `MtClass`, `MtIndex`, `MtEntryPointDictionary` classes returns the fully qualified name for the schema object.

Examples

Matisse Java binding examples have been updated to define the database schema in a specific namespace.

1.12 .NET Binding

The .NET binding has been extended to support the management of database schema object namespaces.

Generating Stub Classes

The `mt_dnom` utility with the `stubgen` command allows you to generate .NET source code from the database schema classes defined in the database. The new `-sn <namespace>` and `-ln <namespace>` options define the mapping between the schema class namespace and the language class namespace.

For example, to generate the .NET classes in the root namespace from the schema classes defined in the `examples.chap_3` namespace.

```
mt_dnom -d example stubgen -lang C# -sn examples.chap_3
```

To generate the .NET classes in the `Corp.Myapp` package from the schema classes defined in the root namespace.

```
mt_dnom -d example stubgen -lang C# -ln Corp.Myapp
```

To generate the .NET classes in the `Corp.Myapp` package from the schema classes defined in the `Examples.Sample03` namespace.

```
mt_dnom -d example stubgen -lang C# -sn Examples.Sample03 -ln
    Corp.Myapp
```

To generate the .NET classes in the `Corp.Myapp` namespace from the schema classes defined in the `Corp.Myapp` namespace.

```
mt_dnom -d example stubgen -lang C#
```

To generate the .NET classes in the root namespace from the schema classes defined in the root namespace.

```
mt_dnom -d example stubgen -lang C#
```

Object Factory

The object factory classes handle the namespace mapping between the .NET classes and the schema classes. The `getDatabaseClass()` method was added to `MtObjectFactory` interface to handle the bidirectional namespace mapping.

The `MtPackageObjectFactory` class constructor takes the .NET namespace and the Schema namespace.

SQL Execution

The new `SqlCurrentNamespace` property defined on the `MtDatabase` class sets the SQL `CURRENT_NAMESPACE` and returns the `CURRENT_NAMESPACE` value.

Schema Names

The new `MtFullName` property defined on the `MtClass`, `MtIndex`, `MtEntryPointDictionary` classes returns the fully qualified name for the schema object.

Examples

Matisse .NET binding examples have been updated to define the database schema in a specific namespace.

1.13 C++ Binding

The C++ binding has been extended to support the management of database schema object namespaces.

Generating Stub Classes

The `mt_sdl` utility with the `stubgen` command allows you to generate C++ source code from the database schema classes defined in the ODL file. The new `-sn <namespace>` and `-ln <namespace>` options define the mapping between the schema class namespace and the language class namespace.

For example, to generate the C++ classes in the root namespace from the schema classes defined in the `cxx_examples.chap_3` namespace.

```
mt_sdl stubgen -lang cxx -sn cxx_examples.chap_3 examples.odl
```

To generate the C++ classes in the `corp.myapp` namespace from the schema classes defined in the `Examples.Sample03` namespace.

```
mt_sdl stubgen -lang cxx -sn Examples.Sample03 -ln corp.myapp
    example.odl
```

To generate the C++ classes in the `corp.myapp` namespace from the schema classes defined in the `corp.myapp` namespace.

```
mt_sdl stubgen -lang java example.odl
```

To generate the C++ classes in the root namespace from the schema classes defined in the root namespace.

```
mt_sdl stubgen -lang java example.odl
```

Object Factory

The object factory classes handle the namespace mapping between the C++ classes and the schema classes. The `getDatabaseClass()` method was added to the `MtObjectFactory` interface to handle the bidirectional namespace mapping.

The `MtDynamicObjectFactory` class constructor takes the C++ namespace and the database schema namespace.

SQL Execution

The new `setSqlCurrentNamespace()` method defined on the `MtDatabase` class sets the SQL `CURRENT_NAMESPACE`. The `getSqlCurrentNamespace()` returns the `CURRENT_NAMESPACE` value.

Schema Names

The new `getMtFullName()` method defined on the `MtClass`, `MtIndex`, `MtEntryPointDictionary` classes returns the fully qualified name for the schema object.

1.14 Eiffel Binding

The Eiffel binding has been extended to support the management of database schema object namespaces.

64-bit Support

Matisse Eiffel binding has been updated to support 64-bit eiffel application.

Generating Stub Classes

The `mt_sdl` utility with the `stubgen` command allows you to generate Eiffel source code from the database schema classes defined in the ODL file. The new `-sn <namespace>` option define the mapping between the schema class namespace and the language class.

