precisely

Location Intelligence
Geographic Information Systems

Precisely MapInfo Data Access Library

Version 1.0

Developer Guide

6 - Spatial Objects and Coordinate Systems   94

7 - Working with GeoPackage   110

1 – Introduction to MapInfo Data Access Library

Welcome Developers to Precisely's latest offering for MapInfo Pro extensibility programming support for Microsoft's .NET Framework for Windows. The MapInfo Data Access Library (MDAL) reflects a single object model for developing Add-in extensions for MapInfo Pro.

The MDAL delivers the Data Access functionality of the MapXtreme SDK to the MapInfo Pro Add-in developer, providing a robust object model for creating, accessing and modifying MapInfo spatial tables from within the .NET Add-in. This provides the organization the flexibility of developing a spatial tool in .NET that integrates with MapInfo Pro.

In this chapter:

Overview of MapInfo Data Access Library (MDAL)

The MapInfo Data Access Library (MDAL) is a set of classes and interfaces that allow .NET developers to create MapInfo Pro add-ins that can easily create, search, and update MapInfo Tables and other supported database formats such as Oracle, SQL Server and GeoPackage. The Library is based on a subset of the MapInfo MapXtreme ™ SDK product, mainly the Data Access functionality

> (i) Currently the MapInfo Data Access Library is only available for use with MapInfo Pro add-ins.

The following components and features are included in MDAL:

- Fully Capable Data Access Object Model – Create MapInfo Tables, Insert, Update, Delete, Select, Join multiple tables, Search using a well thought out API.
- Full MiSql Support – Note that there are differences from MapBasic Syntax.
- Complete Geometry Object Model – Supports efficient reading, creation and editing of all MapInfo Geometry types, including text objects. Also supports conversion to and from Well Known Binary, Well Known Text and GeoJson formats.
- Full Coordinate System Support.
- Thread-Safe – Can be used to create background tasks in MapInfo Pro or run processing on secondary threads.
- Supports Most Pro Data formats – not all. Ex: No spatial support for postgis
- Pro style transactions on MapInfo and MapInfo Extended tables - This allows for background threads to edit tables open in Pro and let the end user decide to commit or revert the changes.
- Extensive Documentation – API Reference Guide, User Guide, MiSql reference.

Samples and Examples

There are code snippets in the help topics to demonstrate how to use the new types.

Refer to the Sample Applications installed with MapBasic. You can run MapInfoDataAccessLibraryExamples to create a window that lets you click on an example in the User Interface for a property or methods. It then takes you to the debugger in the sample code.

Samples are placed at:

SAMPLES\RIBBONINTERFACE\DotNet\MapInfoDataAccessLibraryExamples.

Technical Support

Precisely offers unparalleled technical support for users of MapInfo software products. Our Technical Support department provides technical assistance to registered users of MapInfo software – so you don't need to be an expert in all aspects of our products in order to get results. See the Precisely Web site at http://support.precisely.com for information on the tech support offerings.

2 – Working with Data

The MapInfo.Data namespace contains the classes and interfaces that provide multiple ways of accessing data from a MapInfo Data Access Library (MDAL) application.

In this chapter:

Overview of MapInfo.Data Namespace

The MapInfo.Data namespace contains the classes and interfaces that provide multiple ways of accessing data from a MapInfo Data Access Library (MDAL) application. Within this namespace is the MapInfo ADO.NET data provider with a MapInfo SQL language for standard querying of databases and tables. The Feature object model is another way to access data that uses objects instead of SQL. The Catalog is the starting point for data access, containing methods for managing tables (open, close, create) and searching for data in a variety of ways.
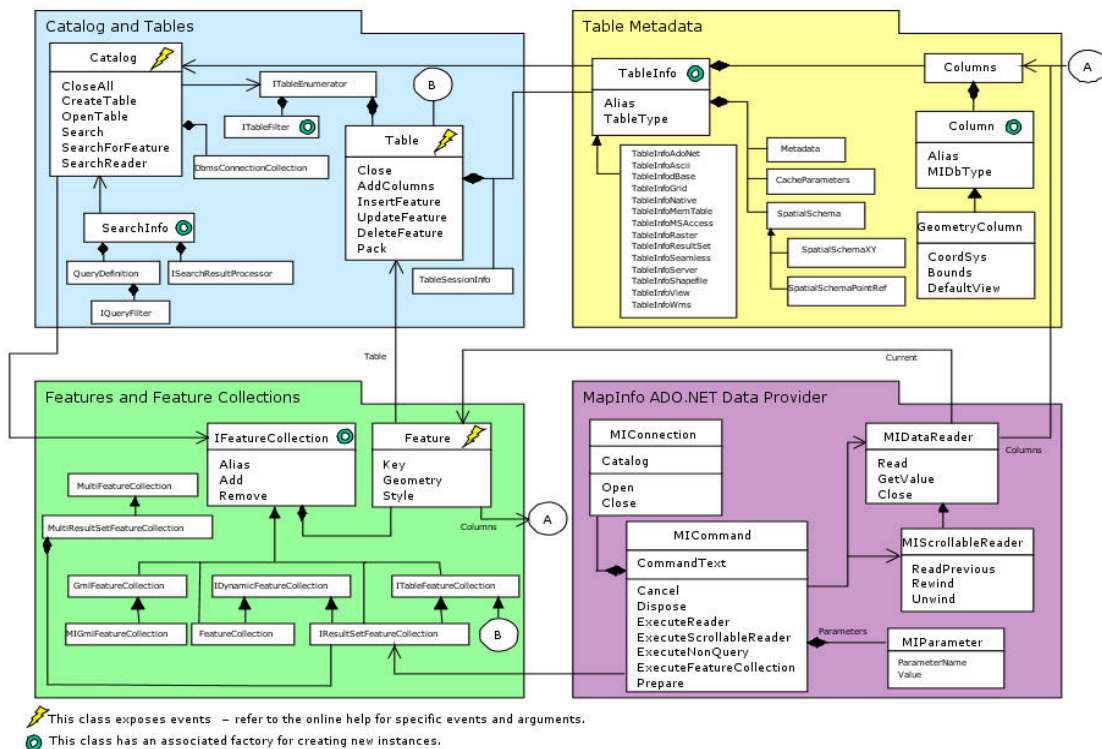
This chapter is organized to follow the MapInfo Data Access Library (MDAL) Data Access Model diagram above, and includes these topics:

• Catalog and Tables
• Supported Table Types
• Table Metadata (TableInfo)
• MapInfo ADO.NET Data Provider
• Features and Feature Collections

Data access is central to any MDAL application, and covers a wide variety of topics. Within the topics listed above are other important topics of information that should not be overlooked.

Following this chapter are two additional chapters related to data access: Chapter 4 Creating Expressions, and Chapter 5 Accessing Data from a DBMS.

Catalog and Tables

Catalog is the manager of the MapInfo Data Access Library (MDAL) data access model. Tables are a fundamental unit of MapInfo Data Access Library (MDAL). Tables hold the data that you want to display and analyze in your application. The Catalog, as manager, holds a list of tables that are currently open in the session. Tables are also opened, created and closed from the Catalog.

Nearly all of MDAL's data access operations involve the Catalog and tables.
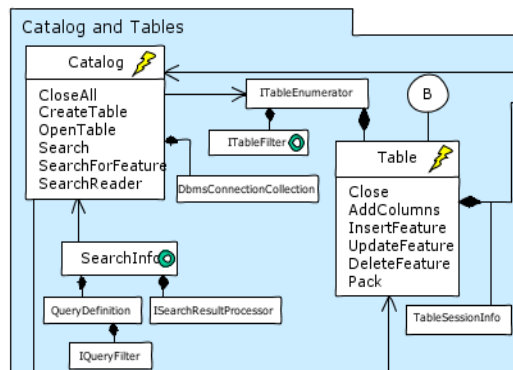
Tables

The Table class is the basic unit of all data access. Table, Column, and all TAB file metadata information is accessible from a MapInfo Table. Tables may be mappable (contain a column of type FeatureGeometry) or non-mappable.

Table Aliases

When tables are opened, they can be assigned a name (or alias) which is used while the table is open for referencing the table. For example, the table may be referred to by its alias in SQL statements. A table that is opened from a TAB file is assigned a default alias if no alias is specified. The default alias is based upon the name of the TAB file. This property is optional and may be set to null. However, it is good practice to assign an alias.

Columns

A Column object identifies the properties of a column in a table, feature, or feature collection and specifies the column's name (alias), data type, width (for string and decimal columns), and other properties of the column.

Supported data types include:

Data Type	Description
Int	Provides a 32-bit signed integer. This maps to the .NET Framework datatype Int32.
SmallInt	Provides a 16-bit signed integer. This maps to the .NET Framework datatype Int16.
Double	A floating point number within the range of -1.79E +308 through 1.79E +308. This maps to Double.
dBaseDecimal	Provides a floating point number which is treated internally the same as a Double. The dBaseDecimal has a fixed precision and scale when persisted in a table. This is a legacy data type derived, as its name suggests, from the dBase file formats. This maps to Double.
Boolean	Provides a boolean value. This maps to Boolean.
String	Provides a variable-length, null terminated UNICODE string value. This maps to String.
Date [*][†]	Provides a date value. The Date type is implemented as a structure in the MapInfo.Data namespace.
DateTime [*][‡]	Provides a combined date and time value. The DateTime type is mapped to System.DateTime.

Data Type	Description
Time	Provides a time value. Supports the Time type in MapInfo Professional tables (TAB files) version 9.0 and later. The Time type is implemented as a structure in the MapInfo.Data namespace.
FeatureGeometry	Provides a FeatureGeometry.
Binary	Provides an array of binary data. This maps to an Array of Byte values.
Key	Provides a key from a table. This is the data type of the Key pseudo column on a Table.
CoordSys	Provides a coordinate system. This type exists only for the purposes of binding a coordinate system object to an MICommand for functions which require the specification of a coordinate system.
Style	Provides an instance of a Style class. See MapInfo.Styles.Style. This is the data type of the Style object stored in the style column on a Table.
Raster	Provides a RasterInfo from table's raster column. This is the data type of the RasterInfo object stored in raster column on a Table.
Grid	A GridInfo from table's grid column. This is the data type of the GridInfo object stored in grid column on a Table.
Wms	Provides a WmsClient from the table's Wms column. This is the data type of the WmsClient object stored in the Wms column on a Table.
TileServer	Provides a TileServerInfo from the table's raster column. This is the data type of the TileServerInfo object stored in the raster column on a Table.

*   To ensure backward compatibility with earlier versions of MDAL, the MapInfo.Data.MIDataReader.GetDateTime method works with both the DateTime and Date types. In both cases, a System.DateTime value is returned. However, the MapInfo.Data.Column.DataType will reflect the actual data type, either Date or DateTime.

† The Time and DateTime types are not supported for MapInfo SQL functions. However, in MapInfo SQL functions that call/use a DateTime type, the function will return the date portion of the DateTime value. Please see the MapInfo SQL Reference for more information.

‡ The Time and DateTime types are not supported for MapInfo SQL functions. However, in MapInfo SQL functions that call/use a DateTime type, the function will return the date portion of the DateTime value. Please see the MapInfo SQL Reference for more information.

Time and DateTime Data Source Support

MapInfo Data Access Library (MDAL) can read Date, DateTime, and Time data (and save it back if applied) on the supported data sources and data providers. The different data sources may have different type definitions on date/time, which may or may not match MapInfo Data Access Library (MDAL) types exactly.

The new data types are supported for the following data sources:

- Mem tables
- Native tables (TAB files)
- ADO.NET
- Oracle via OCI
- MS SQL Server via ODBC

The ASCII and dBase, and Microsoft Access data sources are not supported.

Date and DateTime Support in Remote Databases

Remote databases may not support all the data types that MapInfo Data Access Library (MDAL) supports. The table below shows the date and time-based types supported in native TAB files and in each supported database.

MDAL	Native(X) TAB Files	ADO.NET	Oracle (OCI)	MS SQL Server	GeoPackage
Date	Date		Date		
Time	Time				
DateTime	DateTime	DateTime	DateTime	DateTime	DateTime

The following sections provide you with MI_Key, MI_Geometry, and MI_Style column information.

MI_Key

All tables have a pseudo column named MI_Key which returns instances of Key. The MI_Key pseudo column is similar in concept to the rowid pseudo column in MapInfo Professional and MapBasic. Unlike rowid, this column is not a numeric column. A Key instance may be converted to or from a string literal.

MI_Geometry

A Geometry column object in a table, feature, or feature collection contains FeatureGeometry objects and specifies properties such as the coordinate system of the column and the entire bounds of all the geometry objects it contains.

Geometry columns for most table types are given the name "Obj". To be compatible with previous versions of MapX and MapInfo Professional, the alias "Obj" is resolved to the first GeometryColumn in the table. Additionally, the alias "MI_Geometry" may also be used for any table to refer to the same column that "Obj" refers to.

MI_Style

Tables with a Geometry column also have a column with the name "MI_Style", or if not found, from the first column with type MIDbType.Style. This column is used to hold the style information for Geometry objects such as line width for polygons and symbol size for points. This column cannot be updated independently. The Style and Geometry columns must be updated at the same time.

The MI_Style column is created automatically when you are opening a table in MapInfo native format (.TAB). For all other table types, you must specifically create the column. If you use MapInfo.Data.ColumnFactory.CreateStyleColumn it will create a column with the name (alias) of "MI_Style" and a data type of MIDbType.Style.

When using MISQL to insert rows into a table, be sure to include the MI_Style column in the insert statement. See the code example below:

```
Table tab = MapInfo.Engine.Session.Current.Catalog.GetTable("MapViewer");
        TableInfo ti = TableInfoFactory.CreateTemp("Test",
((MapInfo.Data.GeometryColumn)tab.TableInfo.Columns["Obj"]).CoordSys);
        Table tabTemp = MapInfo.Engine.Session.Current.Catalog.CreateTable(ti);

MIConnection conn = new MIConnection();
conn.Open();
MICommand comm = conn.CreateCommand();
comm.CommandText = "Insert Into " + tabTemp.Alias +
   " (Obj, MI_Style) SELECT MI_Point(MI_X(Obj), MI_Y(Obj), '" +
   ((MapInfo.Data.GeometryColumn)tab.TableInfo.Columns["Obj"]).Coord
   Sys.SrsString + "'), MI_Style" + " FROM " + tab.Alias + " WHERE msaname
   = 'Minneapolis-St. Paul, MN-WI' AND Not Obj = Null";
MessageBox.Show(comm.CommandText);
```

```
int numChanged = comm.ExecuteNonQuery();

mapControl1.Map.Layers.Add(new FeatureLayer(tabTemp));
mapControl1.Map.SetView(mapControl1.Map.Layers["Test"] as FeatureLayer);
```

Catalog

The Catalog is essentially the manager of the MapInfo Data Access Library (MDAL) data access model. The Catalog holds a list of tables that are currently open in the MDAL Session. Tables are also opened, created and closed from the Catalog. The Catalog can be thought of as a single database holding all the tables opened in it, regardless of their actual data source.

Each MDAL Session manages a single Catalog.

Catalog initially contains no tables. When a table is opened, an alias (or name) is assigned to the table or provided by the caller. The alias is used to identify the table in queries and other operations.

Tables can be mappable (contain a spatial component) or be non-mappable and contain only data columns. The MapInfo Data Access Library (MDAL) Catalog can open both types and use either in queries and joins.

Catalog provides facilities for creating new table definitions and enumerating through tables which are currently opened. Catalog also contains search methods that can be used to access data in open tables.

The Catalog has an SQL engine that allows you to select, insert, update, and delete tables and data within tables. The SQL engine allows you to join any tables defined in the catalog (for example, Native to SQLServer, or SQLServer to Oracle). The Catalog handles the integration from various sources so you don't have to. This is a powerful tool when organizing data from various sources.

The MDAL Catalog is exposed through the MapInfo ADO.NET Data Provider. Access to tables and result sets is controlled through this interface. See MapInfo ADO.NET Data Provider.

Code Sample

The following example illustrates how to access the Catalog through the MDAL Session object, open some tables and enumerate through all the tables in the Catalog followed by only the editable tables in the Catalog.

VB example:

```
Public Shared Sub MapInfo_Data_Catalog()
    ' Catalog is accessible off the Session object
```

```
    Dim catalog As Catalog = Session.Current.Catalog

    ' Open a bunch of tables
    Dim table As Table = catalog.OpenTable("States.tab")
    table.SessionInfo.ReadOnly = True ' Make states ReadOnly
    table = catalog.OpenTable("world.tab")
    table = catalog.OpenTable("worldcap.tab", "World Capitals")

    ' Enumerate the catalog directly - includes All tables
    Dim t As Table
    For Each t In catalog
        Console.Out.WriteLine("Table : {0}", t.Alias)
    Next
    Console.Out.WriteLine()

    ' Now enumerate through only tables that are editable (not ReadOnly)
    Dim tEnum As ITableEnumerator = _
    catalog.EnumerateTables(TableFilterFactory.FilterEditableTables())
    While tEnum.MoveNext()
        Console.Out.WriteLine("Table: {0}", tEnum.Current.Alias)
    End While

    Session.Current.Catalog.CloseAll()
End Sub
```

Supported Table Types

One of the strengths of MapInfo Data Access Library (MDAL) is its ability to access data "where it lives." This means we strive to handle a wide variety of data formats. Here are the supported table types in MapInfo Data Access Library (MDAL):

MapInfo .TAB format	MapInfo native table format.
	This file-based table may have an associated .MAP file containing FeatureGeometry and Style information. Non-spatial data is stored in a .DAT file. TAB is available as a storage format to be used when caching. See Creating a New Table.
dBase	Data stored in a dBase file.
	The table may have an associated .MAP file containing FeatureGeometry and Style information. Non-spatial data is stored in a .DBF file. An associated .IND file holds one or more B-Tree indices for non-spatial attribute values (strings, numbers, and dates)

ASCII	Data stored in a delimited .CSV or text file. The maximum string length is 255 characters (including up to two quotation marks). ASCII tables are Insert only.
	The table may have an associated .MAP file containing FeatureGeometry and Style information. Non-spatial data is stored in a .CSV or .TXT file.
MS Access	Microsoft Access database table.
	This file-based table located inside of a Microsoft Access .MDB database may have an associated .MAP file containing FeatureGeometry and Style information. Non-spatial data is stored in an Access file.
Shapefile	An ESRI Shapefile table.
	These tables are read-only and support three-dimensional geometries (X, Y, Z, M). Non-spatial attribute data is stored in .DBF file format. FeatureGeometry values stored in ESRI .shp file format. MDAL does not have access to the spatial index. Caching is supported as a .MAP file that can be temporary or persistent. A persistent cache can be shared with MapInfo Professional. It is controlled by the PersistentCache property on the TableInfoShapefile class.
MemTable	An in-memory storage of non-spatial attribute data.
	FeatureGeometry data and indices are stored on disk. These are temporary tables—all data is lost when the table is closed. MemTables are serializable. The data can be persisted in workspaces (data only; keys may be changed when reloading). This table type is available as a storage format to be used when caching. There is no .TAB file equivalent to a MemTable. See Create a Temporary MemTable.

RDBMS Server	A spatial table stored in a remote database management system (such as SQL Server or Oracle). The table is defined by a native SQL SELECT statement. MDAL performs query parsing and modification. Caching is enabled by default. Supported protocols (toolkits) include: OCI (Oracle Spatial) and ODBC (, SQL Server, SpatialWare and XY). See Chapter 5 Accessing Data from a DBMS.
ADO.NET	A table of non-spatial data that is based upon an ADO.NET DataTable or IDbCommand. This table type supports many different data providers with provider-specific implementations. ADO.NET is the choice when there is no MDAL supported data provider. ADO.NET is designed to support both Connected (IDBCommand) and Disconnected (DataTable) ADO.NET models. IDBTables are read-only. Cache may be applied forcibly (implicit keys). DataTables are editable and run-time serialization is supported. See Using an ADO.NET Data Provider.
Raster	A table containing a raster image. This typically provides a base map for other spatial table types. Tables have only a single record and a fixed column schema (RasterInfo, MI_Geometry, MI_Style). These tables may be joined with vector tables using spatial predicates (for example, "within").
Grid	A table containing a grid image. This table type provides a base map for other spatial table types. Tables have only a single record and a fixed column schema (GridInfo, MI_Geometry, MI_Style). These tables may be joined with vector tables using spatial predicates (for example, "within"). GridRead class provides access to grid cell values. MapInfo.Raster.GridCreatorFromFeatures class creates a grid using an interpolator.

WMS	A table containing an image from a Web Map Service (WMS).
	This table type provides a base map for other spatial table types. Tables have only a single record and a fixed column schema (GridInfo, MI_Geometry, MI_Style). These tables may be joined with vector tables using spatial predicates (for example, "within"). WMS tables are accessed like dynamic raster through a MapInfo.Wms.WmsClient.
Seamless	A table that combines two or more base tables with contiguous geography. It displays as a single map layer.
	Seamless tables are specifically tuned for spatial queries, such as drawing a map, which uses seamless tables for optimally querying appropriate component tables. Component tables that make up a seamless table may be vector or raster. They must all have the same schema. They are read-only. The underlying component tables cannot be modified directly. Sorting and aggregating operations examine every record of every component tables (could be have a significant performance impact when working with vector tables.)
View	A view based on a MapInfo SQL Select statement (not a native SQL supported by Server tables). See View Tables.
ResultSet	A table containing the results of a search. ResultSet is used exclusively for IResultSetFeatureCollections. See Result Sets.
TileServer	A table containing a TileServer image. This typically provides a base map for other spatial table types. Tables contain a single record and a fixed column schema (TileServerInfo, MI_Geometry, MI_Style).

Geopackage	A table containing information in a Geopackage format.
	Indicates a table that has both FeatureGeometry objects and attribute data stored in an OGC Geopackage database file format.
NativeX	MapInfo Extended (NativeX) Tab file formats.
	This file-based table may have an associated .MAP file containing FeatureGeometry and Style information. Non-spatial data is stored in a .DAT file. TAB is available as a storage format to be used when caching.
	The NativeX format supports table caches larger than 2GB in size and character sets UTF-8 and UTF-16.

Working with Catalog and Tables

This section covers some basic table operations, including:

- Locating Open Tables
- Closing a Table
- Packing a Table
- Listening to Table and Catalog Events

See also the MapInfo.Data.Table class in the MapInfo Data Access Library (MDAL) Developer Reference.

Locating Open Tables

To locate open tables, you must enumerate the catalog. This is done by using the methods in the following sections.

Catalog.GetTable

The MapInfo.Data.Catalog.GetTable method returns the Table object referenced by the TableAlias parameter. This must be a table which has already been opened. If no such table is found (or the table has subsequently been closed), then the method returns null.

Catalog.Item (Indexer)

MapInfo.Data.Catalog.Item property can be used as an indexer for locating a Table by its Alias. This is functionally equivalent to using the Catalog.GetTable method but generates code that is easier to read. The Alias must specify a table which has already been opened.

VB example:

```
Public Shared Sub MapInfo_Data_Catalog2()

    Dim tbl As Table
    For Each tbl In Session.Current.Catalog
        System.Console.WriteLine("Table: " + tbl.Alias)
    Next

End Sub
```

TableEnumerators

Table enumerators may be obtained through the various overloaded EnumerateTables methods. A table enumerator may be created with a filter. The filter determines which tables are actually included in the enumeration while the enumerator simply provides the mechanics of enumeration. You can create your own table filters to use in the TableEnumerator. You can also create your own table enumerator by implementing the ITableEnumerator interface.

VB example:

```
Public Shared Sub MapInfo_Data_Catalog3(ByVal catalog As Catalog)
    Dim te As ITableEnumerator = _
catalog.EnumerateTables(TableFilterFactory.FilterEditableTables())

    While te.MoveNext()
        Dim tbl As Table = te.Current

    End While
End Sub
```

Closing a Table

Three methods are available to close tables. MapInfo.Data.Catalog.CloseAll closes all open tables while Catalog.CloseTable closes a single, open table. The Table class also has a Close method.

Packing a Table

The MapInfo.Data.Table.Pack method removes records from the table that were previously marked for deletion. When the table is packed, the table's TablePacked event is raised. The arguments for the event indicate whether or not the table's keys changed as a result of the pack (which would be caused by removing deleted records). Keys only change if the PackType includes RemoveDeletedRecords and if there actually were deleted records in the middle of the table. If the only deleted records in the table are at the end of the table, then no keys are changed. The event does not indicate that keys were changed.

ⓘ Since ResultSet tables hold collections of keys, these are vulnerable to pack operations on the table from which they were derived. The ResultSet is no longer valid if the keys have changed.

PackType Enumeration provides you with the following options.

- PackGeometry – Indicates that the geometry objects are packed. Packing the objects attempts to remove as much unused space as possible. A fully packed RTree (the spatial index used to spatially access the geometry objects) may reduce performance by causing many more unnecessary reads. To balance disk space and processing speed, packing the geometry objects may continue to leave some unused space in the RTree. Also note: a packed RTree results in a slight performance penalty for insert and update operations as there is a higher likelihood that the RTree needs to be expanded.

- RebuildGeometry – Rebuilding the geometry objects removes unused space that has resulted from a series of insert, update, and/or delete operations. Unlike packing the geometry objects, this option intentionally leaves unused space in the RTree index to improve the performance of future insert and update operations.

- PackIndex – Non-spatial indices are maintained as B*trees. These structures do not always have filled internal or leaf nodes. This is intentional by default to allow room for the index to accommodate insert and update operations without requiring a significant restructuring of the index. The unused space may be exacerbated by the occurrence of insert, update, or delete operations. Packing an index fully packs every internal and leaf node (except possibly the "last" node). This option reduces the disk space used by the index as much as possible and also improves the read-performance of the index. There is a performance penalty for insert and update operations on a fully packed index.

- RebuildIndex – Rebuilding an index does not fully pack the internal and leaf nodes like the PackIndex option. Instead, rebuilding an index recreates the index with the amount of unused space that is intentionally put into the index to balance disk space, read performance, and modify performance. After several modification operations, an index may contain a considerable amount of unused space. This option regains that unused space.

- RemoveDeletedRecords – Some data sources, including MapInfo Native and dBase data sources, do not physically remove records when they are deleted. To physically remove the deleted records, the table must be packed with this option specified. The record number is typically used as the record key for these data source types. Removing deleted records from a table may cause keys to become invalid since they may change as a result of the pack.

- CompactDb – If the table's data source is Microsoft Access (TableType of Access), then the MDB file containing the table's data may also be compressed using the Pack method and specifying this option.

- All – This is a convenience option that is equivalent to PackGeometry | PackIndex | RemoveDeletedRecords.

Listening to Table and Catalog Events

Table exposes several events which applications may subscribe to. They are:

- RowInsertedEvent – Occurs when a new row is added to the table.
- RowUpdatedEvent – Occurs when an existing row in the table is updated.
- RowDeletedEvent – Occurs when a row in the table is deleted.
- TablePackedEvent – Occurs when the table is packed.
- TableCloseRequestEvent – Occurs when the table has been asked to close.
- TableIsClosingEvent – Occurs when the table is closing.
- TableClosedEvent – Occurs when the table is closed.

Catalog also exposes the following events.

- TableOpenedEvent – Occurs when a table is opened.
- TableCreatedEvent – Occurs when a new table is created.
- TableIsClosingEvent – Occurs when the table is closing.

Table Metadata (TableInfo)

The TableInfo class in the MapInfo.Data namespace is an abstract base class that contains information, or metadata, about an existing table, including:

- Columns – number, names, data types, etc.
- Table alias, and description and pathname of the data source.
- Client metadata (the information between the begin_metadata/end_metadata tags in the TAB file).

TableInfo is used to open tables and create new tables. It is also used for retrieving information about the open table.

Classes that derive from TableInfo include provider-specific metadata. There is a TableInfo implementation for every MapInfo Data Access Library (MDAL) supported table type. See Data Sources.

TableInfo instances may be constructed manually, or from a .TAB file definition (without opening the table), as shown below.

```
TableInfo.CreateFromFile(…)
```

TableInfo contains properties for enabling Table Services, including caching and making a table mappable via a spatial schema. See Working with the Cache and Making Tables Mappable.
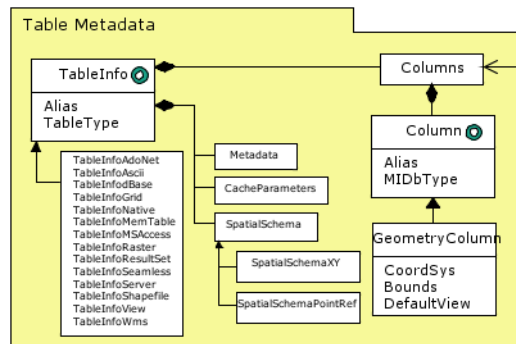
MapInfo Data Access Library (MDAL) provides table column metadata support for M and Z values. This feature is useful when you want to know whether geometries of a particular data provider can support 3D and Measured values without evaluating its individual geometries.

Metadata for a table can be accessed from the table's TableInfo property. From the table metadata you can access the GeometryColumn to interrogate if the table supports M or Z values and what the range of values for that table is if the range is known. For more information on support for M and Z values, see Support for M and Z Values.

Examining TAB File Metadata

TAB file metadata is accessible and editable. The TableInfo class can be obtained from the Table to get information about the table structure.

The following code demonstrates how to get the metadata for an open table. The code also demonstrates how the geometry column can be used to determine the coordinate system and bounds of the table. For a code example that returns M and Z values, see MapInfo.Data.TableInfo in the Developer Reference.

VB example:

```
Public Shared Sub MapInfo_Data_TableInfo2()
  ' Get the metadata for an open table
  Dim ti As TableInfo = Session.Current.Catalog("states").TableInfo

  ' Print out some information to the console
  Console.Out.WriteLine("Table Alias={0}, Datasource={1}, _
    Description={2}, Type={3}", _
  ti.Alias, ti.DataSourceName, ti.Description, ti.TableType)

  ' Print out some information about each column
  Dim col As Column
  For Each col In ti.Columns
  Console.Out.WriteLine("Column {0} Type={1} Width={2}", _
    col.Alias, col.DataType, col.Width)
  ' If the column is a geometry column, print csys and bounds.
  If col.DataType = MIDbType.FeatureGeometry Then
  Dim geocol As GeometryColumn = col
  Dim csys As MapInfo.GeomeTry.CoordSys = geocol.CoordSys
  Console.Out.WriteLine("CSys : {0}", csys.MapBasicString)
  Dim dr As MapInfo.GeomeTry.DRect = geocol.Bounds
  Console.Out.WriteLine("Bounds=({0},{1}),({2},{3})", dr.x1, _
    dr.y1, dr.x2, dr.y2)
  End If
  Next

End Sub
```

Creating a New Table

The following sections illustrate how to create a permanent native table, a temporary native table, and a temporary MemTable.

Create a New Permanent Native Table

The MapInfo.Data.Table.TableInfo property for a MapInfo native table returns an instance of TableInfoNative. A native table is a MapInfo .TAB file. This class may be used to access properties that are specific to native table types. New instances of this class may be created and used to construct new tables. See also Data Sources.

Note the use of the ColumnFactory class. This is provided to help you know which arguments are necessary for different data types. For example, a geometry column requires a coordinate system.

VB example:

```
Public Shared Sub MapInfo_Data_TableInfoNative()
   Dim ti As TableInfoNative = New TableInfoNative("NewTable")
   ti.TablePath = "c:\data\Capitals.TAB"
   ti.Columns.Add(ColumnFactory.CreateIndexedStringColumn("Capital",_
      25))
   ti.Columns.Add(ColumnFactory.CreateStringColumn("Country", 30))
   ti.Columns.Add(ColumnFactory.CreateDoubleColumn("Pop_Grw_Rt"))

' Make the table mappable
   ti.Columns.Add(ColumnFactory.CreateStyleColumn())
   Dim Robinson As CoordSys = _
   Session.Current.CoordSysFactory.CreateFromPrjString("12, _
      62, 7, 0")

ti.Columns.Add(ColumnFactory.CreateFeatureGeometryColumn(Robinson))
' Note we do not need to (nor should we) add a column of type Key.
' Every table automatically contains a column named "MI_Key".
   Dim table As Table = Session.Current.Catalog.CreateTable(ti)

End Sub
```

Create a Temporary Native Table

VB example:

```
Public Shared Sub MapInfo_Data_TableInfo3(ByVal conn As MIConnection)
   Dim ti As TableInfoNative = New TableInfoNative("NewTable")
   ti.Temporary = True
   Dim col As Column

   col = New Column
   col.Alias = "FString30"
   col.DataType = MIDbType.String
   col.Indexed = True
   col.Width = 30
   ti.Columns.Add(col)

   col = New Column
   col.Alias = "FInt32"
   col.DataType = MIDbType.Int
   col.Indexed = True
   ti.Columns.Add(col)

   col = New Column
   col.Alias = "FInt16"
   col.DataType = MIDbType.SmallInt
```

```
      col.Indexed = True
      ti.Columns.Add(col)

      col = New Column
      col.Alias = "FDouble"
      col.DataType = MIDbType.Double
      ti.Columns.Add(col)

      col = New Column
      col.Alias = "FDateTime"
      col.DataType = MIDbType.Date
      ti.Columns.Add(col)

      col = New Column
      col.Alias = "FBoolean"
      col.DataType = MIDbType.Boolean
      ti.Columns.Add(col)
      ' Note we do not need to (nor should we) add a column of type Key.
      ' Every table automatically contains a column named "MI_Key".
      Dim miTable As Table = conn.Catalog.CreateTable(ti)
End Sub
```

Create a Temporary MemTable

The MapInfo.Data.Table.TableInfo property for a memory table returns an instance of TableInfoMemTable. This class may be used to access properties that are specific to memory table types. New instances of this class may be created and used to construct new tables.

Data in formats such as XML or GML from a Web service can be brought into the Catalog and used in this fashion. It can be converted to a MultiPolygon, LineString, Point, or other Geometry via the MapInfo Data Access Library (MDAL) API. MDAL then turns the Geometry into a FeatureCollection, and, in turn, saves it to a memTable or native TAB format.

This approach is also appropriate if you wish to make data available for use in MDAL, but not necessarily for map display.

MapInfo Data Access Library (MDAL) supports reading and writing Z and M values to MemTables. M values on MultiCurves allow you to carry out linear referencing operations and dynamic segmentation.

VB example:

```
Public Shared Sub MapInfo_Data_TableInfoMemTable()
   Dim ti As TableInfoMemTable = New TableInfoMemTable("NewTable")
' Note: The TablePath property does not apply - it can be set but it _
' is meaningless.

   ti.Columns.Add(ColumnFactory.CreateIndexedStringColumn("Capital",_
```

```
    25))
  ti.Columns.Add(ColumnFactory.CreateStringColumn("Country", 30))
  ti.Columns.Add(ColumnFactory.CreateDoubleColumn("Pop_Grw_Rt"))

' Make the table mappable
  ti.Columns.Add(ColumnFactory.CreateStyleColumn())
  Dim Robinson As CoordSys = _
    Session.Current.CoordSysFactory.CreateFromPrjString("12, 62, _
    7, 0")
  ti.Columns.Add(ColumnFactory.CreateFeatureGeometryColumn(Robinson))
' Note we do not need to (nor should we) add a column of type Key.
' Every table automatically contains a column named "MI_Key".
  Dim table As Table = Session.Current.Catalog.CreateTable(ti)
End Sub
```

Adding Expression Columns to a Table

Use the MapInfo.Data.Table.AddColumns method to add expression columns to a table. The form of AddColumns that takes a Columns object creates temporary columns based on expressions comprised of functions, operators, literal values, and other columns on the table. All instances of Column in the columns argument must have an expression string specified.

ⓘ   TableAddColumns is not supported for the following table types: Server, View, Seamless, AdoNet, ResultSet, or Drilldown. MapInfo Data Access Library (MDAL) checks for the table and throws an exception if it encounters one of these table types.

VB example:

```
Public Shared Sub MapInfo_Data_TableAddColumns(ByVal miTable As Table)
  Dim NewCols As Columns = New Columns
  NewCols.Add(New Column("PopDensity1990", "Pop_1990 / _
    MI_Area(Obj, 'sq mi', 'Spherical')"))
  NewCols.Add(New Column("PopDensity2000", "Pop_2000 / _
    MI_Area(Obj, 'sq mi', 'Spherical')"))
  miTable.AddColumns(NewCols)
End Sub
```

The expression string "Pop_1990 / MI_Area(Obj, 'sq mi', 'Spherical')" represents derived information that will be placed in the temporary column. It says 'For each record divide population by area in square miles to yield the population density.' The SQL function MI_Area () in the expression will derive the area from the geometry of the record.

Using the AddColumns method may offer performance improvements in desktop applications where the join can be performed once, rather than on each subsequent access (as in the case of a view).

For more information and code examples, see the MapInfo.Data.Table.AddColumns class in the Developer Reference Help system.

For more information on creating expressions, see Chapter 4 Creating Expressions.

Data Sources

The following table lists the data sources supported by MapInfo Data Access Library (MDAL). Each type of data source is accessed by a specific data provider called TableInfo class, that is derived from MapInfo.Data.TableInfo. For a short summary of each data type see Supported Table Types.

Data Source	Class
Native (MapInfo.TAB)	TableInfoNative
dBase	TableInfodBase
MS Access	TableInfoMSAccess
ASCII	TableInfoAscii
RSBMS Server	TableInfoServer
ESRI Shapefile	TableInfoShapefile
Seamless	TableInfoSeamless
Raster	TableInfoRaster
Grid	TableInfoGrid
WMS	TableInfoWMS
ADONET	TableInfoAdoNet
MemTable	TableInfoMemTable
View	TableInfoView
ResultSet	TableInfoResultSet
TileServer	TableInfoTileServer
GeoPackage	TableInfoGeoPackage
NativeX	TableInfoNativeX

Choosing the Correct Data Source

Choosing the correct data source can make a difference in your application's performance. In some cases you will not have a choice, such as native MapInfo files (.TAB), but in other cases there may be multiple choices. In most cases, you will be using a supported data provider for the data source. In cases where the data is not accessible through one of these, you may be able to use the MapInfo ADO.NET data provider. This is the same data provider interface that the Catalog uses to retrieve data.

Each data source has certain performance characteristics. Native tables offer the best access and map drawing times. Data is stored locally on the system and optimized for your current operation. Other file-based table types perform well, depending on current hardware and file size.

Methods for Accessing Data

MapInfo Data Access Library (MDAL) provides several ways to bring data into the Catalog:

- Direct access to data sources
- Access via an ADO.NET data provider (TableInfoAdoNet)
- XML/GML from third-party web services

The best method to access data is to open it directly using one of the TableInfo classes that are specific to where your data resides.

Use the second method (TableInfoAdoNet) to access data that is not internally supported but has an ADO.NET provider.

A third method allows developers to integrate data to the Catalog who may interact with HTTP services that return XML or GML.

Direct Access to Data Sources

MapInfo Data Access Library (MDAL) provides native support for accessing data stored in file-based table formats and RDBMS servers, such as SQL Server and Oracle. In the case of file-based access, provide the path and filename in the appropriate TableInfo instance (TableInfoNative, TableInfodBase, TableInfoMSAssess, TableInfoAscii, TableInfoGeopackage, and TableInfoNativeX).

For direct access to data stored in RDBMS serves, use the TableInfoServer class to define the connection string and an SQL statement to execute on the remote table. Internally, MDAL uses ODBC or OCI to access the remote database.

TableInfoServer will open a connection to the server, query the table's metadata, and create the appropriate table definition with any spatial characteristics that are defined on the remote server. This tends to be the best performing method with remote data. Internally, MDAL can access only the data necessary to perform the current operation. During a map draw, MapInfo Data Access Library (MDAL) will construct a query that returns only the geometry column, and not the data columns. This minimizes the network traffic. If caching is on, then this is only an issue for the first access, since all subsequent requests will come from the cache. See Chapter 5 Accessing Data from a DBMS.

Access via an ADO.NET DataProvider

The second data access method is to use an ADO.NET data provider. This requires the definition of ADO.NET classes for data retrieval. Only non-mappable tables may be supplied as an AdoNet table. Non-mappable tables are those that do not contain geometry information about the data. Tables retrieved from an ADO.NET provider, however, can be made mappable by applying a SpatialSchema to the table definition. In this method, the MDAL engine calls the ADO.NET data provider whenever data is requested by a user. This tends to be a slower method of accessing data. However, when used in conjunction with caching, it performs well. See Using an ADO.NET Data Provider.