For example, to generate the Eiffel classes from the schema classes defined in the `eif_examples.chap_3` namespace.

```
mt_sdl stubgen -lang eiffel -sn eif_examples.chap_3 examples.odl
```

To generate the Eiffel classes from the schema classes defined in the root namespace.

```
mt_sdl stubgen -lang eiffel example.odl
```

Object Factory

The object factory classes handle the namespace mapping between the Eiffel classes and the schema classes. The `get_database_class()` method was added to the `MT_OBJECT_FACTORY` virtual class to handle the bidirectional namespace mapping.

SQL Execution

The new `set_sql_current_namespace()` method defined on the `MT_DATABASE` class sets the SQL `CURRENT_NAMESPACE`. The `get_sql_current_namespace()` returns the `CURRENT_NAMESPACE` value.

Schema Names

The new `mtfullname()` method defined on the `MTCLASS`, `MTINDEX`, `MTENTRYPOINTDICTIONARY` classes returns the fully qualified name for the schema object.

1.15 PHP Binding

The PHP binding has been extended to support the management of database schema object namespaces.

Generating Stub Classes

The `mt_sdl` utility with the `stubgen` command allows you to generate PHP source code from the database schema classes defined in the ODL file. The new `-sn <namespace>` and `-ln <namespace>` options define the mapping between the schema class namespace and the language class namespace.

For example, to generate the PHP classes in the root namespace from the schema classes defined in the `php_examples.chap_3` namespace.

```
mt_sdl stubgen -lang php -sn php_examples.chap_3 examples.odl
```

To generate the PHP classes in the root namespace from the schema classes defined in the root namespace.

```
mt_sdl stubgen -lang php example.odl
```

Object Factory

The object factory classes handle the namespace mapping between the PHP classes and the schema classes. The `getDatabaseClass()` method was added to the `MtObjectFactory` interface to handle the bidirectional namespace mapping. The `MtDynamicObjectFactory` class constructor takes the PHP namespace and the database schema namespace.

SQL Execution

The new `setSqlCurrentNamespace()` method defined on the `MtDatabase` class sets the SQL `CURRENT_NAMESPACE`. The `getSqlCurrentNamespace()` returns the `CURRENT_NAMESPACE` value.

Schema Names

The new `getMtFullName()` method defined on the `MtClass`, `MtIndex`, `MtEntryPointDictionary` classes returns the fully qualified name for the schema object.

1.16 Python Binding

The Python binding has been extended to support the management of database schema object namespaces.

Generating Stub Classes

The `mt_sdl` utility with the `stubgen` command allows you to generate Python source code from the database schema classes defined in the ODL file. The new `-sn <namespace>` option define the mapping between the schema class namespace and the language class.

For example, to generate the Python classes from the schema classes defined in the `py_examples.chap_3` namespace.

```
mt_sdl stubgen -lang python -sn py_examples.chap_3 examples.odl
```

To generate the Python classes from the schema classes defined in the root namespace.

```
mt_sdl stubgen -lang python example.odl
```

Object Factory

The object factory classes handle the namespace mapping between the Python classes and the schema classes. The `getDatabaseClass()` method was added to the `MtObjectFactory` interface to handle the bidirectional namespace mapping. The `MtDynamicObjectFactory` class constructor can take one argument that is the Schema Namespace.

SQL Execution

The new `setSqlCurrentNamespace()` method defined on the `MtDatabase` class sets the SQL `CURRENT_NAMESPACE`. The `getSqlCurrentNamespace()` returns the `CURRENT_NAMESPACE` value.

Schema Names

The new `getMtFullName()` method defined on the `MtClass`, `MtIndex`, `MtEntryPointDictionary` classes returns the fully qualified name for the schema object.

1.17 Objective-C Binding

The new Objective-C binding has been added to the MacOS X product line. The Objective-C binding is open source. It is comprised of 2 main files `matisseObjC.h` and `matisseObjC.m`. This binding complies with the design pattern used in the other Matisse Language bindings while respecting the programming guide line recommended for Objective-C.

Generating Stub Classes

The `mt_sdl` utility with the `stubgen` command allows you to generate Objective-C source code from the database schema classes defined in the ODL file. The `-sn <namespace>` options define the mapping between the schema class namespace and the language class.