Data from Third-Party Web Services

MapInfo Data Access Library (MDAL) can integrate Web service XML or GML output into the Catalog for use in a MDAL desktop or web application. Data can be brought into the Catalog and converted to a MultiPolygon, LineString, Point, or other Geometry via the MapInfo Data Access Library (MDAL) API. MDAL then turns the Geometry into a FeatureCollection, and, in turn, saves it to a memTable or native TAB format.

This approach is appropriate also if you wish to make data available for use in MDAL, but not necessarily for map display.

Data Readers, MemTables and Result Sets

The methods to access data return a data reader or result set. A data reader allows access in a sequential manner and does not store copies of data. It retrieves the data from the data source, except in the case where the data source is cached. Result sets are collections of keys. These keys allow you access back to the original tables and do not create copies of the data.

A MemTable also allows you to store data from various sources into one table. This table type stores data in a combination of memory arrays and temporary disk storage. When data is added, the MemTable makes a copy of the data and does not have a key or

pointer back to the original table. These are useful for temporary layers for maps and containers for return values of processes such as a geocoding or routing result. MemTable access and map rendering performance is equivalent to native tables.

Result sets are a great tool when you need access to a defined set of rows and when you need to get data from the source. If the source data may change during your session then this method allows you to see the results if the data source supports concurrent access. Since MemTables are copies of data they are a static set of data rows and will not reflect changes from the original data sources.

Using an ADO.NET Data Provider

Data that cannot be directly accessed with a specific TableInfo data source can use TableInfoAdoNet. The ADO.NET table can be in one of two forms: DataTable (a collection of rows from a single table kept in-memory and allows read-write access); or IDbCommand (an SQL statement executed at the data source that yields read only, dynamic data).

Accessing Data in a DataTable

When using a DataTable, the Catalog is essentially holding on to a reference to the DataTable you supply to the call to Catalog.OpenTable (using the TableInfoAdoNet class). DataTables are editable using the MapInfo ADO.NET Data Provider by issuing Insert, Update, and/or Delete commands. Your application may continue to access the DataTable directly as well. Note, however, that the structure of the table should not be changed while the Catalog has a reference to it. Also note that changes to the data outside of the MapInfo Data Provider (e.g., without using the MICommand to issue Insert, Update, or Delete commands) will not result in the raising of the insert, update, or delete table events.

The DataTable contains almost enough information for the Catalog to define the table. For string columns, however, the Catalog needs to assign a length to this field. The length would be used when constructing temporary indices, temporary tables for aggregation, etc. For these types of operations, it is important to get the string length correct. The DataColumn has a MaxLength property that should be set to indicate the maximum length string the column could hold. If not set, this value defaults to -1 in which case the value of 254 is used. Before checking the MaxLength property, the Catalog looks to see if the DataColumn has a property defined in its ExtendedProperties collection with the name "StringWidth". If found, the value for this property is used as the column's width.

This example illustrates how to create a MapInfo Table whose data is stored in a DataTable.

VB example:

```
Public Shared Sub MapInfo_Data_TableInfoAdoNet(ByVal connection As _
    MIConnection)
  ' Create a new DataTable.
  Dim dt As DataTable = New DataTable("CityData")
  Dim dc As DataColumn
  dc = dt.Columns.Add("City", Type.GetType("string"))
  dc.MaxLength = 30
  dc = dt.Columns.Add("Country", Type.GetType("string"))
  dc.MaxLength = 30
  dc = dt.Columns.Add("Continent", Type.GetType("string"))
  dc.MaxLength = 30
  dc = dt.Columns.Add("Population", Type.GetType("string"))

  ' Populate the DataTable...
dt.Rows.Add(New Object() {"Madrid", "Spain", "Europe", 1500000})
dt.Rows.Add(New Object() {"Stockholm", "Sweden", "Europe". 985000})

  ' Now open a MapInfo Table which accesses this DataTable
  Dim ti As TableInfoAdoNet = New TableInfoAdoNet("Cities")
  ti.ReadOnly = False
  ti.DataTable = dt
  Dim table As Table = connection.Catalog.OpenTable(ti)
End Sub
```

Accessing Data Using an IDbCommand

The second form of ADO.NET table is based on the connected object types in ADO.NET: Connection, Command, and DataReader. MapInfo Tables constructed in this fashion are read-only. These types of tables are created by passing to the Catalog an IDbCommand object that is already configured to return all of the data that is to comprise the table. When the table is initially created (by calling Catalog.OpenTable), ExecuteReader is called on the IDbCommand. The resulting data reader is used to determine the columns and their data types. All subsequent cursor requests (other than cursors which retrieve a specific record - called a key fetch) also call ExecuteReader to fetch the data to satisfy the cursor. Notice that this may be very inefficient. If at all possible, use one of the other table types to access your data.

Since the Command-based form of the ADO.NET table is designed to use the generic interfaces without requiring any specific knowledge of any particular implementation of these interfaces, the table also does not assume that the IDbCommand.CommandText is any form of standard SQL. In fact, it may not be SQL at all. This table type does not access, parse, or modify the CommandText. This means that this table type has no mechanism for knowing which column(s) in the results formulate a unique, non-null key

value. For this type of table, it is required to tell the table which column(s) constitute the key. This is accomplished by specifying the KeyType as Explicit and setting the KeyColumns property.

There are many operations inside the MapInfo Data Provider which require the retrieval of a specific record by key (also referred to as a key fetch). Select statements with a where clause of the form MI_Key = '5' is a simple example in which we need to retrieve the record whose MI_Key column can be represented by the string literal '5'. Key retrievals are very common in mapping selections, labeling, and scrolling in a MIScrollableReader (in which case the reader may be scrolling through a list of key values). MapInfo tables are dependent upon the ability to efficiently fetch records by key value. Just as the Command-based form of the ADO.NET table does not read, parse, or modify the CommandText of the IDbCommand object that defines the table (the "Sequential" IDbCommand), it has no ability to modify the IDbCommand object to fetch a specific record. Thus, a second IDbCommand object must be supplied for this purpose. The "FetchByKey" IDbCommand object must meet the following requirements:

• When ExecuteReader is called on this command object, it must produce a data reader that has the same columns as the sequential command object and in the same order.

• The FetchByKeyCommand must contain a Parameters collection and must contain one parameter for each member of the key. For example, if the TableInfo.KeyColumns specifies a key as consisting of the "city" and "state" columns, then the FetchByKeyCommand must contain two parameter objects. The first parameter object is assigned a value representing the first column specified in the TableInfo.KeyColumns collection (e.g., a value for "city"), the second parameter object is assigned a value representing the second column specified in the TableInfo.KeyColumns collection (e.g., a value for "state"), and so on. When ExecuteReader is called on the FetchByKeyCommand, the reader must return the record which represent s the specified key.

This example illustrates how to create a MapInfo Table that accesses data through the ADO.NET connected command objects.

VB example:

```
Public Shared Sub MapInfo_Data_TableInfoAdoNet2(ByVal connection _
    As MIConnection)
  Dim ti As TableInfoAdoNet = New TableInfoAdoNet("EuropeanCities")
  Dim _conn As OleDbConnection = New _
    OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data _
    Source=C:\Data\EuropeCities.mdb")
  Dim selectQuery As String = "SELECT City, Country, Continent, _
    Pop_1994 FROM EuropeCities"
  Dim _OleDbCommand As OleDbCommand = New OleDbCommand(selectQuery)
  _OleDbCommand.Connection = _conn
```

```
    selectQuery = selectQuery + " where City = @City AND _
        Country = @Country"
    Dim _OleDbKeyCommand As OleDbCommand = New _
        OleDbCommand(selectQuery)
    _OleDbKeyCommand.Parameters.Add("@City", OleDbType.Char)
    _OleDbKeyCommand.Parameters.Add("@Country", OleDbType.Char)
    _OleDbKeyCommand.Connection = _conn

' The MapInfo Table will Open/Close the connection as necessary.
' If this is expensive the application could open the connection
' before opening the table and closing the connection after the
' table is closed.
    ti.SequentialCommand = _OleDbCommand
    ti.FetchByKeyCommand = _OleDbKeyCommand

' Tell the table which column(s) constitute a key - for this table
' it is acompound key consisting of values from the City and County
' columns.
    Dim sc As StringCollection = New StringCollection
    sc.Add("City")
    sc.Add("Country")
    ti.KeyColumns = sc
    ti.KeyType = KeyType.Explicit

' Ask the Catalog to open the table.
    Dim tbl As Table = connection.Catalog.OpenTable(ti)

' Now the MICommand object may be used to select data from the table
' (by the name EuropeanCities since that is the alias we assigned to
' it). The data in thistable may be joined with any other table and
' it may be used as source data in a call to AddColumns to populate
' temporary columns with data from this table.
End Sub
```

Data Binding

Data binding is the process of bringing data from a data source into MapInfo Data Access Library (MDAL). Data binding of external data (ADO.NET and other legacy sources) to MapInfo.Data.Table is accomplished by opening an ADO.NET DataTable as a Table using TableInfoAdoNet. The table can then be joined with another table, joined to itself or use Table.AddColumns to bind columns to a second table.

To join a table to itself, following this example:

```
Select ... From T as A, T as B Where A.X = B.Y
```

If an application has data stored in a DataTable or data that is accessible through an ADO.NET data provider, that data can be presented to the Catalog and treated as a MapInfo table. This would be primarily useful if the data were not accessible through one of the other table types.

For example, if the data is stored in a dBase file, Microsoft Access table, or is accessible through ODBC or Oracle's OCI interface, it is recommended that those TableInfo types be used to access the data. Data which cannot be accessed through one of these types of tables, but that can be loaded into a DataTable or is accessible through some ADO.NET Data Provider that implements the Command, Parameter and DataReader object types can still be accessed by the Catalog.

An application may need to make data available as a MapInfo native table so that queries can be executed to join the data with other MapInfo table data. It may also need to be made available to the Catalog so it can be used as the source data in a call to the Table.AddColumns.

View Tables

A view is a way to relate information from one or more tables based on a named select statement. The Catalog allows you to create views based on any table definition. View tables have the following characteristics:

- The data is not copied.
- Access to views always accesses its base tables.
- View is an MapInfo SQL Select Statement with a name (Alias).
- Queries may be joins (forms composite keys).
- Membership in the View is live.
- Exception: Views that aggregate cache the data. Data changed events trigger recomputation.

For more information and code examples, see the MapInfo.Data.TableInfoView class in the Developer Reference Help system.

Result Sets

ResultSets are similar to view tables in that both are defined using a MapInfo SQL select statement and have an associated name (Alias). ResultSets, however, have a fixed membership of records based on the evaluation of the where clause (if any) at the time the result set is created. Any access to the data in a ResultSet always reflects the data in the source table. However changes to the source data will not cause the ResultSet to add/remove a record based on the original where clause. ResultSets manage a set of keys internally.

In general ResultSets are lightweight and temporary. Some of the characteristics of result sets are:

- The data is not copied.

- Access to result sets always accesses its base tables.

- A ResultSet is a sorted list of keys, a collection of column definitions, and a name.

- Membership in the ResultSet is fixed.

- Exception: ResultSets that aggregate, cache the data. Data changed events trigger recomputation.

- ResultSets are vulnerable to Delete and Pack operations.

For more information, see the MapInfo.Data.TableInfoResultSet class in the Developer Reference Help system.

Source Rows

Source rows represent a match between the table records involved in Table.AddColumns. When adding temporary columns to a table, multiple records from the data source may be aggregated together to compute a value for each record in the destination table (also referred to as the bind table). The MapInfo.Data.SourceRows class is a collection of SourceRows that identify the records from the data source that were aggregated together,

SourceRows only exist if the BindType property is DynamicCopy, which indicates that changes to the source data are propagated to the temporary column automatically.

ⓘ   Table.AddColumns is not supported for the following table types: Server, View, Seamless, AdoNet, ResultSet, or Drilldown.

See also Adding Expression Columns to a Table.

The GeoDictionary

The GeoDictionary maintains information about which map entities can be matched to which information. The GeoDictionaries class is a collection of GeoDictionary objects. The MapInfo.Data.GeoDictionary namespace provides support for data autobinding by being a programmatic representation of the GeoDictionary file. The GeoDictionary file contains information about tables (TAB files only). The GeoDictionary is used to automatically determine the table to which application data should be bound. The GeoDictionary is persisted in a file (typically GeoDict.DCT) and is maintained using the GeoDictionaryManager utility application.

AutoMatching Using the GeoDictionary

The MatchResolver.AutoMatch method in the Data.GeoDictionary namespace initiates the AutoMatching process. It does not call AddColumns, i.e., does not do the binding. A subsequent call to BindColumn is required to perform the autobinding, or a direct call to AutoMatchAndBind.

Automatching can encounter ambiguous situations. These situations include:

- multiple source columns are detected in the user data

- multiple tables/layers are detected that match the source column

- multiple geosets/workspaces are available for the matched table/layer.

It is the MatchResolver object with which the GeoDictionary communicates during the match process to solve the ambiguity. It provides the matching algorithm. The basic class selects the first or the one with the highest matching percentage. This class is not sealed and client applications may derive their own class from this and override its behavior.

Making Tables Mappable

Tables can either be mappable (contain a GeometryColumn) or non-mappable (no spatial attribute data). Mappable tables are added to a MapInfo Data Access Library (MDAL) application as a layer in a map. Non-mappable tables, such as customer data, can be made mappable when a GeometryColumn is created for it. MapInfo Data Access Library (MDAL) provides spatial schemas to accomplish this.

Spatial schemas are services that can be applied to a table to enhance its spatial capabilities. There are two type of spatial schemas: XY and PointRef. Non-mappable tables that have attribute columns that represent X and Y values (such as longitude and latitude) use SpatialSchemaXY and tables that have an attribute column which can be used to reference a record in a mappable table uses SpatialSchemaPointRef.

SpatialSchemaXY

SpatialSchemaXY uses the X and Y values of each record in the table to construct point objects and store them in a temporary column known as MI_Geometry. This spatial schema may be applied to tables of any data source except Seamless, View, and ResultSet.

By having a GeometryColumn, the table can now be displayed as a layer in a Map and used for spatial analysis.

SpatialSchemaXY has the following characteristics:

- The Geometry column is editable.

- Editing the Geometry automatically changes the X and Y values.

- You can define styles for each point in the table.

- You can store the spatial information as a TAB file and open like any other table.

This spatial schema can be used for traditional server XY data without a MapCatalog. (Using a MapCatalog may offer better performance on RDBMS's, since more work is done on the server. See The MapInfo_MapCatalog.)

MI_Geometry is a temporary column unless you write out the TAB file explicitly using the TableInfo.WriteToTab method. The schema is automatically regenerated when the table is opened.

VB example:

```
Public Shared Sub MapInfo_Data_SpatialSchemaXY()
Dim ti As TableInfo = _
   TableInfo.CreateFromFile("c:\data\customers.TAB")
      ' a non-mappable table
   Dim xy As SpatialSchemaXY = New SpatialSchemaXY
   xy.XColumn = "Xcoord"
   xy.YColumn = "Ycoord"
   xy.NullPoint = "0.0, 0.0"
      ' Any customer at 0,0 means we don't know their location.
   xy.StyleType = StyleType.None
   xy.CoordSys = _
      Session.Current.CoordSysFactory.CreateLongLat(DatumID.WGS84)
   ti.SpatialSchema = xy
      ' Now set the spatial schema information before
      ' opening the table.
   Dim table As Table = Session.Current.Catalog.OpenTable(ti)
End Sub

Public Shared Sub MapInfo_Data_TableInfoNative2(ByVal ti As _
TableInfoNative)
    ti.WriteTabFile()
End Sub
```

SpatialSchemaPointRef

This spatial schema uses a value in the table's data to create a Point geometry object by matching the value against an equivalent value in a mappable table.

For example, if your table of customers contains addresses with postal codes, the customer records can be tied to the spatial points in a postal code reference table.

SpatialSchemaPointRef is actually a join between two tables, one containing data and the other containing a join column and an object column. The join column contains the same values as the data column in the non-mappable table, such as postal codes. The result of

applying SpatialSchemaPointRef is a table that contains a spatial geometry column for records that were previously non-spatial. This geometry column has the following characteristics:

- The data table may match more than one record in the geometry table. When this happens the similar rows are aggregated into a MultiPoint geometry.

- The geometry is the centroid of the geometry from the other table.

SpatialSchemaPointRef has these characteristics:

- The temporary Geometry column is read-only.

- Any edits to a value in the reference table changes the Geometry value in the data table.

- SpatialSchemaPointRef can be applied to any data source except Seamless, View, and ResultSet.

- You can define styles for each point in the table.

- You can store table information as a TAB file and open like any other table.

For more information and code examples, see the MapInfo.Data.SpatialSchemaPointRef class in the Developer Reference Help system.

VB example:

```
Public Shared Sub MapInfo_Data_SpatialSchemaPointRef(ByVal _
   map As _Map)

 ' a non-mappable table
   Dim ti As TableInfo = _
      TableInfo.CreateFromFile("c:\data\customers.TAB")
   Dim pr As SpatialSchemaPointRef = New SpatialSchemaPointRef
      pr.CoordSys = map.GetDisplayCoordSys()
      pr.StyleType = StyleType.None
      pr.RefTable = "us_zips"

' the column in RefTable which will match the MatchColumn in my data
      pr.RefColumn = "zipcode"

 ' a column in the Customer table
      pr.MatchColumn = "zip"
      pr.RefTableLocation = "c:\data\us_zips.tab"

' Now set the spatial schema information before opening the table.
   ti.SpatialSchema = pr
   Dim table As Table = Session.Current.Catalog.OpenTable(ti)
End Sub
```

MapInfo ADO.NET Data Provider

MapInfo Data Access Library (MDAL) provides mechanisms for issuing SQL commands which return record sets from tables using ADO.NET. The MapInfo ADO.NET Data Provider is one mechanism for accessing data in .NET applications in this fashion. For an alternative that uses the Feature class and SearchInfo methods on the Catalog, see Features and Feature Collections.
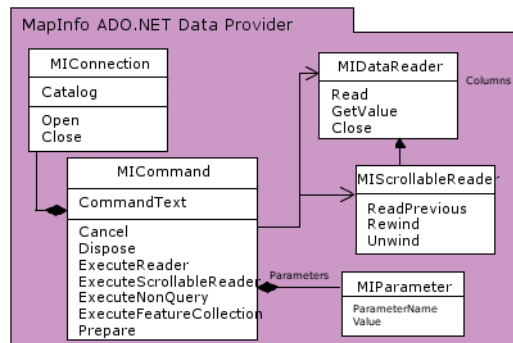
The following sections present the key interfaces and classes for accessing data via the MapInfo ADO.NET Data Provider.

- MIConnection
- MICommand
- MIDataReader
- MapInfo SQL

MIConnection

An MIConnection represents a connection to the Catalog. The connection provides a starting point for issuing SQL commands and obtaining results. Whereas most data provider connections allow the user to immediately begin issuing queries or other commands against existing tables (or schema objects), the MapInfo ADO.NET Data Provider initially has no tables available. Tables need to be opened or created before they can be accessed. When opened, a name (alias) can be associated with the table which is used when resolving identifiers in the query engine.

Connections are not pooled in the MapInfo Data Provider and there is no connection string required to create a new connection.

The MapInfo.Engine.Session class creates and initializes the Catalog which may be accessed through the Session.Current.Catalog property. The MIConnection.Open method obtains a reference to the Catalog using the Session.Current.Catalog property and the MIConnection.Close method sets the internal reference to the Catalog to null.

VB example:

```
Public Shared Sub MapInfo_Data_MIConnection()
    Dim connection As MIConnection = New MIConnection
    Dim command As MICommand = connection.CreateCommand()
    command.CommandText = "Select * From States Where Pop > 1000000"

    connection.Open()
    Dim reader As MIDataReader = command.ExecuteReader()
    Dim i As Integer, n As Integer = reader.FieldCount
    For i = 0 To n - 1 Step i + 1
        Console.Out.Write("{0}\t", reader.GetName(i))
    Next
    Console.Out.WriteLine()
    While reader.Read()
        For i = 0 To n - 1 Step i + 1
            Dim o As Object = reader.GetValue(i)
            If o Is DBNull.Value Then
                Console.Write("null\t")
            Else
                Console.Write("{0}\t", o.ToString())
            End If
        Next
        Console.Out.WriteLine()
    End While
    reader.Close()
    command.Dispose()
    connection.Close()
End Sub
```

MICommand

MICommand provides the necessary interface for executing SQL commands against the MapInfo Data Provider. MICommand creates MIDataReader and MIScrollableReader instances for obtaining data via the ExecuteReader and ExecuteScrollableReader methods, respectively.

Supported Commands

The commands that are understood by the MICommand are:

Select
```
SELECT < select_list >
   FROM { < table_source > } [ ,...n ]
```

```
[ WHERE < search_condition > ]
[ GROUP BY expression [ ,...n ] ]
[ ORDER BY {expression | column_position [ ASC | DESC ] } [ ,...n ]]

< select_list > ::=
  {
      *
      | { table_name | table_alias }.*
      | { expression } [ [ AS ] column_alias ]
  } [ ,...n ]

< table_source > ::=
    table_name [ [ AS ] table_alias ]
```

Insert
```
INSERT [INTO] { table_name } [ ( column_list ) ]
  { VALUES ({expression | NULL}[, ...n]) | query_specification
```

Update
```
UPDATE { table_name }
  SET {{ column_name } = { expression | NULL }} [, ...n]
  [WHERE < search_condition > ]
```

Delete
```
DELETE [FROM] { table_name } [ WHERE < search_condition > ]

< search_condition > ::=
   { [ NOT ] < predicate > | ( < search_condition > ) }
     [ { AND | OR } [ NOT ] { < predicate > |
     ( < search_condition > ) } [ ,...n ] ]

< predicate > ::=
   {
     expression [ { = | < > | != | > | >= | < | <= } expression ]
     | string_expression [ NOT ] LIKE string_expression [ ESCAPE
           'escape_character' ]
     | expression [ NOT ] BETWEEN expression AND expression
     | expression IS [ NOT ] NULL
   }
```

expression
Is a column name, pseudo column, column alias, constant, function, or any combination of column names, column aliases, constants, and functions connected by an operator(s). Column names and pseudo columns may be prefixed with a table name or a table alias followed by the dot (".") character.

group_by_expression
Is a reference to a column in the select list - either an exact copy of the select list expression, the alias, a 1-based number indicating the position of the column, or col$n$ where n is a number representing a column.

order_by_expression

Is a reference to a column in the select list - either an exact copy of the select list expression, the alias, a 1-based number indicating the position of the column, or col*n* where *n* is a number representing a column.

For more information on expressions, where they are used and how to create them, see Features and Feature Collections.

ExecuteFeatureCollection

The ExecuteFeatureCollection method in the MICommand class is the bridge between the MapInfo ADO.NET Data Provider and the Feature object model. This method executes command text (SQL statements) against the data source connection, and builds an IResultSetFeatureCollection. The Feature model is discussed in Features and Feature Collections.

MIDataReader

The MIDataReader provides forward-only, read-only access to the data returned from executing a SQL Select statement. To create a MIDataReader, you must call the ExecuteReader method of the MICommand object, rather than directly using a constructor.

The MapInfo Data Provider allows multiple MIDataReader instances to be in use on the same connection. However, if the Table being accessed resides on a Microsoft SQL Server database, only one MIDataReader may be open at a time.

IsClosed and RecordsAffected are the only properties that you can call after the MIDataReader is closed.Although the RecordsAffected property may be accessed while the MIDataReader exists, always call Close before returning the value of RecordsAffected to ensure an accurate return value.

You must explicitly call the Close method when you are through using the MIDataReader.

When accessing the DataReader through the IEnumerator or IFeatureEnumerator interface, Close() is automatically called when MoveNext() returns false. Only one enumerator can be used on a DataReader.

ⓘ For optimal performance, MIDataReader tries to avoid creating unnecessary objects or making unnecessary copies of data. As a result, multiple calls to methods such as GetValue may return a reference to the same object. Use caution if you are modifying the underlying value of the objects returned by methods such as GetValue.

The MIDataReader provides a means of reading a forward-only stream of rows from the MapInfo data provider. This cursor type is the best performing for accessing a selection of rows since there is little setup or overhead.

Scrollable Data Readers

MIScrollableReader derives from MIDataReader and offers forward and reverse reading. Some of the options available with MIScrollableReader include:

- ReadPrevious
- Rewind
- Unwind
- ReadTop
- ReadBottom
- AtTop / AtBottom

ⓘ   An MIScrollableReader is more expensive to create than MIDataReader. This is the most expensive cursor since there is setup and extra resources necessary to keep track of record order to allow scrolling. Use this cursor only if you need to scroll through the record set.

MapInfo SQL

The MapInfo SQL Language allows you to add powerful analytical processing to your MapInfo Data Access Library (MDAL) application. MapInfo Data Access Library (MDAL) exposes SQL processing to users via the MapInfo ADO.NET Data Provider for accessing data (specifically the MICommand object). Expressions are also used for labeling, thematics, legends, AddColumns, Feature searching, and Selection processing.

MapInfo SQL is standardized based on SQL-3 Specification. For example, you will find that:

- String constants are enclosed in single quotation marks
- Identifiers may be enclosed in double quotation marks
- Select has no relationship to the Selection

A complete reference including code examples for the MapInfo SQL language is provided in the MapInfo SQL Reference, which you can view directly from Visual Studio's Help system.

Features and Feature Collections

The Feature class object model in MapInfo Data Access Library (MDAL) offers a non-SQL-based approach to access and manipulate data. This section covers the Feature class and IFeatureCollection interface. A key task in working with features is the ability to search for them using a query definition object.
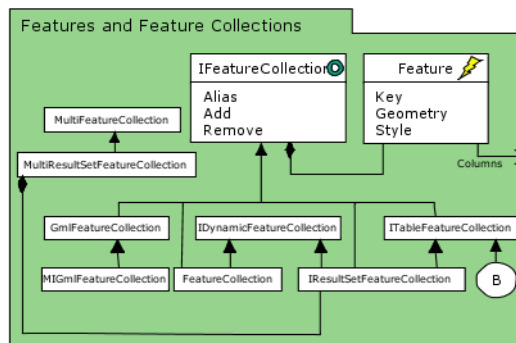
Feature

Features are described by their geometry, style, data source, key and attributes. Typically a feature is a row in a table. A feature's geometry is a FeatureGeometry object. FeatureGeometries can cover a given area (MultiPolygon), a location (Points, MultiPoints); and distance (MultiCurves, LegacyArcs). Additional Geometry classes that derive from FeatureGeometry and are used for map features are FeatureGeometryCollection and LegacyText. (Rectangle, rounded rectangle and ellipse objects also derive from FeatureGeometry, but are used primarily for cosmetic display purposes.)

One of the main uses of computerized maps is to gather information about the features. In MapInfo Data Access Library (MDAL) features are returned in FeatureCollections by any of several methods, either created from scratch using a schema, selected using selection tools or methods or by searching the Catalog for those that meet a specific set of criteria.

You can force a Load using the Load method. Changes made to the Feature are not reflected in the underlying table (if there is one) until the Feature is saved back to the table. This is done using the Update method, or UpdateFeature or InsertFeature. You can throw away any edits done to the Feature object before it is saved using the DiscardEdits method.

A Feature has a schema that describes the attributes of the Feature. The Columns property describes the schema.

Retrieving Features from a Table

A Table is a type of Feature collection. As such, the Features within the table may be enumerated directly. For example:

VB example:

```
Dim ftr As Feature
For Each ftr In table
...
```

The default feature enumerator for a table uses an MIDataReader internally with the following command:

```
command.CommandText = "Select MI_Key, * From \"" + table.Alias + "\"";
```

To retrieve a subset of the features in a table, use one of the Catalog.Search methods or use one of the MICommand.ExecuteFeatureCollection methods.

Modifying Features in a Table

To modify features in a table, use one of the following methods.

- Feature.Update
- Table.UpdateFeature
- Table.InsertFeature

Feature Collections

Feature collections are a group of Feature objects. All Features in a collection share the same Schema (columns). The Feature collection has a schema which is the schema of all of its member feature instances. Some Feature collections own their Features while other Feature collections maintain references to Features.

Searching for Features

One of the most common tasks in Precisely's mapping applications is to search for features that meet certain criteria. Once you have the features you are interested in, you can carry out further analysis, such as thematic mapping. In MapInfo Data Access Library (MDAL), searching for features can be done in a number of ways: using tools, using Catalog search methods, or using SQL and the MapInfo ADO.NET Data Provider.

The following code sample shows two ways to search for the same thing, in this case, cities in New York.

```
// Using SQL
command.CommandText = "Select Obj From States Where state = 'NY';
FeatureGeometry nyGeom = command.ExecuteScalar() as FeatureGeometry;
command.CommandText =
            "SELECT * FROM Cities WHERE Obj within @newyork";
command.Parameters.Add("@newyork", nyGeom);
MIDataReader reader = command.ExecuteReader();
// or… to get a FeatureCollection
IFeatureCollection fc = command.ExecuteFeatureCollection();

// Using Features
Feature fNY = catalog.SearchForFeature("States", _
     SearchInfoFactory.SearchWhere("state='NY'"));
SearchInfo si = SearchInfoFactory.SearchWithinFeature(fNY, _
     ContainsFilter.ContainsType.Centroid);
IDynamicFeatureCollection dfc = _
        catalog.Search("Cities", si) as IDynamicFeatureCollection;
Console.Out.WriteLine( _
        "There are {0} cities whose centroid is within NewYork." _
        dfc.Count);
```

SQL searches are more fully discussed in MapInfo ADO.NET Data Provider. The following sections focus on searches using the Catalog and SearchInfo.

Catalog Search Methods

The Catalog has a number of search methods as members. The overloaded Search method can be used to search on one or more tables. They include different arguments to make each search unique. For example, the basic Search (Table, SearchInfo) searches the given table and returns a FeatureCollection. The Search (ITableEnumerator, SearchInfo) method searches on multiple tables and returns a MultiResultSetFeatureCollection.

The SearchForFeature method returns the first Feature from the results. The SearchReader method returns an MIDataReader cursor with the results.

Code Sample: SearchForFeature

The following example shows how to use Catalog.SearchForFeature and Catalog.SearchWithinGeometry. It finds all the cities in the uscty_1k table that are within Florida. It assumes that tables "usa" and "uscty_1k" are open and that there is one map.

VB example:

```
Public Shared  Sub MapInfo_Data_SearchInfo(ByVal catalog As Catalog)
```

```
    Dim fFlorida As Feature = _
       catalog.SearchForFeature("usa",MapInfo.Data._
       SearchInfoFactory.SearchWhere_("State='FL'"))
    Dim si As SearchInfo =
       MapInfo.Data.SearchInfoFactory.SearchWithinGeomeTry(fFlorida._
       Geometry,ContainsType.Centroid)
    Dim fc As IResultSetFeatureCollection =  _
       MapInfo.Engine.Session.Current.Catalog.Search("uscty_1k",si)

    ' Add results to selection.
MapInfo.Engine.Session.Current.Selections.DefaultSelection.Add(fc)
End Sub
```

SearchInfo and SearchInfoFactory

The MapInfo.Data.SearchInfo class defines the query used in a search and handles any necessary post processing of the search results.

The SearchInfoFactory creates SearchInfo objects. SearchInfoFactory contains a number of search methods that allow you to search using spatial references to your search location or by using geometries that are drawn on the screen.

The following table describes the SearchInfoFactory search methods.

SearchInfoFactory Methods	Behavior
SearchAll	Returns all the rows.
SearchNearest	Returns the rows with table geometries that are closest to the given search point.
SearchWhere	Returns the rows specified by the given where Clause.
SearchWithinDistance	Returns the rows where the table geometry is contained within a distance of the search point, rectangle or geometry. This method uses the Geometry.Distance method to determine if an object is in or out the search area. Previously SearchWithinDistance had buffered the distance and searched within the buffer, leading to less accurate results.
SearchWithinFeature	Returns the rows where the table geometry is contained within the search features's geometry.

SearchInfoFactory Methods	Behavior
SearchWithinGeometry	Returns the rows where the table geometry is contained within the search geometry.
SearchWithinRect	Returns the rows where the table geometry intersects the given rectangle.
SearchIntersectsFeature	Returns the rows where the table geometry intersects with the search features's geometry.
SearchIntersectsGeometry	Returns the rows where the table geometry intersects with the search geometry.

Analyzing Data

Once your data is available in the Catalog, you will want to analyze it to meet your business objectives. The Catalog has an SQL processor that allows you parse and aggregate your data. Here you have two options:

- OGC object-based query interface
- ADO.NET SQL-based interface

The diagram below shows the relationship between the two.

Group 1 shows the OGC query interface. Use these objects to construct a query. The interface allows you to create queries to filter columns and rows, as well as add spatial and non-spatial conditions. The queries interact through the Search methods off those query objects to return data readers and result sets. Use these objects if you are more comfortable with object-oriented programming and less so with SQL syntax. See SearchInfo and SearchInfoFactory.

The ADO.NET interfaces, shown in group 2, use the defined ADO.NET model to allow access via the MapInfo SQL language. The ADO.NET interfaces use SQL syntax to interact with the Catalog. In this instance you need to generate the SQL statement and assign it to the MICommand object. These objects use the Execute command to return a data reader or result set. See MapInfo ADO.NET Data Provider.
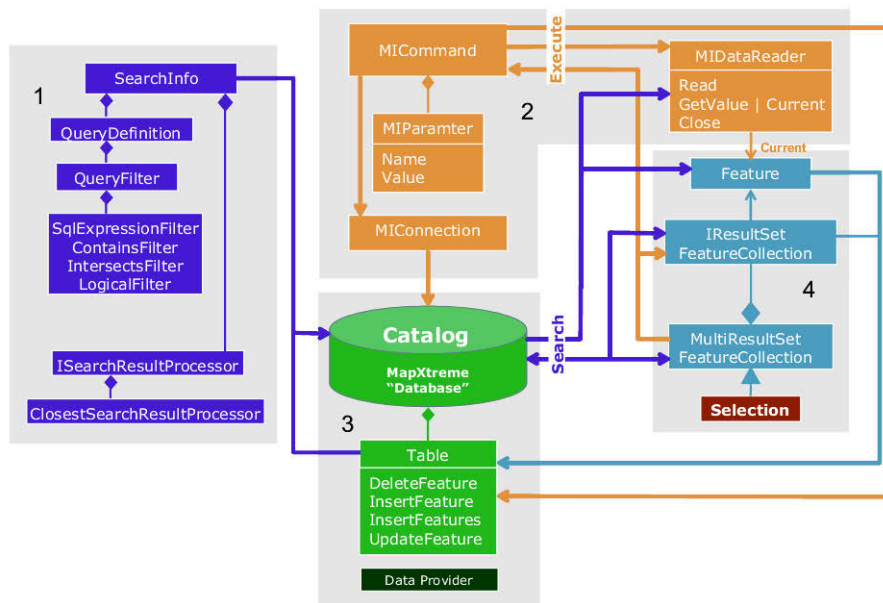
MapInfo Data Access Library (MDAL) Data Model

Both the OGC query-based and ADO.NET command-based approaches use the Catalog (group 3) to organize the data sources as a response to the object or SQL query. The object-based query API will generate SQL and pass this to the Catalog for processing. In some instances you may be able to generate more efficient SQL by hand, but the objects are well defined and the interfaces restrict how you interact so the SQL tends to be optimal. If you are comfortable with the SQL language using the ADO.NET method may be more comfortable. But if you are inexperienced with SQL then the OGC object based query will work just as well.

The MapInfo SQL syntax is defined in the SQL Reference which ships with MDAL. The language is based on SQL3 and has special MapInfo operators defined for spatial analysis. These operators begin with the MI_ prefix.

Data Readers, MemTables and Result Sets

The methods to access data return a data reader or result set. A data reader allows access in a sequential manner and does not store copies of data. It retrieves the data from the data source, except in the case where the data source is cached. Result sets are collections of keys. These keys allow you access back to the original tables and do not create copies of the data.

A MemTable also allows you to store data from various sources into one table. This table type stores data in a combination of memory arrays and temporary disk storage. When data is added, the MemTable makes a copy of the data and does not have a key or pointer back to the original table. These are useful for temporary layers for maps and containers for return values of processes such as a geocoding or routing result. MemTable access and map rendering performance is equivalent to native tables.

Result sets are a great tool when you need access to a defined set of rows and when you need to get data from the source. If the source data may change during your session then this method allows you to see the results if the data source supports concurrent access. Since MemTables are copies of data they are a static set of data rows and will not reflect changes from the original data sources.

Improving Data Access Performance

Performance is always an important aspect to any application that accesses data. Consider the following list in your design and development plans for your application.

- Only request the data you need (especially from an RDBMS). This limits the amount of data sent over the connection.

- Only sort tables if you need an ordered list. This process takes time to read through the entire table to build an order. Also it will be slower if there is no index on the column.

- Only scroll if you need random access to a table. This also builds indexes to speed up access and remember order. Data readers access the data directly with no need to read extra data.

- Use consistent coordinate systems for Join and Search operations. This eliminates the need to convert geometries for every access.

- Use indexed columns for Join / Filter / Sort / Aggregate operations.

- Use CentroidWithin, ContainCentroid, and EnvelopesIntersect prior to actually checking for geometry intersects. These tests are very quick and in most cases eliminate a lot of geometries from your list with little effort.

- Use BeginAccess/EndAccess (especially for file-based tables) when performing multiple queries and/or edits.

- Try to avoid calls such as Area and Buffer in the Where clause because the operation will have to be done each time a new cursor is created.

- Try to avoid calls such as Area and Buffer in the Select list when defining a view or result set for similar reasons.

- Use result sets for intermediate results or operations where you manage keys. These are very light weight and afford quick direct access back to the original data.

3

3 – Working with Core MDAL Classes

The MapInfo.Engine namespace contains the interfaces and classes that relate directly to the core MDAL functionality. This includes the core ISession interface which is the starting point for all MDAL applications. Classes in this namespace include Session and Selections, and SearchPath. Other types in the namespace are supporting classes, delegates, structures, and enumerations for Collections, Resources, and CustomProperties.

In this chapter:

Session Interface

The ISession interface is the starting point for all applications integrating MDAL. It manages the initialization of resources needed for a MDAL application.

An instance of ISession holds components of the object model such as the DataAccess engine, MapFactory, CoordSysFactory so that the application can do work. The following diagram illustrates the classes that implement ISession interface.
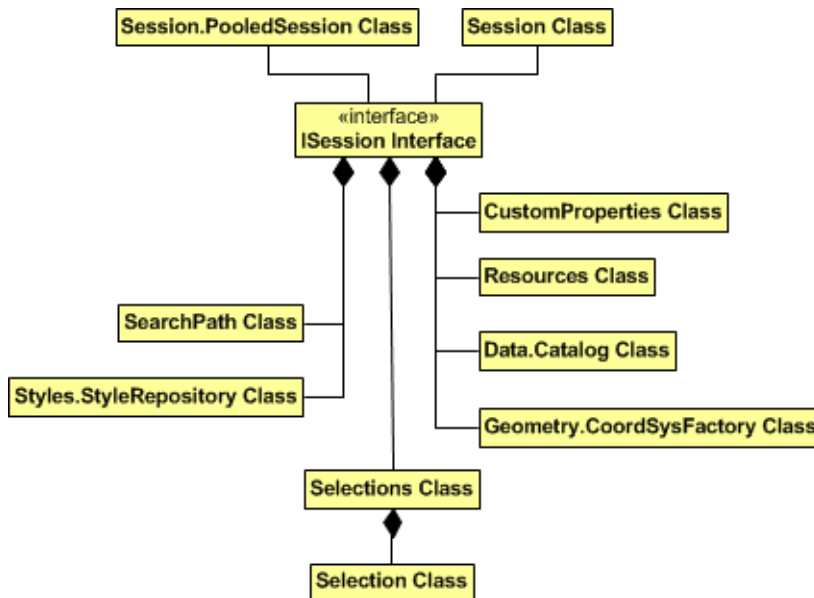
For an ASP.NET application each client request has its own ISession instance. This instance resides in the calling context and is available throughout the lifetime of the client's request.

For a single-threaded desktop application there is only one instance. On a multi-threaded desktop application there is one instance per thread.