For example, to generate Objective-C classes from the schema classes defined in the `cxx_examples.chap_3` namespace.

```
mt_sdl stubgen -lang objC -sn cxx_examples.chap_3 examples.odl
```

To generate the Objective-C classes from the schema classes defined in the root namespace.

```
mt_sdl stubgen -lang objC example.odl
```

Examples

Matisse Objective-C binding examples are available in a standalone package.

1.18 Database Configuration

A new optional database configuration parameter has been added to minimize the file system cache effects of the write and read operations to and from the data file. The capacity of a datafile as well as their number managed by a Matisse server has been modified to better utilize the storage hardware.

DATIODIRECT

This parameter defines the I/O mechanism for data files on the file system. When set to 1, it minimizes the file system cache effects of the I/O to and from the data file. This parameter has no effect on systems that do not support this OS feature.

Datafiles

The database server can manage up to 31 primary datafiles on individual disks or RAID-enabled disks and up to 31 mirrored datafiles. The datafile maximum capacity has been extended to 512 Gigabytes with a standard page size of 8 Kilobytes and therefore the maximum database size of roughly 16 Terabytes. Multi-media databases with page size of 32 Kilobytes have a maximum database size of roughly 64 Terabytes.

1.19 MacOS X

The Matisse DBMS 32-bit and 64-bit as well as Matisse Lite products are available on MacOS X.

1.20 License Key Format

The customer license key format has changed in release 9.0. Matisse 9.0 does not recognize license keys issued for prior releases. Upon installation of Matisse 9.0, a license key with limited features is automatically issued.

2 Compatibility with Previous Releases

2.1 Matisse 9.0 Data Migration

Matisse Server 9.0 comes with several changes in the data format. You must use the `mt_xml` tool to convert an existing database (8.4.x or prior) into the 9.0 format.

Step 1

Before installing 9.0.x, check if your data and your application are compatible with Matisse 9.0 since the array datatypes are no longer public. Most likely you are already using the list datatypes since the vast majority of the Matisse language bindings do not support Matisse array datatypes. To double-check, run the following SQL statement, which does not return any row if your application is compatible.

```
  mt_sql -d <dbname>
SQL>SELECT
MtBasicType,
MtAttributeTypeOf.MtName AS AttributeName,
MtAttributeTypeOf.MtAttributeOf.MtName AS ClassName
FROM MtType
WHERE MtBasicType = 16 OR
MtBasicType = 20 OR
MtBasicType = 24 OR
MtBasicType = 40 OR
MtBasicType = 44 OR
MtBasicType = 48;
```

However if the query returns one or more rows, you need to migrate your data and upgrade your application. Replace the Array data type with the corresponding List data type.

Step 2

Before installing 9.0.x, save your schema in ODL and your data in XML format:

```
  mt_sdl -d <dbname> export -odl schema.odl

  mt_xml -d <dbname> export -f data.xml -full
```

You may check the *Matisse XML Programming Guide* for more options to export in XML format.

Step 3

You may now install Matisse 9.0.x on your machine and then restore the schema and the data as follows:

```
  mt_sdl -d <dbname> import -odl schema.odl

  mt_xml -d <dbname> import -f data.xml
```

2.2 Client Connections

Only 9.0.x clients may be used with 9.0.x servers.

The clients for earlier releases of Matisse are incompatible with the 9.0.x server. Consequently, you must upgrade any older clients to 9.0.x before attempting to access a 9.0.x server.

3 Platform-Specific Topics

3.1 Linux

The most popular Linux distributions on x86 (32-bit) and x86_64 (64-bit) chip families are supported. Any Linux distribution, where Matisse DBMS has not been tested, require Linux kernel 2.6.18 on systems based on x86 (32-bit) or x86_64 (64-bit) chip families.

3.2 MacOS

Support for MacOS X version for Intel (10.5 Leopard or higher).

3.3 Solaris

Support for Solaris 10 on x86 (32-bit) and x86_64 (64-bit) chip families.

The Solaris 10 on SPARC with 32-bit kernel and 64-bit kernel is available upon request.

3.4 Windows

Support for Windows (XP/2003/2008/2012/Vista/7/8) on systems based on x86 (32-bit) and x86_64 (64-bit) chip families.

4 Update History

This section contains the list of bug fixes and minor feature changes between releases. You may refer to it before upgrading to see if the new release resolves a known problem or adds a needed feature.