The MapInfo.Engine.Session class provides access to the ISession object. To get the current ISession instance, access the MapInfo.Engine.Session.Current property.

Using Session.Dispose Method

The MapInfo.Engine.Session class has two overloaded Dispose methods. Your choice will depend on the type of application you are building.

Session.Dispose()

Session.Dispose() disposes the ISession instance that is accessible through the Session.Current property. This method is used only for multi-threaded desktop applications.

Do not use this for single-threaded desktop applications. For single-threaded desktop application, Dispose is called automatically when the application is shutdown or when the AppDomain using MDAL is unloaded.

Selection Class

A Selection is a collection of IResultSetFeatureCollection objects that holds lists of features. These features are a subset of rows in a table. They could be property boundaries, street networks, cell tower locations, or natural features such as rivers. They are typically drawn with special highlighting when they display in a Map. There can only be one IResultSetFeatureCollection for a given table in a Selection.

There can be more than one Selection in a ISession. The Selections collection contains all of the selections in the application. There is always at least one selection, known as the DefaultSelection.

Each Selection must have a name and unique alias. By default, map selection tools modify the Selection when used. Each tool can be set to use any particular Selection.

A selection in MDAL is not a copy; it is a reference to an IResultSetFeatureCollection for a given table in a Selection. If you attempt to modify a Selection after you have closed the table that you are working with, the reference to the IResultSetFeatureCollection will be invalid, causing an exception.

Features can also be selected through search methods from the MapInfo.Data.Catalog class which returns IResultSetFeatureCollection collections. A Selection object can be passed into a search, which can be used to populate or change a Selection.

Features can also be selected via the ExecuteFeatureCollection method from the Data.MICommand class. In this case, you would execute SQL commands against the MapInfo Data Provider.

For more information on features, tables, the Catalog, and the MICommand see Working with Data.

SelectionChangedEvent

A delegate method is attached to the SelectionChangedEvent in order to receive notification that this selection has changed. For example, if a record is added, the SelectionChangedEvent is fired.

Selection Code Examples

The following are code examples of common selection operations. Additional code examples are included in many topics of the MDAL *Developer Reference*.

Selecting Features Within Another Feature

A common search technique using MDAL is to find features within another feature. You may do this to find all the customers within a postal code boundary or all the highways that are under construction in a sector. Follow the example below. The parameter f is a MapInfo.Data.Feature.

VB example:

```
Dim si As MapInfo.Data.SearchInfo = _
MapInfo.Data.SearchInfoFactory.SearchWithinFeature(f, _
   MapInfo.Data.ContainsType.Centroid)
Dim irfc As MapInfo.Data.IResultSetFeatureCollection = _
   MapInfo.Engine.Session.Current.Catalog.Search("USCty_8k", si)

MapInfo.Engine.Session.Current.Selections.DefaultSelection.Clear()
MapInfo.Engine.Session.Current.Selections.DefaultSelection.Add(irfc)

irfc.Close()
```

Checking a Table for Selections

Follow the code example below to learn how to get a count of selections in a table.

VB example:

```
Public Shared Sub MapInfo_Engine_Selection2()
   Dim session As ISession = MapInfo.Engine.Session.Current
   Dim tableUsa As Table = session.Catalog("usa")

   ' Get fc for selection on usa.
   Dim fc As IResultSetFeatureCollection = _
     session.Selections.DefaultSelection(tableUsa)
   Dim nCount As Integer = 0
   If Not fc Is Nothing Then
```

```
    nCount = fc.Count
   End If
End Sub
```

You can also perform selection operations using MapInfo SQL queries and with the ADO.NET data provider. See Chapter 2 Working with Data.

Returning All Columns From a Table

The following sample shows how to return all columns from a selection:

VB example:

```
Dim Connection As MIConnection = New MIConnection
        Connection.Open()
        Dim ti As MapInfo.Data.Table = _
          MapInfo.Engine.Session.Current.Catalog.GetTable("usa")
        Dim si As MapInfo.Data.SearchInfo = _
          MapInfo.Data.SearchInfoFactory.SearchAll()
        si.QueryDefinition.SetColumns("*")
        Dim irfc As MapInfo.Data.IResultSetFeatureCollection = _
          MapInfo.Engine.Session.Current.Catalog.Search(ti.Alias, si)
        Dim l As MapInfo.Data.Feature
        For Each l In irfc
            Dim column As MapInfo.Data.Column
            For Each column In l.Columns
                MessageBox.Show(column.ToString())
            Next
        Next
```

Event Arguments

The MapInfo.Engine namespace contains various event argument classes that provide data for events. Refer to the online help for more information. Some of the event argument classes include:

- CollectionCancelableEventArgs – Provides data for a collection event that can be cancelled.

- CollectionEventArgs – Provides data for a collection event.

- NodeSelectionChangedEventArgs – Fires these event arguments when the node selection changes.

- SelectionChangedEventArgs – Other objects can attach delegates to this event to get notified when a selection changes.

Exceptions

The Engine namespace contains various exception classes. Refer to the online help for more information. Some of the exception classes include:

- ResourceNotFoundException – Throws this type of exception when the requested object is not found in the Resource table.

- ResourceTypeMismatchException – The exception that is thrown when the object read from Resources was not of the expected type.

- TimeoutException – The exception is thrown on Current timeout while waiting for a pooled ISession to become available.

4 – Creating Expressions

Expressions are used throughout MDAL to describe the exact pieces of information you need to display and analyze in your mapping application. This chapter covers creating expressions for a wide range of product areas, including data access, creating themes, labeling maps and more.

In this chapter:

Expressions Overview

Expressions are statements that are used to describe and format data. For example, in English, an expression might read like "a median income of more than $50,000, or "female percent of population."

Expressions are formed using column names, constants (i.e., specific data values), along with functions and operators that act upon the columns and constants. The operators and functions are defined in the MapInfo SQL Language, developed to support MapInfo .NET supported products. For details, see the MapInfo SQL Reference via the Help Viewer in Visual Studio.

Use expressions to make the most of your data. By using expressions you can:

- Show only the columns and rows of data that interest you.
- Derive new columns by calculating new values based on the contents of your existing columns.
- Aggregate data to work with subtotals instead of the entire table.
- Combine data from two or more tables into one results table.

Many of the data sets you will use include more objects and information than necessary for your projects. In many cases it is easier to work with a subset of the complete data product. For example, if you were tracking crime statistics for a certain county by census tract, you would not need the census tracts for the entire state. You would use an expression to extract just the census tracts for the county.

Expressions are used throughout MDAL, in the following areas:

- SQL statements (select, insert, update, delete, group by, order by)
- SQL functions that take expressions as an arguments (e.g., the geometry argument in MI_Area() is an expression that returns a geometry object.)
- Adding columns (MapInfo.Data.Table.AddColumn creates a temporary column based on an expression.)
- Feature searches (SearchInfo and SearchInfoFactory)

Creating Expressions

The simplest possible expression consists of a constant, such as "2" (numeric example) or "Friday" (text example).

Other simple expressions consist of a column name, for example:

```
POP_2000
STATE
```

When you request specific multiple columns in a select statement, for example, these columns together are known as an expression list.

```
Select colA, colB, colC from Table1, Table2
Select colA/2, ColB/ColC from Table1
```

You can also write expressions that perform mathematical operations on your data.

For example, RENT + UTILITIES is an expression that adds two columns together. This expression could be used in a SQL statement to find all apartments that have a total cost of less than $800 per month.

Where Clause – Boolean Expressions

A Boolean expression is a search condition that results in a value of either True or False. For example, the expression

```
2 < 5
```

is a Boolean expression because the result is True.

All expressions that contain relational operators, such as the less than sign (<), are Boolean. The operators AND, OR, and NOT, are Boolean operators. Boolean expressions are also called comparison expressions, conditional expressions, and relational expressions.

```
POP_2000 > 500000
POP_2000 <= POP_1990
PROVINCE <> 'Ontario'
County = 'Columbia' AND VALUE >= 250000
```

Supported operators in MapInfo SQL are defined in the MapInfo SQL Reference.

Boolean expressions are used in the "where clause" of an SQL statement. The where clause is the expression that controls the rows that are returned (the rows that result in True).

For example, the boolean expression in this statement follows WHERE. Only objects in the Europe table that fall within the boundary of France will be returned as True.

```
"SELECT * FROM Europe WHERE MI_Geometry within @France";
```

Functions In Expressions

Functions in MapInfo SQL are used to create even more complex expressions to retrieve data that meets specific criteria. For example, MapInfo SQL supports many of the usual database functions that work with strings, dates/time, and numbers,

The most powerful functions in MapInfo SQL are those that take advantage of the spatial nature of mapping data. These geographic functions are used to create new geometries, measure area and length, return spatial information, validate spatial relationships among geometries, and others. Supported functions are defined in the MapInfo SQL reference.

An example of using a function in an expression might be when you wish to look at the area of a table of boundaries, such as school districts. Use the function MI_Area() to return the area of each record in the table.

Additional examples of functions in expressions are found in the Expressions Examples section below.

DateTime and Time Expressions

When using DateTime and Time Expressions with MDAL, please be aware of the following:

- If a DateTime column or Time column is used alone in an expression, it is formatted using the current locale.
- If a DateTime or Time column is in an expression, its string value is TimeToNumber or DateToNumber + space + TimetoNumber.
- Operator math on Time or DateTime is not supported. You can add a number to a Date, but not to a Time or DateTime.

Expression Examples

The following highlights some uses of expressions.

SQL Statement Examples

This example will select all records from the Eurcity_1K table that are within Germany and have a population of over 1 million.

```
Select * from Eurcity_1K WHERE (MI_Geometry MI_Within @Germany) AND Tot_Pop >
1000000
```

The following examples make selections based on Time and Date columns in the table. This example will select all crime records from a "CrimeActivity" table where the crime occurred between 12:00:00 AM and 6:00:00 AM:

```
SELECT * FROM CrimeActivity WHERE CrimeTime BETWEEN '12:00:00 AM' AND '6:00:00
AM'
```

Where CrimeTime is a Time column that stores the time at which the crime occurred.

This example will select employee Names from an "Employee" table who were born before December 31, 1970.

```
SELECT Names FROM Employee WHERE BirthDay < '12/31/1970 12:00:00 AM'
```

Where BirthDay is a Date column that stores the birthdays of employees.

MapInfo SQL Function Example

The following expression uses a MapInfo SQL function to find features within a buffer.

```
Obj CentroidWithin MI_Buffer(Obj, 5, 'km', 'Spherical', 24)
```

This expression uses a MapInfo SQL special keyword reserved for geographic objects called 'Obj'. This keyword describes the geometry of the object such as its coordinate system and bounds. This keyword is compatible with previous versions of MapX and MapInfo Professional. It is equivalent to the column name MI_Geometry.

Note that km and Spherical are enclosed in single quotes. In MapInfo SQL, string literals must be enclosed in single quotation marks while identifiers such as column names, table names, aliases, etc.) should be enclosed in double quotation marks, but only needed if the parsing logic is unable to correctly parse the identifier. This would include identifiers that have spaces in their names or other special characters.

To find features that fall outside the buffer, the expression would look like:

```
NOT Obj CentroidWithin MI_Buffer(Obj, 5, 'km', 'Spherical', 24)
```

Add Columns Example

When adding temporary (computed) columns to a table using the AddColumns method, the columns supplied contain an expression that defines how the value for the column is computed. The expression may contain an aggregation function if multiple source records are expected to match up to a single record in the table to which the columns are being added.

The example below uses expressions to represent population density "Pop_1990 / MI_Area(Obj, 'sq mi', 'Spherical')". The expressions are preceded by their new column names. PopDensity1990 and PopDensity2000.

VB example:

```
Public Shared Sub MapInfo_Data_TableAddColumns(ByVal miTable _
      As Table)
  Dim NewCols As Columns = New Columns
```

```
   NewCols.Add(New Column("PopDensity1990", "Pop_1990 / _
      MI_Area(Obj, 'sq mi', 'Spherical')"))
   NewCols.Add(New Column("PopDensity2000", "Pop_2000 / _
      MI_Area(Obj, 'sq mi', 'Spherical')"))
   miTable.AddColumns(NewCols)
End Sub
```

For more information on adding columns, see Adding Expression Columns to a Table.

Feature Search Example

The following example uses a boolean expression SearchWhere("State='FL'") that, when executed, will return a value of 1 for each row that contains FL.

VB example:

```
Public Shared Sub MapInfo_Data_SearchInfo(ByVal catalog As Catalog)
   Dim fFlorida As Feature = catalog.SearchForFeature("usa", _
      MapInfo.Data.SearchInfoFactory.SearchWhere("State='FL'"))
   Dim si As SearchInfo = _
      MapInfo.Data.SearchInfoFactory.SearchWithinGeometry(fFlorida._
      Geometry, ContainsType.Centroid)
   Dim fc As IResultSetFeatureCollection = _
      MapInfo.Engine.Session.Current.Catalog.Search("uscty_1k", si)

' Set the map view to show search results

   MapInfo.Engine.Session.Current.MapFactory(0).SetView(fc.Envelope)
' Set the view of the first map.

 ' Add results to selection.
MapInfo.Engine.Session.Current.Selections.DefaultSelection.Add(fc)
End Sub
```

For more information on the Feature class and search methods, see Features and Feature Collections.

5 – Accessing Data from a DBMS

5

MapInfo Data Access Library (MDAL) provides access to a number of spatially aware DBMS. This is a powerful feature that allows developers to connect to live data stored in spatial servers, such as Microsoft SQL Server or the Oracle Spatial databases. Spatial servers allow companies to host their map data in their enterprise database for central management and security.

In this chapter:

Accessing Remote Spatial Data

You can access data using MDAL with different DBMS servers. The servers include:

- Microsoft Access 2007 and Excel 2007
- Microsoft Access 2003
- Oracle 11G (11.1.0.6.0 and 11.1.0.7.0)
- Oracle 10G, 10GR2
- Microsoft SQL Server 2012 (with SQL Native Client 11)
- Microsoft SQL Server 2008 (with SQL Native Client 10)
- Microsoft SQL Server 2014

You can add a table from data in a DBMS using the TableInfoServer class in the MapInfo.Data namespace.

The details for adding spatial data are included in the following sections.

Accessing Remote Tables Through a .TAB File

Using the MDAL, an application can access DBMS data "live", or can open a MapInfo Professional linked table. However, the linked table will be read-only and cannot be refreshed by your application. The data is actually from the remote database and does not reflect the data in the local linked version.

You can create a .TAB file to provide access to remote data. To generate a .TAB file using MapInfo Professional, choose File > Open a DBMS table.

The .TAB file is a text file; you can create a .tab file using any text editor. Once you have created the .tab file, you can access it the way you access any other MapInfo .TAB file programmatically through the Catalog object or through the Workspace Manager.

Accessing Remote Tables Without a .TAB File

An application does not need a .TAB file to access remote data. The following code sample illustrates this process.

VB example:

```
Public Shared Sub MapInfo_Data_TableInfoServer(ByVal connection As _
     MIConnection)
   ' Note: Do not specify any columns. These are determined
```

```
' dynamically from the query

Dim ti As TableInfoServer = New TableInfoServer("Provinces")
ti.ConnectString = "SRVR=ontario;UID=mapx;PWD=mapx"
ti.Query = "Select * From Provinces"
ti.Toolkit = ServerToolkit.Oci

ti.CacheSettings.CacheType = CacheOption.Off ' On is the default
Dim tbl As Table = connection.Catalog.OpenTable(ti)
```

End Sub

Mapping DBMS Data with X/Y Columns

You can access data from a DBMS table that has X/Y coordinates. You need to create a MapInfo_MapCatalog and register the tables as SpatialType 4.0 and specify two column names as the coordinates. The columns should be indexed on the table. Connect to the DBMS via ODBC and specify the new columns as "Obj" or "MI_Geometry" in your query.

Accessing Data from Oracle

To connect to an Oracle database from an MDAL application, the Oracle OCI connectivity client must be installed and appropriate permissions granted. See your Oracle documentation for detailed information.

Geometry Conversion

The table below shows the translation from MDAL objects to Oracle Spatial (SDO_GEOMETRY).

From MapInfo	To Oracle
NULL geometry	NULL
Point	1 POINT
MultiCurve (with IsLegacyLine = true)	2 LINESTRING Geometry contains one line string
Polygon	3 POLYGON Geometry contains one polygon.

From MapInfo	To Oracle
FeatureGeometryCollection	4 Collection Geometry is a heterogeneous collection of elements.
MultiPoint	5 MULTIPOINT
MultiCurve	6 MULTILINESTRING Geometry has multiple line strings.
MultiPolygon	7 MULTIPOLYGON Geometry has multiple polygons.
Ellipse	NULL
LegacyArc	NULL
Rectangle	NULL
LegacyText	NULL
RoundedRectangle	NULL
PieTheme, BarTheme	NULL

The table below describes the translation from Oracle (GTYPES) to MapInfo Spatial objects.

From Oracle GTYPES	To MapInfo
0 *UNKNOWN_GEOMETRY (Spatial ignores this geometry.)	
1 POINT Geometry contains one point.	Point
2 LINESTRING Geometry contains one line string.	MultiCurve
3 POLYGON Geometry contains one polygon.	MultiPolygon
4 *Collection Geometry is a heterogeneous collection of elements.	FeatureGeometryCollection
5 MULTIPOINT Geometry has multiple points.	MultiPoint

From Oracle GTYPES	To MapInfo
6 MULTILINESTRING Geometry has multiple line strings.	MultiCurve
7 MULTIPOLYGON Geometry has multiple polygons (more than one exterior boundary).	MultiPolygon

*Some data loss may occur when translating to or from MapInfo object format. They are interpreted (when possible) as single point SDO_POINTTYPE values if not already NULL. They "grab" the first point in the ordered array which would be interpreted as a NULL geometry.

Oracle Support for Z and M Values

MDAL supports reading and writing Oracle GTYPEs with Z and M values. The presence and order of Z and M is determined by inspecting the DIM_INFO array in the USER_SDO_GEOM_METADATA for the table. MDAL checks for the following dimension names (case insensitive):

- For Z dimension: "Z", "Elevation", " Depth" and "Z Dimension"
- For M dimension: "M", "Measure", and "M Dimension"

Tables that contain M and/or Z values now return FeatureGeometry objects that contain the data for the dimensions present. FeatureGeometry instances inserted or updated into an Oracle table will preserve each of the four dimensions of the new geometry (XYZM) that the Oracle table is defined to support. For geometries containing dimensions unsupported by the Oracle table, the values for those dimensions are ignored during insert/update operations. For geometries not containing dimensions supported by the Oracle table, NULL values are supplied for the missing dimensions during insert/update operations. For example, when inserting a geometry with no Z or M values into a table that is defined with dimensions x, y, and m, the M values stored in the table will be NULL.

The success of any of these insert/update operations may also be dependent upon additional server-side validation including explicit column constraints and validation of values against the dimensional extents specified in the SDO_GEOM_METADATA system table.

SDO_GEOMETRY Arc and Circle Translation

Circles and circular arcs can be resolved to MultiCurves with a resolution of 25 segments per 360 degree circle.

Visualization of Non-translatable Oracle Objects

An Oracle Spatial Object that your MDAL application is unable to translate produces a Point object with a default style (a black star) at the location of the SDO_Spatial point, or the first SDO_Spatial ordinate in the ordinate array. This is to enable a visual representation of the non-translatable object in the proper geographic area to which it belongs. Examples of non-translatable objects are user-defined objects GTypes 0,4,5, or invalid SDO_geometries containing unrecognized GTypes, ETypes, or interpretations. The second class should also fail using SDO_VALIDATE_GEOMETRY().

Centroid Support

An MDAL application uses the SDO_POINT as the centroid value for polygons. This centroid feature is used to position labels, and also affects the tool selection of the object. The Oracle SDO_GEOMETRY.SDO_POINT_TYPE field (if not NULL) is interpreted as the feature centroid if the point exists inside the region. If the point exists outside of the region, its centroid is calculated as always.

ⓘ　There is currently no method or tool in MDAL to set the centroid of a region feature, but one may read and use a stored centroid.