Resolved in Matisse 9.0.8

- The log files for the Port Monitor, Server Manager, SMListener and Enterprise Manager programs now report the edition of the running program (32-bit or 64-bit).
- In the C API, the `MT_VERSION_NAME_PREFIX_MAX_LEN` constant was extended to 23 characters from 20 characters to support snapshot name up to 31 characters.
- On Windows, updating the product license key in the Enterprise Manager may stop the Matisse Server Manager running as a Windows service.
- Creating a schema object of type `MtObject`, `MtType`, `MtMethod` or `MtMetaClass` while not being connected to the database in `MT_DATA_DEFINITION` mode fails to report the correct error message.
- In the C API function `MtCtxSetListElements` replacing a `MT_LIST` value with a `MT_TEXT` value or vise versa may fail to report the expected error message.
- In some cases SQL Version Travel queries fail to return the correct count of objects for the `count(*)` built-in function.
  ```
  SELECT count(*) FROM DELETED (Document, BEFORE DAY03_0000000C)
  SELECT count(*) FROM UPDATED (Document, AFTER DAY01_0000000A)
  SELECT count(*) FROM INSERTED (Document, AFTER DAY01_0000000A)
  ```
- In some cases SQL Version Travel queries fail to return objects which have only updated a few objects in large relationships.
- In some cases loading in parallel with at least 8 tasks an entire database from XML files containing a majority of objects with attributes containing large LIST values may fail.
- In some cases, loading an entire database from an XML file may turn the index key uniqueness check off until the database is stopped and restarted.
- On Linux, a database with `DATIODIRECT` configuration parameter enabled may fail to report a specific enough error message in case the file system settings prevent from using of the operating system call associated with this database configuration parameter.
- The `mt_server init` command with the `--wait` option takes at least 10 seconds to return whatever the database configuration.
- On Windows the `mt_server list --stopped` command may fail when the number of databases to be listed become very large.

Resolved in Matisse 9.0.7

- In the Enterprise Manager, the Monitor as well as the Datafiles panels provide information about the high watermark reached for each datafile. When a version collection is running, the datafile status now includes collection level and scanning progress.
- The `mt_server monitor` command with the `-w` (`--wide`) option now provides information about the high watermark reached for each datafile. When a version collection is running, the datafile status now includes collection level and scanning progress.
- The `mt_file list` command with the `-w` (`--wide`) option now provides information about the high watermark reached for each datafile. When a version collection is running, the datafile status now includes collection level and scanning progress.
- In the Java binding, the `MtDatabase.getObjectFactory()` method which returns the object factory associated to a connection has been added:
  ```
  MtObjectFactory MtDatabase.getObjectFactory()
  ```
- In the Java binding, the `toString()` methods defined on `MtCoreObjectFactory`, `MtExplicitObjectFactory`, and `MtPackageObjectFactory` (`MtObjectFactory` class implementations) return details about the mapping between the database schema classes and their Java class implementations.
  ```
  String MtPackageObjectFactory.toString()
  String MtExplicitObjectFactory.toString()
  String MtCoreObjectFactory.toString()
  ```
- In the java binding in some cases, an object read from the database fails to load the corresponding Java class leading to the error:
  ```
  java.lang.ClassCastException: com.matisse.reflect.MtObject cannot be cast to
  com.examples.myclass.
  ```
- A new optional database configuration parameter (`DATIODIRECT`) has been added to minimize the file system cache effects of the write and read operations to and from the data file.
- In some cases, the automatic version collection does not clear up the entries from the Object Table while it should.
- On Linux, the execution of the `mt_databases.sh` script reports a syntax error during the machine boot unless the `start()` and `stop()` functions have been properly updated to manage databases.

Resolved in Matisse 9.0.6

- In the C++ binding, the `MtDatabase::getConnectionId()` method which returns the connection-id has been added:
  ```
  int MtDatabase::getConnectionId() const
  ```
- The `mt_file` utility now allows to change the `AUTOEXTEND` and `DATEXTENDSIZ` configuration parameter values on an online database.
  ```
  Usage: mt_file [OPTIONS] autoextend {-e|-d}
     -e, --enable     Enable autoextend (default)
     -d, --disable    Disable autoextend
  ```

```
   -h, --help      Display this help and exit

Usage: mt_file [OPTIONS] setextendsiz -s <size>
   -s, --size=...  Minimum file extension size in Mega bytes
   -h, --help      Display this help and exit
```

- The new `-w` (`--wide`) option has been added to `mt_transaction list` command to display more information about active transactions.
- The `mt_transaction list` command truncates the hostname and username at 16 characters.
- In the Objective C binding, the `[MtDatabase connectionId]` method does not return the correct connection-id when the application is linked to the Matisse Lite library.
- In very rare cases, the server version collection operation may fail to complete with the error: `GOM-F-VCTASKEXCEP, Exception in VC task for file 2 at page 1099122927`. This may also lead to a substantial increase of the database size.
- On `UNIX` installations, the `MATISSE_CPP` environment variable defined in `mt_env.sh` and `mt_env.csh` is now commented out.

Resolved in Matisse 9.0.5

- In the Enterprise Manager, the Scheduled Tasks dialog adds the option to schedule the recycling of the server log file.
- In the Enterprise Manager, the Scheduled Tasks dialog adds the option to schedule the execution of a user-defined script located in `scripts/task` directory in `MATISSE_HOME`. The database name is the script only parameter.
- The `mt_server recyclelog` command line receives 2 new options `-s` and `-b` to recycle respectively the server log file and the backup journal log file.
- In SQL, the compilation of a block statement which includes a `SELECTION` construct in a `RETURN` statement fails with a syntax error.

```
BEGIN
  DECLARE emp_sel SELECTION(Employee);
  DECLARE mgr_sel SELECTION(Manager);
  RETURN SELECTION(emp_sel UNION mgr_sel);
END;
```

- In SQL, the compilation of a block statement which includes a `SELECTION` construct with only one object does not produce the correct instruction.

```
BEGIN
  DECLARE res SELECTION(Employee);
  DECLARE obj Employee;
  SET res = SELECTION(obj);
  RETURN res;
END;
```

- In SQL, the compilation of a `SELECT` statement calling a static method of a class defined in a namespace fail to compile with the `MATISSE_NOSUCHCLASS` error.

```
SELECT
  e.EmpId,
  e.GetAccrualsQuantity(tutorials.reports.Bank::ListBankName(FALSE)) AS Total
FROM
  tutorials.reports.Employee e
```

- In some cases, the SQL compiler does not select the best index when the `WHERE` clause includes an `OR` condition as well as when classes in the navigation path matches with multiple indexes.

- Ins some cases, the execution of `SELECT` statement in which the select clause refers directly to a Pseudo-relationship fails with the `MATISSE_OBJECTNOTFOUND` error

```
SELECT MtMame,MtAllIndexes FROM MtClass
```

- In the Enterprise Manager, deleting an active task in the Scheduled Tasks dialog fails to stop the task.

- In some cases, the execution of an incremental backup in a scheduled task actually executes a full backup since it fails to recognize that there is no increment to backup.

- The `mt_file list` command reports by default the datafile information in Mbytes from previously in Kbytes.

- The long option `--mirror` for the `mt_file add` command is inactive while the short option `-m` adds a mirrored datafile.

- The `MAXBKPLOGFILES` configuration parameter does not properly limit the number of backup journal log file preserved.

Resolved in Matisse 9.0.4

- In the Enterprise Manager, the Import XML dialog and Export XML dialog have been redesigned to add an Advanced Options panel.

- In the Enterprise Manager, the Create Database dialog manages in the Server panel the following configuration parameters:

```
AUTOCOLLECTFREQ
OBJTABLESIZ
OBJTABCLRFREQ
```

- In the Enterprise Manager, A `Checkpoint database` menu item has been added to the main menu under `Tasks` for in-memory databases. In addition a pop-up menu was added to the `Management` node of an online database to modify server parameters. The menu items include `Collect object versions`, `Change auto-collect mode`, `Extend page cache`, `Extend object table cache`, `Set auto-collect run frequency`, `Set clear object table run frequency`, and `Change transaction mode`.

- New SQL pseudo-relationships have been defined on `MtClass` to improve class description of classes with inheritance:

```
MtClass.MtAllIndexes
MtClass.MtAllEntryPointDictionaries
```