Oracle Spatial Reference Support (SRID)

An Oracle SDO_GEOMETRY column may be defined with a spatial referencing system. This is done by providing the Oracle SRID in the USER_SDO_GEOM_METADATA and also by assigning that SRID in the stored SDO_GEOMETRY values. If a table contains an Oracle Spatial column with an assigned SRID, your MDAL application is able to query and properly interpret the data. The MapInfo_MapCatalog must contain the same MapInfo Professional CoordSys string as indicated in the SRID of the data, since it is the Coordsys in the MapInfo_MapCatalog that is currently used to interpret and update the data.

If the Spatial column does not contain an SRID value, (the value is NULL), your MDAL application is also able to interpret the data via the MapInfo Professional Coordsys defined in the MapCatalog.

When loading tables that use the Latitude/Longitude coordinate system (Geodetic Data) to Oracle Spatial, it is important to verify that all geometry coordinates are between (-180,180) longitude and (-90, 90) latitude. Geodetic data coordinates beyond that range

are not supported in Oracle Spatial and may cause problems. You can check your data using MapInfo Professional before loading, or use the Oracle Spatial SDO_GEOM.VALIDATE_LAYER() function on the table after loading it to Oracle Spatial.

Accessing Data from MS SQL Server

MDAL supports data stored in Microsoft's SQL Server 2008, SQL Server 2012 and SQL Server 2014. The following information pertains to SQL Server 2008.

SQL Server 2008 Support

MDAL supports reading and writing data from and to Microsoft SQL Server 2008, including the spatial data types GEOMETRY and GEOGRAPHY, along with M and Z value support for both spatial formats.

To access data from SQL Server 2008, MDAL requires SQL Server Native Client 10. Data is then handled like data from other remote database management systems that MDAL supports[1]. Use the MapInfo.Data.TableInfoServer class to define the connection string and an SQL statement to execute on the remote table. Internally, MDAL uses ODBC to access the remote database.

The following table shows how objects are handled in MDAL given a specific object type from SQL Server 2008.

SQL Server 2008 Spatial GEOGRAPHY or GEOMETRY	MDAL FeatureGeometry
Sql Server 2008 Spatial GEOGRAPHY/GEOMETRY	FeatureGeometry
Point	Point
LineString	MultiCurve
Polygon	Multipolygon
MultiPoint	MultiPoint
MultiLineString	MultiCurve
MultiPolygon	MultiPolygon

1.

SQL Server 2008 Spatial GEOGRAPHY or GEOMETRY	MDAL FeatureGeometry
GeometryCollection	FeatureGeometryCollection
GeometryCollection containing only Points and/or MultiPoints	MultiPoint
Geometrycollection containing only LineStrings and/or MultiLineString	MultiCurve
Geometrycollection containing only Polygons and/or MultiPolygons	MultiPolygon
An EMPTY GEOMETRY/GEOGRAPHY, e.g., Point empty	NULL

This table shows how an MDAL FeatureGeometry is written back to SQL Server 2008

MDAL FeatureGeometry	SQL Server 2008 Spatial GEOGRAPHY or GEOMETRY
Point	Point
MultiPoint	MultiPoint
MultiPoint containing only one Point	Point
MultiCurve	MultiLineString
MultiCurve containing only one Curve comprised of two points	LineString
Multipolygon	MultiPolygon
FeatureGeometryCollection	GeometryCollection [*]
Rectangle	NULL
RoundedRectangle	NULL
Ellipse	NULL

MDAL FeatureGeometry	SQL Server 2008 Spatial GEOGRAPHY or GEOMETRY
LegacyArc	NULL
LegacyText	NULL

*   This GeometryCollection may contain any or all of the following types: MultiPoint, MultiLineString, and MultiPolygon.

SQL Server 2008 provides new types for date and time information. The following table shows how date and time types are mapped to MDAL date and time types.

SQL Server	MDAL
Date	Date
Time	Time
DateTime	DateTime
SmallDateTime	DateTime
DateTime2	DateTime
DateTimeOffset	No support

Spatial tables from SQL Server 2008 must be registered in the MapInfo_MapCatalog so that MDAL understands what it reads.

The MapCatalog provides four new spatialcolumn values to represent SQL Server 2008 tables:

   17.x for GEOMETRY without M and Z values

   18.x for GEOGRAPHY without M and Z values

   20.x for GEOMETRY with M and Z values

   21.x for GEOGRAPHY with M and Z values.

Data can be uploaded using MapInfo Professional or EasyLoader or you can use MapInfo Professional to make existing data mappable, which will create the entry in the MapCatalog. See The MapInfo_MapCatalog for more information on the MapCatalog.

DBMS Connection String Format

ODBC Connection String Format

The format of the ODBC connection string is defined by several clauses separated by semicolons (;). Each clause has the form Key=Value. Important keys are listed below.

Keyword	Description
DSN=	Specifies the ODBC data source name.
	Caution: If you use the DSN= syntax key, the name that you specify must match the data source name in use on the user's system. Note that different users might use different names to refer to the same data source. If you cannot know in advance what data source name to use, use the DRIVER= syntax key instead of the DSN= syntax key.
DRIVER=	Specifies the exact driver name of the installed driver. Used instead of the DSN= syntax key.
	Example:
	`DRIVER={SQL Server}`
	ⓘ This does not apply to Oracle Spatial.
UID=	Specifies the desired UserId for the data source, if required.
PWD=	Specifies the user's password for the data source, if required. Passwords do not need to be in the connection string for the two strings to match. If two tables are in the same database, the connection string is the same.

Oracle Spatial Connection String Format

These are the Oracle Spatial keywords. The string is defined by several clauses separated by semicolons (;). Each clause has the form Key=Value. Important keys are listed in the table below.

Keyword	Description
SRVR=	Reflects the service name for the server set in the Oracle Net8 EasyConfig utility. This is required for Oracle connectivity, but does not apply to ODBC connections.
UID=	Specifies the desired UserId for the data source, if required.
PWD=	Specifies the user's password for the data source, if required.

Sample Connection Strings

Here are sample connection strings for Oracle Spatial and Microsoft SQL Server 2008 ODBC drivers.

Oracle Spatial connection string:
```
UID=george;PWD=password;SRVR=OracleSpatial9i
```

Microsoft SQL Server 2008 connection string:
```
DRIVER={<driver>};
SERVER=<server>;UID=<uid>;PWD=<pwd>;Database=<database>
```

where <driver> for SQL Server Spatial should be the most current available, SQL Server Native Client 10.0 or higher version.

Defining Mappable Tables in Server Table Queries

The query you specify for a server table defines the result set of data from your DBMS that represents the data in the table being added. You can formulate a fairly complex query to do powerful server-side analysis that defines a mappable table in MDAL. Your MDAL application uses this query internally to access the data.

MDAL generates several internal queries based on your query to access the data in a map as well as selection/key based queries. The table from which the geometry column is selected must be registered in the MapInfo MapCatalog on the server. MDAL requires this to obtain certain metadata about the geometry column such as the coordinate system, spatial type, and default styles.

In order for a query to define a mappable table, the query must contain both a geometry column and a key column. Sometimes for more complex queries on small sets of data (where the spatial indexing or spatial predicate cause the query to fail), you can specify TableInfoServer.MbrSearch=false to enable the results to be mapped.

The Geometry Column

If you do not specify a geometry column that your MDAL application can recognize, the table is opened, but cannot be added to a map (the table is unmappable). MDAL determines the geometry column of the table by looking it up in the MapCatalog and by examining the result set datatype of the column. You can reference the geometry column generically via the pseudo column name "Obj", or you may refer to the geometry column using its specific column name. This form is required to reference the geometry column for an X/Y mappable layer. You can specify a geometry column via any server-supported geometry function/expression.

Example
```
Select Obj from rdbsdata
Select sw_geometry from rdbsdata
select sw_member, ST_Buffer(geometry, 66.0, 0.1) from rdbsdata
     // a geometry function
Select st_geometry(st_point(72.5, 42.5) from rdbsdata
     // a geometry constructor
```

Oracle sdo_buffer example:
```
Select mi_prinx, mdsys.sdo_geom.sdo_buffer(geoloc, (select diminfo from
sdo_geom_metadata where table_name = 'ALINE'), 20) from aline where prinx = 1
```

Oracle constructor example:
```
Select 1 "mi_prinx",
mdsys.sdo_geometry(3,null,null,mdsys.sdo_elem_info_array(1,3,3),
mdsys.sdo_ordinate_array(-79.919909,40.553465,-71.060457,45.363657)) from dual
```

SQL Server 2008 Spatial function example:
```
select location_id, geography::Point(lat, long, 4326 /*WGS84*/) as geog from
dbo.store_locations
```

The Key Column(s)

A key column(s) must be returned in the query to enable it to be opened as a table. This is what enables your MDAL application to identify each row in the result set.

The key column does not need to be specified in the query in most cases.

Your MDAL application can look up and determine the best key column(s) to use in order to uniquely reference a row in the result set, and then add them to the query if they are not present. In most cases, this is the primary key/unique index.

For Oracle Spatial tables, the MI_PRINX may be used.

For some queries, it is not possible for your MDAL application to identify the key. This is the case in a query on a view or a synonym. The view or synonym must appear in the MapInfo MapCatalog. They also must be registered as required with the underlying Spatial index system in most cases. Since MDAL cannot determine the key on these, a mechanism is provided to allow the application developer/query writer to identify the key column in the result set. The key must be a single column and must be a distinct value in the result set. To identify the column that is to be used as the key column, you can specify column alias of prinx or mi_prinx,   (e.g., select custid mi_prinx, custname, Obj from mycust).

Example
```
Select customer_id mi_prinx, obj from customer_view
```

The column alias "mi_prinx" is used to identify and use the customer_id column as the key column for the table. You can alternately alias the desired key column in the create view statement to identify the key column automatically for any query on that view.

Example
```
Create view customer_view as select customer_id mi_prinx, geoloc from customer
```

In general, if a column name or column alias of prinx, or mi_prinx is found in the result set, that column is used as the key column for the table. This enables the application/query writer to specify the key column they desire.

Accessing Attribute Data

To use all available data columns, specify a query such as Select * From tablename. You are not required to specify * (asterisk); instead, you can designate specifically which columns you want to use. For the best performance, limit your query so that it retrieves only the needed columns.

When you add a DBMS table, for performance sake, you should only specify the columns in the query that you intend to use in your application. These are the spatial column, the key column(s), which are added automatically if you do not specify them, and columns you want to label with, or create a theme from. You may use the pseudo columns "OBJ"

for any mappable table to refer to the column(s) containing the spatial data. This is required for a table using the MapMarker MDIGEOADDRESS column on a table with an X/Y column.

You can use any server side expression/function to specify a column. Also, avoid select * from tab in a real application.

The following code example defines a server table using a TableInfoServer.

VB example:

```
Public Shared Sub MapInfo_Data_TableInfoServer(ByVal connection As _
MIConnection)
   ' Note: Do not specify any columns. These are determined
   ' dynamically from the query

   Dim ti As TableInfoServer = New TableInfoServer("Provinces")
   ti.ConnectString = "SRVR=ontario;UID=mapx;PWD=mapx"
   ti.Query = "Select * From Provinces"
   ti.Toolkit = ServerToolkit.Oci

   ti.CacheSettings.CacheType = CacheOption.Off ' On is the default
   Dim tbl As Table = connection.Catalog.OpenTable(ti)
End Sub
```

Performance Issues

Establishing a connection with the database server may take several seconds. This is a one-time cost, incurred when the table is first opened.

The retrieval speed depends on how much data is retrieved from the server. In some cases, working with data from a server is noticeably slower than displaying a map from a local file. Speed also depends on whether your MDAL application has already cached the map features that are being fetched.

Working with the Cache

Knowing how to work with cache in MDAL enables you to improve your application's performance. The sections below describe what the cache is, how it works in the MDAL object model, and the CacheSettings property of the TableInfoServer object.

What Is the Cache?

In place of local files, applications can access MDAL table features from a remote database. In place of reading these records from the database each time the map needs to be acted upon, your application can temporarily store these records in the cache. This limits the number of calls between the application and the remote database. Records in a server table can be cached to improve the performance of your application. Server table data is cached internally as it is read. All subsequent redraws read from this cache instead of going to the server database for the same data. The cache is able to offer significant redraw performance improvement.

There are several settings that developers can use to customize cache usage. The cache can be enabled (or disabled) when the server table is added by specifying the values for the CacheSettings property of the TableInfoServer object and is On by default.

How the Cache Works

For each record that is cached, each attribute data value is stored in memory and each feature object is stored on disk in a temporary Rtree file. For tables with a lot of records and/or a large record size (for example, number of bytes per record for the attribute data), caching may use a significant amount of memory. If an application tries to cache too much data, too much virtual memory usage may be required, which can degrade performance. Applications should be selective about how the cache is utilized. MDAL offers a variety of mechanisms for controlling the cache.

The TableInfoServer Object and the CacheSettings Property

When a table is added to the map, the cache is enabled by default but can be further configured using the CacheSettings property of the TableInfo object. This property has four possible values: ON, OFF, ALL, USER, with ON being the default for TableInfoServer, OFF is default for other TableInfo objects.

Parameter	Description
OFF	A value of 'Off' means that the table will not use the cache at all. All data operations will go directly to the database server.
ON	Caching is enabled and the table automatically performs caching of the retrieved records. The user may additionally control the cache through the cache constraint objects.
	To avoid having a cache that grows excessively large, there are controls that can be placed on the table's cache to determine when to discard old cached data. These controls are properties of the CacheParameters object, which can be set at the time the table is initially opened. This allows the developer to set limits on the maximum amount of memory or disk space used by the cache, and/or the maximum number of records to maintain in the cache. These limits can be used individually or in combination to provide the cache management that best suits the application's needs.
	ON is the default setting for the CacheParameter setting for a TableInfoServer. For other TableInfo data sources, the default is OFF. For example, TAB files are not cached by default.

Parameter	Description
USER	A value of USER for the LayerInfo CACHE parameter means that your application creates a cache, but the only records that are placed in the cache are those specified by the application developer. The mechanisms available for specifying which records are placed in the cache are BoundConstraint, FeaturesConstraint, and AllFeaturesConstraint objects. The word constraint implies that these objects are constraining the cache to include the specified records. The BoundsConstraint object can be used to place all records into the cache for which the MBR of the feature intersects the MBR of the constraint.
	A FeaturesConstraint object can be used to add specific records to the cache. For example, if an analysis is going to be performed that involves multiple steps and/or reads of the Feature or RowValues of the feature, possibly on a set of features returned from a Layer.Search, Layer.SearchWithinDistance, etc., it may be advantageous to place these records into the local cache for the duration of the analysis and remove them when finished. The FeaturesConstraint provides this capability. If an application is going to perform an analytically intensive operation that may hit every record, it may be desirable to temporarily cache the entire set of data for the layer. This is accomplished by using the AllFeaturesConstraint. These cache constraint objects can also be used when the cache is set to ON. In this case, they may add records to the cache but have no effect on the cache's history of previous map window views. The constraint objects can also be used when the cache is set to OFF or ALL in which case they have no effect.
	ⓘ The constraint objects have no effect on non-server tables.
ALL	The entire table is cached. With this option, the table's data is retrieved from the server once and accessed locally from that point forward. To refresh the data in the cache, use the Refresh method on the table.

If you try to cache too much data or too many tables, virtual memory usage may be forced, and performance gain could be lost.

Cache Storage Type:

Cache Storage type indicates the table type used to store the cached records of RDB server tables. Following Storage Types are used in MDAL:

1. **Native**: The cached records are stored on disk in a temporary MapInfo Native table. If multiple tables are opened with this cache storage type, then a different set of MAP, DAT, etc. files will be created for each table. This storage type has a maximum file-size limits of 2GB.

2. **MemTable**: The cached records are stored in memory in a temporary MemTable table. Geometry objects are stored on disk in a temporary MAP file. If multiple tables are opened with this cache storage type, then a separate set of in-memory cache and on-disk MAP files will be created for each table. This storage type also has an upper size limit of 2GB.

3. **NativeX**: The cached records are stored on disk in a temporary MapInfo NativeX table. If multiple tables are opened with this cache storage type, then a different set of MAP, DAT, etc files will be created for each table. This NativeX storage type supports UTF-8 and UTF-16 charsets as well as it can store the data files that are greater than 2GB in size.

4. **MemNativeX**: The cached records are stored in memory in a temporary MemNativeX table. Geometry objects are stored on disk in a temporary MAP file. If multiple tables are opened with this cache storage type, then a separate set of in-memory cache and on-disk MAP files will be created for each table. MemNativeX cache supports UTF-8 and UTF-16 charsets as well as it can store the data files that are greater than 2GB in size.

5. **Geopackage**: The cached records are stored on disk in a temporary Geopackage database(*.GPKG) file. All RDB server tables opened in a session with this cache storage type are cached in a single Geopackage database and for each open RDB table, a separate Geopackage cache table is created in that single Geopackage database.

6. **MemGeopackage**: The cached records are stored in an in-memory Geopackage database.

The MapInfo_MapCatalog

In order to display data on a map, your MDAL application needs to access a special table, known as the MapInfo_MapCatalog. One catalog must be created per database before any tables in that database can be viewed as a map layer in an MDAL application. The

MapCatalog must contain information about the spatial columns in each of the mappable tables you want to access from the database. The MapInfo EasyLoader utility automatically inserts the appropriate row into the MapInfo_MapCatalog when the table is uploaded into the database.

Your application can use a MapInfo_MapCatalog that already exists on the server. (This same catalog is shared by various MapInfo client applications). If there is no MapInfo_MapCatalog on the server, you need to create one. MDAL supports the storage of style information for individual features in remote databases.

Loading Spatial Data to DBMS

If you have spatial data in the form of a MapInfo table, you can import it into your DBMS database.

To load data into Microsoft SQL Server and Oracle Spatial, use the MapInfo EasyLoader, that is distributed with MapInfo Professional and available for download from support.precisely.com. The EasyLoader utility automatically creates a MapInfo_MapCatalog when you upload a table, if there is no MapInfo_MapCatalog already present.

Manually Creating a MapInfo MapCatalog

If you are not a MapInfo Professional or EasyLoader user, you or your database administrator will need to create the MapCatalog manually, as described below. You only have to create the MapCatalog once per server/database.

1. Create the user MAPINFO in the specific database where the mappable tables are located.

2. Create the table MAPINFO_MAPCATALOG in the database.

   The Create Table statement needs to be equivalent to the following SQL Create Table statement:

```
Create Table MAPINFO_MAPCATALOG (
SPATIALTYPE Float,
TABLENAME Char(32),
OWNERNAME Char(32),
SPATIALCOLUMN Char(32),
DB_X_LL Float,
DB_Y_LL Float,
DB_X_UR Float,
DB_Y_UR Float,
VIEW_X_LL Float,
VIEW_Y_LL Float,
VIEW_X_UR Float,
```

```
VIEW_Y_UR Float,
COORDINATESYSTEM Char(254),
SYMBOL Char(254),
XCOLUMNNAME Char(32),
YCOLUMNNAME Char(32),
RENDITIONTYPE INTEGER,
RENDITIONCOLUMN CHAR(32),
RENDITIONTABLE CHAR(32)
NUMBER_ROWS INTEGER
)
```

ⓘ   It is important that the structure of the table looks exactly like this statement. The only substitution that can be made is for databases that support varchar or text data types; these data types can be substituted for the Char data type.

---

3. Create a unique index on the TABLENAME and the OWNERNAME, so only one table for each owner can be made mappable.

```
create unique index mapcat_i1
on mapinfo.mapinfo_mapcatalog (OwnerName,TableName)
```

4. Grant Select, Update, Insert, and Delete privileges on the MAPINFO_MAPCATALOG. This allows users to make tables mappable.

```
grant select, insert, update, delete on mapinfo.mapinfo_mapcatalog to public
```

Adding Rows to the MapInfo_MapCatalog

For each spatial table that you want to access from your application, you need to add a row to the MAPINFO_MAPCATALOG table. If you do not use MapInfo Professional to manage the MapInfo_MapCatalog, you will have to add rows to the MAPINFO_MAPCATALOG table manually.