- New C API functions have been added to improve class description of classes with inheritance:
  `MtSTS MtCtxMGetAllIndexes(MtContext ctx, MtSize* numHandles, MtOid** indexes, MT_CONST MtChar* className);`
  `MtSTS MtCtx_MGetAllIndexes(MtContext ctx, MtSize* numHandles, MtOid** indexes, MtOid classe);`
  `MtSTS MtCtxGetAllIndexes(MtContext ctx, MtSize* numHandles, MtOid* indexes, MT_CONST MtChar* className);`
  `MtSTS MtCtx_GetAllIndexes(MtContext ctx, MtSize* numHandles, MtOid* indexes, MtOid classe);`
  `MtSTS MtCtxMGetAllEntryPointDictionaries(MtContext ctx, MtSize* numDicts, MtOid** dictionaries, MT_CONST MtChar* className);`
  `MtSTS MtCtx_MGetAllEntryPointDictionaries(MtContext ctx, MtSize* numDicts, MtOid** dictionaries, MtOid aClass);`
  `MtSTS MtCtxGetAllEntryPointDictionaries(MtContext ctx, MtSize* numDicts, MtOid* dictionaries, MT_CONST MtChar* className);`
  `MtSTS MtCtx_GetAllEntryPointDictionaries(MtContext ctx, MtSize* numDicts, MtOid* dictionaries, MtOid aClass);`
- In the Enterprise Manager, in some cases stopping a large in-memory database may block all other operations until the database is offline.
- In some cases, forcing to stop a database during a restore operation may fail to indicate that the backup has not been fully restored.
- In some rare cases, adding thousands of entries at once into a large text index may fail.
- In some rare cases, adding thousands of objects into a relationship one at a time to over 60,000 objects modified into a single transaction may fail.
- In some cases, a `SQL DELETE` in a block statement may fail with `MATISSE-E-STMT_TOO_COMPLEX, MATISSE SQL statement too complex; too many statements in a block` when the class has a very large number of sub-classes.
- In some cases, a `SQL INSERT`, `UPDATE` or `DELETE` in a block statement may produce unnecessary index maintenance instructions.
- The `mt_server extendcache` command does not always return the expected error message when the new requested cache size goes beyond the memory available on the machine.
- The `mt_backup start --incremental` command now returns the `MTS_NOINCBACKUP` error instead of having to run `mt_backup write` to receive the same error.
- The `mt_backup end` command includes a new `--cancel` (`-c`) option to force the cancellation of the active backup.
- The `mt_backup end` command may record in the backup journal log file (`.bjl`) an explicit `Backup Cancelled` record to indicate that all the saved files prior to the last completed backup are to be ignored.
- In some cases, the database server may fail to create large datafiles (> 8Gbytes) for an in-memory database.

Resolved in Matisse 9.0.3

- The Matisse products are now supported on Windows 8 and Windows Server 2012.
- The Matisse products are now supported on Mac OS X 10.8 Mountain Lion.
- The Matisse products on Mac OS X are fully compatible with the latest version of native LLVM compiler.
- The Objective-C binding for MacOS X works with ARC (Automatic Reference Counting).
- The `mt_xml` utility with the `import` command provides the `-parallel <n>` option to load in parallel XML documents.
- The new `mt_datafile` administration command has been added to provide information about offline datafiles.
- In the Java binding, the URL protocol for the JDBC driver class respects the following format:
  ```
  jdbc:matisse://hostname[:port]/dbname
  ```
- The `'MATISSE-E-INVALTYPE, 72 is not a valid Matisse type'` error message returned when the argument of a built-in function is invalid has been improved.
  ```
  SELECT min(oid), max(MemberId) FROM MemberProfile;
  ```
- In some cases, loading all the objects with `MtCtxLoadNumObjects` in the Access-For-Update mode may not prevent a deadlock when it is expected.
- The `mt_dts` help provides an inaccurate description of the possible commands.
- In some cases, the multi-segment XML files produced by a XML export command does not respect the `<filename>_xds_<documentid>.xml` format.

Resolved in Matisse 9.0.2

- The new Objective-C binding has been added to the MacOS X packages.
- New C API functions have been added to support schema object namespaces
  ```
  MtSTS MtCtxMGetAllSubnamespaces(MtContext ctx, MtSize* numSubnamespaces, MtOid**
  subnamespaces, MT_CONST MtChar* namespaceName);
  MtSTS MtCtx_MGetAllSubnamespaces(MtContext ctx, MtSize* numSubnamespaces, MtOid**
  subnamespaces, MtOid aNamespace);
  MtSTS MtCtxGetAllSubnamespaces(MtContext ctx, MtSize* numSubnamespaces, MtOid*
  subnamespaces, MT_CONST MtChar* namespaceName);
  MtSTS MtCtx_GetAllSubnamespaces(MtContext ctx, MtSize* numSubnamespaces, MtOid*
  subnamespaces, MtOid aNamespace);
  ```
- A new SQL pseudo-relationship has been defined on `MtNamespace` to improve description of namespace hierarchy.
  ```
  MtNamespace.MtAllSubnamespaces
  ```
- In the Enterprise Manager, the status bar message does not display the exact number of databases when a Host node is selected.