The following table describes the syntax and meaning of each column:

Column Name	Values to Assign	Examples
SPATIALTYPE	**MapInfo Spatial Object Format**	14.0 = SQL Server
	1: Point layer in X/Y columns indexed	14.1
	with micode (a serialized quadtree key)	14.2
	4: Point layer in X/Y columns	14.3
	5.x: SpatialWare for Oracle	
	6.x: Ingres SQL - Not Supported	
	7.x: Sybase SQS - Not Supported	
	8.x: Oracle SDO version 2 - Not Supported	
	9.x: MapInfo Geocoding DataBlade SpatialWare Point Module	
	10.x: MapInfo Geocoding DataBlade XY Module	
	11.x: SpatialWare IDS/UDO datablade	
	13.x: Oracle Spatial	
	14.x: SpatialWare for Microsoft SQL Server	
	17.x:SQL Server 2008 GEOMETRY without M and Z values	
	18.X: SQL Server 2008 GEOGRAPHY without M and Z values	

SPATIALTYPE (continued)	20.x: SQL Server 2008 GEOMETRY with M and Z values	
	21.x: SQL Server 2008 GEOGRAPHY with M and Z values	
	Spatial Object Type	
	x.0: Points only	
	x.1: Lines only	
	x.2: Regions only	
	x.3: All types supported	
	ⓘ This column describes the Spatial Object Format of how the data is stored and indexed and the Spatial Object type(s) supported and not supported in the column. The digits to the left of the decimal point are the Spatial Object Format. The digits to the right represent the type of Spatial Object Type that can be stored in the column.	
	Maps to	
	MapInfo.GeometryColumn.PredominantGe-ometry Type, and	
	Has<Line/Point/Region/Text>Geometries	
TABLENAME	The name of the table.	STATES

SPATIALCOL-UMN	The name of the column, if any, containing spatial features:	SW_GEOMETRY
	SW_GEOMETRY (mappable using Spatial-Ware Type/IDS/UDO)	
	NO_COLUMN (mappable using X–Y)	
	MI_SQL_MICODE (mappable using MI Code) Or the name of the IDS/UDO, or Oracle column that is ST_SPATIAL datatype. Name of the Oracle SDO_GEOMETRY column.	
DB_X_LL	The X coordinate of the lower left corner of the layer's bounding rectangle, in units that are indicated by the COORDINATESYS-TEM (see below). Maps to Map-Info.Data.GeometryColumn.Bounds	-360
DB_Y_LL	The lower left bounding Y value.	-90
DB_X_UR	The upper right bounding X value.	360
DB_Y_UR	The upper right bounding Y value.	90
VIEW_X_LL	The X coordinate of the lower left corner of the default view. The default view only applies if this is the first table to be opened. Maps to MapInfo.Data.GeometryColumn.Default-View	-180
VIEW_Y_LL	The lower left bounding Y value of the default view.	-45

VIEW_X_UR	The upper right bounding X value of the default view.	180
VIEW_Y_UR	The upper right bounding Y value of the default view.	45
COORDINATE-SYSTEM	A string representing a MapInfo CoordSys clause (but without the keyword CoordSys at the very start), which specifies a map projection, coordinate units, etc. For simple Lon/Lat maps, specify Earth Projection 1, 0. Maps to MapInfo.Data.GeometryColumn.CoordSys	Earth Projection 1, 0
SYMBOL	A MapInfo Symbol clause (if the layer contains only points); or a Symbol clause followed by a Pen clause (indicating styles for linear features) followed by another Pen clause (indicating styles for the borders of regions) followed by a Brush clause. Maps to MapInfo.Data.GeometryColumn.DefaultStyle	Symbol(35,0,12) Pen(1,2,0) Pen(1,2,0) Brush(2,255,255)
XCOLUMN-NAME	For the X/Y mappable tables, specify the name of the column containing X-coordinates. If there is no such column (i.e., if this table uses a single spatial column instead of a pair of X-Y columns) then specify *NO_COLUMN* or leave empty. Maps to MapInfo.Data.SpatialSchemaXY	NO_COLUMN
YCOLUMN-NAME	For the X/Y mappable tables, specify the name of the column containing Y–coordinates, or specify *NO_COLUMN* Maps to MapInfo.Data.SpatialSchemaXY	NO_COLUMN

RENDITION-TYPE	This indicates how the object style information is applied.	0 or 1
	0 – Indicates that all the objects in the table will have the style specified in the symbol field of the MapCatalog applied to them. No per-record styles are in effect. Objects will be read/updated using the default style for the table.	
	1 – Indicates that the table uses per-record styles. The table has a separate column that contains a MapBasic string representation of the style information for each object in the table (the same format that is currently used in the MapCatalog's SYMBOL column). The style column in the table is recorded in RENDITIONCOLUMN.	
RENDITION-COLUMN	If RENDITIONTYPE is 1, this field stores the name of the column in the spatial table that contains style information. This column is automatically added to any query against the table and is maintained (updated) as the object is updated. Users should NOT specify this column in their queries as problems can occur with intersect or update statements. Queries which include this column in the select clause (excluding the wildcard character "*") may access the values through the Dataset object. Rows with a NULL value in their style column will have the style from the SYMBOL field of the MapCatalog applied to the object. Creates a MapInfo.Column.DataType with MIDBType.Style	MI_SYMBOLOGY

	Currently not used, but reserved for future use.	Null
NUM-BER_ROWS	Currently used by MapInfo Professional.	Null

Per-Record Styles

Per-record style support brings a feature to spatial database implementations that has long been available in MapInfo TAB files. Specifically, it allows each geometry in a single table to have its own style. For example, a single 'public institution' table in Oracle Spatial can have schools, town halls, libraries, and police departments and each point type would be represented with its own symbol (i.e., a school symbol for all the schools). Similarly, a single road table in SpatialWare SQL Server may have different road types such that streets are shown as a single pixel black line, secondary roads as a double pixel red line and interstates as parallel red lines.

To use per-record styles, your table must be represented with an entry in the MapCatalog with appropriate settings for RENDITIONTYPE, RENDITIONCOLUMN, and RENDITIONTABLE.

ⓘ If these columns are not present, the table's default style will be applied to all objects.

Symbol, Pen, Brush Clause Syntax

If you are manually creating a MAPINFO_MAPCATALOG table to provide support for a remote spatial database, you will need to specify a symbol style, and possibly line and fill styles as well.

Specifying Point Styles

Use a Symbol clause to specify point styles. There are three types of Symbol clauses: one for specifying MapInfo 3.0-style symbols; one for specifying TrueType font symbols; and one for specifying bitmap symbols.

Symbol Syntax	Example

Symbol(shape, color, size)
or
Symbol(shape,color,size,font,fontstyle,rotation)
or
Symbol(bitmapname,color,size,custom style)
	`Symbol(35,0,12)`
	
	`Symbol(64,255,12,"MapInfo Weather",17,0)`
	`Symbol("sign.bmp", 255, 18, 0)`

Specifying Line Styles

Use a Pen clause to specify line styles. In a MapInfo_MapCatalog, you may need to specify two pen clauses: one to specify the appearance of linear features, and another to specify the appearance of region borders.

Pen Syntax	Example
Pen(thickness, pattern, color)	`Pen(1, 2, 0)`

Specifying Fill Styles

Use a Brush clause to specify the style for closed features (regions)..

Brush Syntax	Example
Brush(pattern,color,backgroundcolor)	`Brush(2, 255, 65535)`

Text Objects Limitation

LegacyText objects have their own way of displaying style that is separate from the use of the MI_Style column. Therefore any form of text object needs to be treated differently than other objects. The style for any text object is embedded and a NULL value is inserted into the style column.

Troubleshooting

If you encounter problems with your SpatialWare or Oracle applications, use the following table to help analyze and solve the problem.

Problem Description	Possible Cause	Solution
The table is not matchable.	Data binding was attempted against a SpatialWare layer.	AddColumns is not currently supported for SpatialWare layers.
No object was found using the index that you specified.	A query was made against a table that does not exist.	Check that the table name is correct and in the proper case. Also, the table may need to be mappable.
	No spatial object is contained in the result of the spatial query.	Use the EasyLoader Upload utility to make the table a mappable table.
	A query was made against a non-spatial table.	Check the query for possible syntax errors. Also make sure that the result of the query includes the field specified in the spatial column in the MapInfo_MapCatalog.
Map appears to have incorrect zoom level. For example, the map may be zoomed out too far to identify any geography.	The MBR for a DBMS layer is determined by the MapInfo_MapCatalog table. The table extents in the MapCatalog result in a different zoom level than the one you desire for your output.	Edit the extents (DB_X_LL, DB_X_UR, DB_Y_LL, DB_Y_UR) in the MapInfo_MapCatalog using the MapInfo Professional MBX tool, MISETMBR.MBX.

6 – Spatial Objects and Coordinate Systems

6

This chapter covers the MapInfo.Geometry namespace and provides descriptions and examples for writing applications for creating and manipulating geometry objects.

In this chapter:

Introduction to MapInfo.Geometry Namespace

The MapInfo.Geometry namespace is used for creating and manipulating geometry objects, and the coordinate systems in which they are used. Geometry objects are used in maps to represent single points, such as cities (represented as point objects), boundary lines, such as county borders (represented by MultiCurve objects), and regions, such as countries or zip code areas (represented by MultiPolygon objects).

The classes, interfaces, and enumerations in the MapInfo.Geometry namespace define the types representing the geometries and coordinate systems used in displaying geographic features on a map. The Geometry model provides support for Z and M values on FeatureGeometry objects. Interfaces allow for creation and editing of the geometry objects. Methods such as Buffer, Combine, Difference, and Intersection provide object processing on single objects or pairs of objects.

Geometries

The Geometry class allows for the creation, editing, and other manipulation of geometry objects. Classes which inherit from the Geometry class and represent types of Geometry objects include Point, MultiPoint, Polygon, MultiPolygon, Curve, CurveSegment, LineString, and Ring. The following legacy classes are also inherited from the Geometry class: Rectangle, RoundedRectangle, Ellipse, LegacyArc, and LegacyText.

The Geometry class represents the topmost level of the MapInfo Geometry object model. This class is abstract, and cannot be instantiated. All classes that derive from this class contain knowledge concerning their coordinate system. All classes are able to make copies of themselves, and compare themselves to other Geometry objects for equality.

The diagram below shows a representation of the Geometry model.
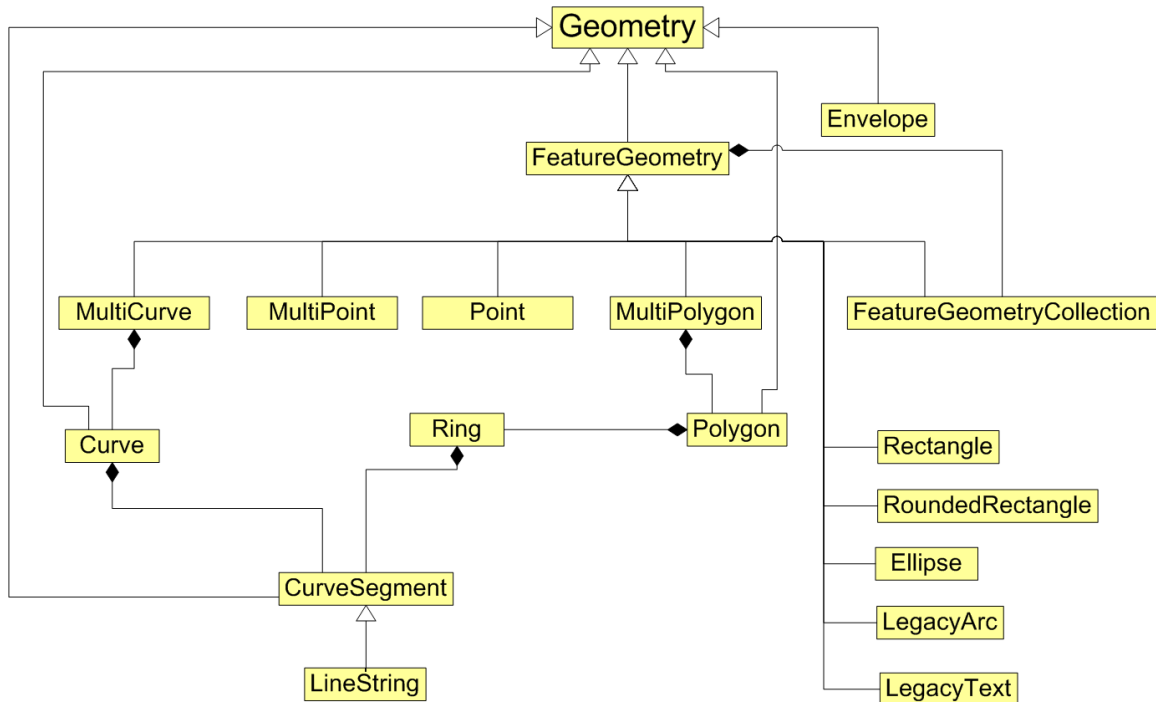
Geometry Objects

All geometry objects in MDAL are created with a specific coordinate system that cannot be changed. If you need to alter the coordinate system of an object you can make a copy of that object in the new coordinate system.

Editing Geometry Objects

All Geometry objects contain a method for retrieving an interface to an editor that places the object into Edit Mode. Once editing is finished the EditingComplete() method needs to be called to signify that the editing of the object is complete. When the EditingComplete() method is called, the order of the objects contained by the Geometry is reshuffled and all references to them are dropped and need to be re-established in order to access them again.

For example, the user creates a MultiPolygon and then edits the MultiPolygon. If the user inadvertently moves a node of the interior ring to be outside of its containing Polygon the Polygon is no longer valid. When EditComplete is called, all the contained objects within the MultiPolygon are reshuffled, fixing the problem.

The geometry objects in the MDAL Object Model are described in the following sections.

FeatureGeometry Objects

The FeatureGeometry class is specifically designed to contain classes that can be placed in tables and that can be parts of Features and FeatureCollections. In order for something to be displayed in a map, it needs to be in a table. FeatureGeometry objects are by definition included in tables. Any object that is a subclass of Geometry and not a subclass of FeatureGeometry cannot be saved to a table or included as part of a Feature or FeatureCollection. An exception is thrown, or the program will not compile, if such an operation is attempted. The FeatureGeometry class, like the Geometry class is abstract and cannot be instantiated.

Support for M and Z Values

Feature geometries support reading and writing M and Z values at each node of the object.

Support for reading and writing M and Z values for linear objects was accomplished by extending the MDAL Geometry model. FeatureGeometry objects (Point, MultiPoint, MultiPolygon, MultiCurve and FeatureGeometryCollection) can hold values for X, Y, Z and M for each node.

IsMeasured and Is3D properties allow you to determine whether the object has M or Z values. Additional properties and methods are provided to read and modify M or Z values at each node. The minimum and maximum ranges of M and Z values can be retrieved as well.

MDAL provides creation and editing capabilities for FeatureGeometries. For more information, see MapInfo.Geometry.FeatureGeometry class in the Developer Reference.

M values for MultiCurves provide valuable information in linear network applications for tracking and managing assets, events and conditions.

Point

Points are derived from the FeatureGeometry class and represent a single point on a map. Points can be contained within a MultiPoint collection and then operated upon collectively.

Use the following example code to model the creation of a Point:

```
using MapInfo.Geometry;

CoordSys longLatNad83;
CoordSysFactory coordSysFactory = new CoordSysFactory();
longLatNad83 = coordSysFactory.CreateLongLat
     (MapInfo.Geometry.DatumID.NAD83);
DPoint point = new DPoint(0.0, 0.0);
```

Developer Guide

```
Point pointGeometry = new Point(LongLatNad83, point);
```

MultiPoint

A MultiPoint contains an unordered, disconnected set of Points and is useful for performing multiple operations on multiple points.

Use the following example code to model the creation of a MultiPoint object:

```
using MapInfo.Geometry;

CoordSys longLatNad83;
CoordSysFactory coordSysFactory = new CoordSysFactory();
longLatNad83=coordSysFactory.CreateLongLat
     (MapInfo.Geometry.DatumID.NAD83);
MultiPoint multiPointGeometry = new MultiPoint
     (longLatNad83, pointArray);
```

where `pointArray` is an array of DPoints.

MultiCurve

The MultiCurve class is derived from the FeatureGeometry class, and contains a possibly disconnected set of Curves. These Curves may interact in many ways; they can be connected or disconnected, and can intersect or overlap each other.

Although the Geometry object model supports multiple CurveSegments for each Curve, the current version of the MapInfo engine is limited to having one CurveSegment per Curve that is part of a FeatureGeometry (i.e., MultiCurve). This limitation derives from the current TAB file format, which remains largely unchanged for this version of MapInfo. Hence, the limitation concerns FeatureGeometry objects only.

Upon construction of a MultiCurve where the constructor takes a Curve or Curves which may contain multiple CurveSegments per Curve, the actual Curves contained in the constructed MultiCurve are altered to always contain only one CurveSegment per Curve. Currently, the only types of CurveSegments that exist are LineStrings. Curves containing multiple LineString CurveSegments have the LineStrings combined to form one large LineString.

Upon completion of editing (signified by calling EditingComplete(), any Curve which was added to the MultiCurve and contained multiple CurveSegments is altered in a similar manner as noted above to produce Curves containing single CurveSegments.This limitation, of Curves contained in MultiCurves always containing only a single CurveSegment, should be removed in future versions of MapInfo as new types of CurveSegments are introduced (e.g., EllipticalArcs, CircularArcs, and Splines), and the TAB file format is altered. Also, during construction and on completion of editing, any empty Curves are automatically removed from the MultiCurve.

Line objects made up of two points that exist in MapInfo TAB files become MultiCurve FeatureGeometry objects. They can be detected as two-point Lines by using the IsLegacyLine property of the MultiCurve:

See the Developer Reference for a code example of creating and editing a MultiCurve object.

Measure Values on MultiCurves

The Geometry object model supports M and Z values on FeatureGeometry objects. M, or measure values, hold data at the nodes of MultiCurve objects that describes anything you wish to map and analyze, including physical assets, conditions or events. The M values play an important role in linear referencing and dynamic segmentation.

Curve Sort Order

The order of the Curves in a MultiCurve may be altered during construction, as compared to the array of Curves passed to the constructor, and upon completion of editing. Due to this, plus the removal of empty Curves, and limitations in the current implementation, any references to Curves contained in a MultiCurve prior to and during editing may no longer be valid after editing is completed (i.e., after EditingComplete() is called). If these objects are referenced, they throw an ObjectDisposedException. After editing, the parts of a FeatureGeometry should be reacquired to obtain a valid reference.

LineStrings

A LineString is a directed collection of sequential points that are connected in a linear manner. Any two consecutive points in the LineString are connected by a straight line. LineStrings can be part of Curves or Rings, or they can exist as a stand-alone Geometry. LineStrings that are part of Curves or Rings inherit the coordinate system of their container. Stand-alone LineStrings can be empty. A LineString that is contained in a Curve or Ring that is not in Edit Mode cannot be empty, and must contain at least two points.

See the Developer Reference for a code example.

Rectangle

A Rectangle Geometry contains two points representing the lower left hand and upper right hand corners of the Rectangle. The other two points are implied. Rectangles are always axis aligned, and always appear to be rectangular in shape, regardless of the coordinate system, and are not projected. They do not contain any warping that may be represented by the coordinate system.

See the Developer Reference for a code example.

RoundedRectangle

A Rounded Rectangle behaves exactly like a Rectangle but is displayed with the corners appearing rounded as a display-time only feature. The corners display as quarter circles and the radius of the circle is controlled by the CornerRadius parameter.

Because RoundedRectangle objects, like rectangle objects, are defined by two points and always display axis-aligned and unprojected, they are designed to be used primarily for cosmetic display purposes. While many operations are available using Rectangle objects (e.g., Combine), internally, a MultiPolygon copy of the Rectangle is used for these operations. The resulting MultiPolygon contains 5 points (with the first and last points being identical), and are effected by the coordinate system. In some instances, the converted Rectangle may no longer appear rectangular. Use the CreateMultiPolygon method to convert a RoundedRectangle to a FeatureGeometry object.

See the Developer Reference for a code example.

Ellipse

The Ellipse is inscribed in an axis-aligned rectangle defined by a DRect. The DRect is defined by two points, the opposite corners of a rectangle, with the other two corners of the rectangle implied. The Ellipse displays as if it were unprojected, regardless of the coordinate system, and any skew that may be represented by the coordinate system.

Because Ellipse objects are defined by two points and always display axis-aligned and unprojected, they are designed to be used primarily for cosmetic display purposes. While many operations are available using Ellipse objects, internally, a MultiPolygon copy of the Ellipse is used for these operations. The resulting MultiPolygon is effected by the coordinate system and in some cases may no longer appear as a perfect ellipse.