- In the Enterprise Manager Object Browser Panel, the Statistics tab is not always properly refreshed when switching databases back and forth.
- In the Enterprise Manager, restoring a backup into a database where the datafile size is smaller than the total of all the restored backup files may fail to complete indicating that the datafile is full.
- In rare cases, the `mt_xml` utility may return the `MATISSE_INDEXKEYEXISTS` error when loading an XML file. This error can occur only if an attribute defined in a sub-class with a default value participates in a multi-segment index.
- In SQL, the execution of an `INSERT` statement may not always return the message corresponding to the `MATISSE_NUMERICOVERFLOW` error code.
- The `mt_sql` utility returns an incorrect message when the result of a `SELECT` statement with navigation is saved into a selection:
  ```
  SELECT REF(c.myrel.Mycls2) FROM Mycls1 c INTO sel01;
  ```
- In replication mode, some very large non-embedded objects (i.e. video) may not be fully replicated on the replica server.
- In replication mode, the `MATISSE_NOREPLSQLDUI` error is not always returned when executing a server-side update on the master database.
- The commit function may take a long time to return a `MATISSE_IDXKEYEXISTS` error if the transaction has modified a very large number of objects with primary key indexes.

Resolved in Matisse 9.0.1

- The Matisse DBMS 32-bit and 64-bit as well as Matisse Lite products are available on MacOS X.
- The Enterprise Manager images have been updated to better integrate with the supported platforms.
- The Enterprise Manager Lite provides access to the log file in the Audit tab of the Host node.
- The Enterprise Manager Lite creates log and resource files named respectively `mt_emgrlite<username>.log` and `mtemgrliterc`.
- The `mt_server create` command allows you to create the configuration file for a new database.
- In some cases, running a backup in the Enterprise Manager may fail.
- In the Enterprise Manager Database Properties dialog, the check forbidding the creation of multiple datafiles at the same location is not always properly enforced.
- All the namespaces defined in the database are not always listed in the pull-down menu of the Meta-Schema panels.
- In some rare cases, removing hundreds of entries at once from a large index or from a large relationship may fail.
- In SQL, the execution of a block statement with a loop retrieving objects in a `SELECTION` variable may fail with the `MATISSE_SQLSTACKOVF` error.

Resolved in Matisse 9.0.0

- The Meta-Schema adds the `MtNamespace` class to support the definition of namespaces for all schema objects.
- The Meta-Schema adds the `MtMetaClass` class which is the super-class of user-defined meta-classes. This defines a straightforward mechanism for developers to extend the Meta-Schema.
- The `MtDomain` and `MtTrigger` classes are no longer public classes of the Meta-Schema.
- In the Meta-Schema, the `MtDocumentation` attribute has been removed from the `MtMethod` class.
- The Array datatypes (`MT_ARRAY_*`) which were not supported by most of the language bindings are no longer public types.
- Schema names follow the naming conventions defined for identifiers or symbols in the supported programming languages.
- The `MT_TEXT` data type generates non-embedded objects for large values which is the current behavior for the list datatypes (`MT_LIST_*`) as well as for bytes datatypes (`MT_BYTES`, `MT_IMAGE`, etc.).
- SQL adds the `MtFullName` and `MtFullClassName` pseudo-attributes to support namespaces.
  ```
  MtObject.MtFullClassName
  MtClass.MtFullName
  MtAttribute.MtFullName
  MtRelationship.MtFullName
  MtIndex.MtFullName
  MtEntryPointDictionary.MtFullName
  MtNamespace.MtFullName
  ```
- New SQL pseudo-relationships have been defined on `MtClass` to improve class description of classes with inheritance.
  ```
  MtClass.MtAllAttributes
  MtClass.MtAllRelationships
  MtClass.MtAllInverseRelationships
  MtClass.MtAllSuperclasses
  MtClass.MtAllSubclasses
  MtClass.MtAllMethods
  ```
- In the Enterprise Manager, the `Tutorial` and `Online Documentation` menu-items in the `Help` menu have been added to improve the developer's experience.
- In the Enterprise Manager, the Manage License menu-item in the Help menu has been added to check and update the product license key from the tool.
- The maximum capacity of the datafiles handled by the database server has been extended. The database server can manage up to 31 datafiles on individual disks or RAID-enabled disks.
- The command lines help for `mt_sdl`, `mt_xml` and `mt_dts` has been updated to be harmonized with the other administration commands.
- The `mt_sdl` utility with the `parse -ddl` command now parses the syntax of any type of SQL DDL script.

- The license key format has been extended to provide more appropriate control for Matisse servers running on Virtual Machines.
- The Java binding has been extended to support Java 7.
- New C API functions have been added to support schema object namespaces

  ```
  MtSTS MtCtxGetNamespace(MtContext ctx, MtOid* nspace, MT_CONST MtChar* nspaceName);
  MtSTS MtCtxGetObjectNamespace(MtContext ctx, MtOid* nspace, MtOid schemaObject);
  MtSTS MtCtxMGetObjectFullName(MtContext ctx, MtChar** fullName, MtOid
  schemaObject);
  ```

- Existing C API functions have been extended to support schema fully qualified names

  ```
  MtSTS MtCtxGetClass(MtContext ctx, MtOid* aClass, MT_CONST MtChar* className);
  MtSTS MtCtxGetAttribute(MtContext ctx, MtOid* attribute, MT_CONST MtChar*
  attributeName);
  MtSTS MtCtxGetRelationship(MtContext ctx, MtOid* relationship, MT_CONST MtChar*
  relationshipName);
  MtSTS MtCtxGetIndex(MtContext ctx, MtOid* index, MT_CONST MtChar* indexName);
  MtSTS MtCtxGetEntryPointDictionary(MtContext ctx, MtOid* dictionary, MT_CONST
  MtChar* dictName);
  ```

- The C++ binding documentation has been improved in term of completeness and usability.
- The Eiffel binding has been enhanced to support Eiffel 64-bit and as well as to be compatible with Eiffel Studio 7.
- The `mt_sdl` utility fails to update the Java file for a class that includes a user-defined Java method containing format string such as in the statement below:

  ```
  String.format("%d-%s", empId, empName);
  ```

- In some case the SQL compiler may fail to produce the most efficient execution plan when an attribute defined in a super-class in used in multiple indexes defined in sub-classes and that the attribute is selected through a relationship.
- SQL pseudo-attributes `OID`, `CLASS_NAME` and `CLASS_ID` defined on `MtObject` are deprecated and replaced by respectively `MtOid`, `MtClassName` and `MtClassOid`.
- In some cases, the compilation of a SQL block statement may return error messages that are not specific enough to help correcting the script.
- In some cases, the error line reported by the `mt_xml` utility when a parsing error occurs may be off by few lines.
- In some cases, the server may fail to extend a datafile beyond 64Gbytes.
- In some cases, `DEADLOCKABORT` errors returned by the database server may lead to a failure in the client application.
- In the Enterprise Manager, in some cases a full backup of a database into multiple slices does not calculate the slice size large enough to backup the full database preventing the restore of the backup to complete successfully.

5 Documentation

Matisse documents available on the Web

The following documents are available at `http://www.matisse.com/developers/documentation`:

- Installation guides for Linux, MacOS, Windows, and Solaris
- *Getting Started with Matisse*
- *Matisse SQL Programmer's Guide* (includes user's guide for `mt_sql`)
- *Matisse .NET Programmer's Guide* (and example applications)
- *Matisse Java Programmer's Guide* (and example applications)
- *Matisse Objective-C Programmer's Guide* (and example applications)
- *Matisse C++ Programmer's Guide* (and example applications)
- *Matisse C API Reference*
- *Matisse ODL Programmer's Guide* (includes user's guide for `mt_sdl`)
- *Matisse Modeler Guide*
- *Matisse Server Administration Guide*
- *Matisse XML Programming Guide* (includes user's guide for `mt_xml`)
- *Matisse Data Transformation Services Guide* (includes user's guide for `mt_dts`)

Documents included with Matisse standard installation

- Guide to Matisse documentation and other resources: `readme.html`
- *Matisse .NET Binding API Reference*: `docs/NET/MatisseNetBinding.chm`
- *Matisse Java Binding API Reference*: `docs/java/api/index.html`
- *Matisse Objective-C Binding API Reference*: `docs/objc/api/index.html`
- *Matisse C++ Binding API Reference*: `docs/cxx/api/index.html`

Open source bindings

- *Matisse Eiffel Programmer's Guide* (and example applications)
- *Matisse PHP Programmer's Guide* (and example applications)
- *Matisse Python Programmer's Guide* (and example applications)