See the Developer Reference for a code example.

LegacyArc

The LegacyArc object is a portion of an Ellipse and is defined by a DRect, a start angle, and an end angle. The Ellipse is constructed to be inscribed in the rectangle defined by the DRect. The rectangle, in which the Ellipse is inscribed, is axis-aligned and is always rectangular regardless of the coordinate system used. The angles are measured in degrees with zero being along the positive X-axis and positive angles being in the counterclockwise direction. The angles are only stored to a tenth of a degree resolution with values between 0.0 and 360.0.

Because LegacyArc objects are defined by two points (for the DRect) and angles, and are always displayed axis aligned, they are designed to be used primarily for cosmetic display purposes. While many operations are available using LegacyArc objects, internally, a MultiCurve copy of the LegacyArc is used for these operations. This can sometimes lead to unexpected results.

See the Developer Reference for a code example.

LegacyText

The LegacyText object is the MapInfo Professional equivalent of a text object. If a given database does not support Text the LegacyText object can be lost when using that format. LegacyText objects are placed within a geographically-sized rectangle with a lower-left anchor point specified. The point-size of the text is based upon what fits best within the rectangle.

LegacyText objects do not fit nicely into the Geometry model. Several methods available on the Geometry FeatureGeometry classes, such as Combine, make no sense for LegacyText and will throw a NotSupportedException. Text objects do exist in MapInfo native TAB files in the Geometry column. The LegacyText class provides a way to access these objects. Refer to online reference for specific behaviors of LegacyText objects.

Geometry Objects

Geometry objects that are not also FeatureGeometry objects need to be converted to a suitable FeatureGeometry object to be displayed on a map. Most FeatureGeometry classes contain constructors that take appropriate Geometry objects and create new FeatureGeometry objects:

```
using MapInfo.Geometry;

Curve curve = new Curve(csys, lineString);
MultiCurve multiCurve = new MultiCurve(curve.CoordSys, curve);
```

The code above creates the Curve using parameters defined elsewhere in the code of a CoordSys (`csys`) and a LineString (`lineString`). A new MultiCurve is then created using the CoordSys property of the Curve and the Curve itself.

In the example above, as in all FeatureGeometries created from objects, a copy of the original object is created because the reference cannot be shared.

Curve

The Curve class inherits from the CurveSegmentList class, and represents a contiguous linear Geometry. Curves contain a collection of CurveSegments that must remain contiguous. This class is included in the model to allow for future expansion and is part of the OGC standards.

Use the following example code to model the creation of a Curve:

```
using MapInfo.Geometry;

DPoint[] points = new DPoint[4];

points[0]= new DPoint(-88.135215,43.998892);
points[1]= new DPoint(-104.875119,43.998892);
points[2]= new DPoint(-120.242895,47.048364);
points[3]= new DPoint(-89.135215 46.998892);

LineString lineString = new LineString(csys, points);
Curve curve = new Curve(csys, lineString);
```

CurveSegments

At present a CurveSegment can only be a LineString. The class is designed to expand in future iterations of the product to include Spline, CircularArc, and EllipticalArc CurveSegments. Curves and Rings are comprised of CurveSegments.

Rings

A Ring is a collection of CurveSegments which must remain contiguous and closed.

Use the following example code to model the creation of a Ring:

```
using MapInfo.Geometry;

dPoints = new DPoint[102];
dPoints[0] = new DPoint(-109.171279,49.214879);
dPoints[1] = new DPoint(-109.169283,49.241794);
...
dPoints[101] = new DPoint(-109.171279,49.214879);
Ring newRing = new Ring(longLatNad83, CurveSegmentType.Linear, dPoints);
```

Polygon

A Polygon is an object made up of Rings. A polygon must have at least a single Ring which defines the exterior boundary of the Polygon. Other Rings can be included inside which then define holes in the Polygon. Once a Ring is placed inside of another Ring the object becomes a MultiPolygon.

Use the following example code to model the creation of a Polygon.

```
using MapInfo.Geometry;

DPoint[][] points = new DPoint[1][];
points[0] = polyPointArrays[0];
Polygon polygon = new Polygon
     (longLatNad83, CurveSegmentType.Linear, polyPointArrays[0]);
```

Checking for Points in Polygons

The following code example shows how to determine whether a point is inside the boundary of a FeatureGeometry (Multipolygon), on the boundary line or falls outside of it.

```
Public Shared Sub MapInfoGeometryContainsPoint()
Dim coordSysFactory As CoordSysFactory = Session.Current.CoordSysFactory
Dim coordSys As CoordSys = _
   coordSysFactory.CreateLongLat(MapInfo.Geometry.DatumID.NAD83)

Dim points(6) As DPoint
   points(0) = New DPoint(-0.705036, -0.122302)
   points(1) = New DPoint(-0.446043, 0.486811)
   points(2) = New DPoint(0.235012, 0.36211)
   points(3) = New DPoint(0.422062, -0.304556)
   points(4) = New DPoint(-0.244604, -0.71223)
   points(5) = New DPoint(-0.705036, -0.122302)
Dim multiCurve As MultiCurve = New _
   MultiCurve(coordSys,CurveSegmentType.Linear,points)
Dim multiPolygon As MultiPolygon = New _
   MultiPolygon(coordSys,CurveSegmentType.Linear,points)

Dim insidePoint As DPoint = New DPoint(-0.115108,0.160671)
Dim boundaryPoint As DPoint = New DPoint(-0.446043,0.486811)
Dim outsidePoint As DPoint = New DPoint(-1.103118,0.021583)

   If multiPolygon.ContainsPoint(insidePoint) Then _
      Console.WriteLine("Points inside area inclosed by closed _
      (GeometryDimension 2) objects are contained")
   End If
   If Not multiCurve.ContainsPoint(insidePoint) Then _
      Console.WriteLine("But this is not true for linear _
      (GeometryDimension 1) objects")
   End If
   If multiPolygon.ContainsPoint(boundaryPoint) Then _
      Console.WriteLine("Points on the boundary of closed objects _
      are contained")
   End If
   If multiCurve.ContainsPoint(boundaryPoint) Then _
      Console.WriteLine("Points lying on linear objects are contained")
   End If
```

```
   If Not multiPolygon.ContainsPoint(outsidePoint) Then _
     Console.WriteLine("Point completely outside closed objects _
     are not contained")
   End If
   If Not multiCurve.ContainsPoint(outsidePoint) Then _
     Console.WriteLine("Point completely outside linear objects _
     are not contained")
   End If
End Sub
```

Coordinate Systems

Coordinate systems describe the domain in which a particular object or set of objects reside. The coordinate system allows for the delineation, in specific terms, of the object or objects being described. The CoordSys classes contain methods, properties and interfaces that allow for the creation, manipulation, and editing of coordinate systems.

When Geometries are created, they are created in a particular coordinate system specified in the creation of the object. Objects cannot change the coordinate system in which they were created. They can only be copied into another coordinate system.

The CoordSys class facilitates the creation and manipulation of coordinate systems. The Coordsys class uses an XML version of the projection file (C:\Program Files\MapInfo\Professional).

The CoordSysFactory object contains registered coordinate systems. CoordSys definitions can be registered by loading one or more XML projection files or by using the RegisterCoordSys, or RegisterCoordSysInfo methods. Create CoordSys objects from the factory, or code-codespace (EPSG, SRID), PRJ string, MapBasic string, and other Factory creation methods. There are also Military Grid Reference System conversion methods in the CoordSys class.

Creating a CoordSys Object

The following sample code shows the creation of CoordSys objects several different ways: using a MapInfo codespace; through EPSG; as longitude/latitude from a PRJ string; from a MapBasic string; and through SRID.

VB example:

```
Public Shared Sub MapInfoGeometryCreateCoordSys()
    Dim factory As CoordSysFactory =  Session.Current.CoordSysFactory

    ' create CoordSys objects from srsName
    Dim csysWGS84 As CoordSys = factory.CreateCoordSys("EPSG:4326")
```

```
    Dim csysNAD83 As CoordSys = factory.CreateCoordSys_
      ("mapinfo:coordsys 1,74")
    Dim csysNAD27 As CoordSys = factory.CreateCoordSys("SRID:8260")

    ' create CoordSys objects from code/codeSpace
    csysWGS84 = factory.CreateCoordSys("4326", CodeSpace.Epsg)
    csysNAD83 = factory.CreateCoordSys("coordsys 1,74", CodeSpace.MapInfo)
    csysNAD27 = factory.CreateCoordSys("8260", CodeSpace.Srid)

    ' create CoordSys objects from user-defined parameters
    Dim dat As Datum = factory.CreateDatum(DatumID.WGS84)
    csysWGS84 = factory.CreateCoordSys(CoordSysType.LongLat, _
        dat, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, nothing)
    dat = factory.CreateDatum(DatumID.NAD83)
    csysNAD83 = factory.CreateCoordSys(CoordSysType.LongLat, _
        dat, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, nothing)
    dat = factory.CreateDatum(DatumID.NAD27ContinentalUS)
    csysNAD27 = factory.CreateCoordSys(CoordSysType.LongLat, _
        dat, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, nothing)
    ' create Long/Lat coordinate system
    csysWGS84 = factory.CreateLongLat(DatumID.WGS84)
    csysNAD83 = factory.CreateLongLat(DatumID.NAD83)
    csysNAD27 = factory.CreateLongLat(DatumID.NAD27ContinentalUS)

    ' create from MapBasic string
    Dim csysRGF93 As CoordSys = _
        factory.CreateFromMapBasicString("CoordSys Earth Projection 3, _
        33, ""m"", 3, 46.5, 44, 49, 700000, 6600000")
    ' create from PRJ string
    csysNAD83 = factory.CreateFromPrjString("1, 74")
Sub
```

Changing the Coordinate System of a Geometry Object

The next example illustrates how to convert a Geometry object from one coordinate
system to another.

VB example:

```
Public Shared  Sub MapInfoGeomeTryCoordSys(ByRef coordSys As _
    CoordSys, ByRef points() As DPoint, ByRef alternateCoordSys _
    as CoordSys
  ' All Geometry constructors require a CoordSys parameter
  ' Note that the points array is assumed to be in coordSys
      Dim lineString As LineString =  New _
        LineString(coordSys,points)

  ' The Geometry has a reference to the CoordSys used during
  ' construction. Unlike the coordinate data represented by the
  ' points array, the CoordSys' is not copied
    If ReferenceEquals(coordSys, lineString.CoordSys) Then
```

```
        Console.WriteLine("Geometry objects hold a reference to _
           the CoordSys used during construction")
        End If

  ' if you want to convert the object to another coordinate
  ' system, you need to make a new copy using one of the copy methods
     If Not coordSys.Equals(alternateCoordSys) Then
        Dim newGeometry as MapInfo.Geometry.Geometry = _
           lineString.Copy(alternateCoordSys)
        End If
     End Sub
```

Adding Coordinate Systems

If the MapInfoCoordinateSystemSet.xml file does not contain a coordinate system to match your needs, you may add it, for the library to reference. This feature supports adding EPSG codes and SRID codes to extend the library's capabilities.

EPSG codes represent a collection of coordinate systems (known as codespaces) maintained in the EPSG Geodetic Parameter Dataset under the auspices of the International Association of Oil & Gas Producers (OPG). The OPG's Survey and Positioning Committee took over this responsibility from the European Petroleum Survey Group in 2005.

SRID codes are unique spatial reference numbers that refer to codespaces for Oracle Spatial tables.

(MDAL supports a third codespace called MapInfo.)

MDAL provides you with many of the common EPSG and SRID mappings. If you need to register a different EPSG or SRID code to a particular coordinate system, this feature provides you with two methods to do so.

To extend MDAL's ability to use any EPSG or SRID codespace, you may add the information programmatically, in which case, the coordinate system information will only last as long as the MDAL Session. Or you may add it to your app.config file as a more permanent solution. Each is discussed below.

Register EPSG and SRID Codes Programmatically

The MapInfo.Geometry.CoordSysFactory class contains methods that allow you to register EPSG and SRID codes to a specified coordinate system.

RegisterEPSGCode() and RegisterSRIDCode() each take two parameters: one being the EPSG or SRID code that represents the codespace, the second is the coordinate system information that first parameter will map to.

The following example demonstrates registering a fictional code with the Long/Lat NAD83 coordinate system.

VB example:

```
Public Shared Sub MapInfo_Geometry_RegisterEPSGCode()
  Dim factory As CoordSysFactory = Session.Current.CoordSysFactory
  ' create CoordSys objects from srsName
  Dim csysNAD83 As CoordSys = _
      factory.CreateCoordSys("mapinfo:coordsys 1,74")
  ' 9998 is a fictional code for demonstration purposes
  Try
    factory.RegisterEPSGCode(9998, csysNAD83)
  Catch ae As ApplicationException
    'code already exists.  Codes cannot be duplicated
  End Try
End Sub
```

ⓘ   If the EPSG or SRID code already exists, an exception will be thrown indicating this fact.

To determine if a coordinate system for the MapInfo, EPSG or SRID codespace is already supported, call this method:

- `MapInfo.Geometry.CoordSys.Code(codespace).`

This method returns the first (or only) occurrence of the codespace that matches or null, if it does not exist.

Similarly, to return the first SRSName in the list that matches the input codespace, call this method:

- `MapInfo.Geometry.CoordSys.SRSName(codespace).`

An SRSName (Spatial Reference System) represents the name of a coordinate reference system written in GML (Geography Markup Language). This is typically, a friendly name for the coordinate system, not a list of parameter values.

To get a list of all the codes and coordinate systems that are mapped to a particular coordinate system, MDAL provides the following methods:

- `MapInfo.Geometry.CoordSys.Codes(codeSpace)`
- `MapInfo.Geometry.CoordSys.SrsNames(codeSpace)`

Keep in mind that the coordinate system information you added programmatically, will only be maintained during the lifetime of the MDAL Session.

107   **Developer Guide**

Register EPSG and SRID Codes to a Web or Desktop Configuration File

The second, and more permanent, way to add EPSG or SRID codes to your MDAL app, is by adding the information to your application app.config[1] file. The code below in bold shows the information to copy and paste into your config file. An explanation of the code follows.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <configSections>
      <section name="MapInfo.DataAccess"
type="MapInfo.Engine.ConfigSectionHandler, MapInfo.DataAccess, Version=17.0,
Culture=neutral, PublicKeyToken=93e298a0f6b95eb1" />
   </configSections>
        <appSettings>
           <add key="MapInfo.Engine.Session.UseCallContext" value="false" />
        </appSettings>
   <MapInfo.CoreEngine>
     <EPSG_Code_Mappings>
        <EPSG_Code_Mapping>
           <srsName>My Custom CRS</srsName>
           <srsID>
              <code>coordsys 8,74,8,-
110.0833333333,47.5,0.9999375,2624666.667,328083.3333</code>
              <codeSpace>mapinfo</codeSpace>
              <remarks>My Custom CRS</remarks>
           </srsID>
           <EPSG_Codes>
              <EPSG_Code>9987</EPSG_Code>
              <EPSG_Code>9988</EPSG_Code>
              <EPSG_Code>9989</EPSG_Code>
           </EPSG_Codes>
        </EPSG_Code_Mapping>
     </EPSG_Code_Mappings>
     <SRID_Code_Mappings>
        <SRID_Code_Mapping>
           <srsName>My Custom CRS</srsName>
           <srsID>
              <code>coordsys 8,74,8,-
114.0833333333,47.5,0.9999375,2624666.667,328083.3333</code>
              <codeSpace>mapinfo</codeSpace>
              <remarks>My Custom CRS</remarks>
           </srsID>
           <SRID_Codes>
              <SRID_Code>9990</SRID_Code>
              <SRID_Code>9991</SRID_Code>
              <SRID_Code>9992</SRID_Code>
           </SRID_Codes>
```

1.   If your desktop application does not have an app.config file, you can create one by adding it to your project from the Visual Studio Application Configuration File template.

```
        </SRID_Code_Mapping>
      </SRID_Code_Mappings>
    </MapInfo.CoreEngine>
</configuration>
```

The code above shows that there are two sections of information to add. One is to identify the correct CoreEngine dll and version number[1], the second is to add elements for EPSG and SRID code mappings.

An EPSG code mapping consists of an SRSName, SRS ID, and EPSG code(s). The SRSID is further defined by the parameters and codespace for the coordinate system.

An SRID code mapping is similar to the EPSG code mapping, except it refers to the Oracle Spatial identification number.

---

1. To determine the correct version number for the MapInfo.DataAccess assembly, examine the properties of the installed MapInfo.DataAccess.dll (e.g., "C:\Program Files\MapInfo\Professional\MapInfo.DataAccess.dll")

7 – Working with GeoPackage

MDAL provides support for opening, displaying, creating and editing GeoPackage files which is an open source format created by OGC.

In this chapter:

Overview

GeoPackage is an open format for Geospatial Information defined by OGC (http://www.geopackage.org). It is a SQLite-based extension defined by the OGC to promote portability of data across platforms and products.

According to OGC definition for GeoPackage:

"*A GeoPackage is a platform-independent SQLite database file that may contain:*

- *vector geospatial features*
- *tile matrix sets of imagery and raster maps at various scales*
- *metadata*

Since a GeoPackage is a database, it supports direct use, meaning that its data can be accessed and updated in a "native" storage format without intermediate format translations. GeoPackages are interoperable across all enterprise and personal computing environments, and are particularly useful on mobile devices like cell phones and tablets in communications environments with limited connectivity and bandwidth.

This OGC® Encoding Standard defines the schema for a GeoPackage, including table definitions, integrity assertions, format limitations, and content constraints. The allowable content of a GeoPackage is entirely defined in this specification."

Opening a GeoPackage file

GeoPackage table and GeoPackage Tab files can be opened programmatically.

> ⓘ If the GeoPackage table being opened have a Coordinate System which is not supported by MapInfo Pro then that table will not be opened.

To open a GeoPackage file programmatically use the code below:

```
TableInfoGeopackage infoGeoPackage = new TableInfoGeopackage("table alias");
infoGeoPackage.DatabasePath = tablePath; //Path to the GeoPackage file .gpkg
infoGeoPackage.DatabaseTableName = "Table Name"; //Table to be opened in
GeoPackage
TableInfo tableInfo = infoGeoPackage;//Create the Table Info object.
Table
openTable=MapInfo.Engine.Session.Current.Catalog.OpenTable(tableInfo);//Open
table
```

A GeoPackage database may contain multiple tables so the database table name is required.

Developer Guide

ⓘ   MDAL only supports Feature Tables of GeoPackage.

The following code sample illustrates getting the list of tables in the GeoPackage database:

```
GeopackageDataSourceDefinition gpkgDSDef = new
GeopackageDataSourceDefinition("c:\data\test.gpkg"); //path to gpkg file
GeopackageDataSource gpkgDS =
GeopackageDataProvider.Instance.OpenDataSource(gpkgDSDef, null) as
GeopackageDataSource;
List<IDataSourceNamedTable> tablesList = gpkgDS.GetSchemaNamedTables("main",
new string[] { "" }) as List<IDataSourceNamedTable>; //Get the list of tables
in database.
```

Opening a GeoPackage Tab file

A GeoPackage tab file, which references a Feature Table by name within a GeoPackage database, may be created using MDAL (programmatically) or MapInfo Pro. Multiple tab files may reference to the same database since one database may contain multiple Feature Tables.

To open a GeoPackage Tab file, use the code below:

```
TableInfo infoGeoPackage = TableInfoGeopackage.CreateFromFile("Path to Tab
file");
Table openedTable = Session.Current.Catalog.OpenTable(infoGeoPackage);
```

Opened table will have table type as GeoPackage

```
openedTable.TableInfo.TableType == TableType.Geopackage
```

A GeoPackage tab file created using MapInfo Pro can be opened in MDAL and vice versa.

Enable GeoPackage as cache for RDB (SQL/Oracle) tables

You have the option to use GeoPackage as Cache for RDB Tables. Following Metadata properties should be configured for to enable this functionality.

```
TableInfoServer tis = new TableInfoServer("GeoPackageCacheAPITest");
tis.ConnectString = "Driver={SQL Server Native Client
11.0};DATABASE=QADB;Server=ServerName;UID=UserName;PWD=Password";//
"DSN=ServerDSN ";
tis.Query = "Select *  From Table";
```

```
tis.Toolkit = ServerToolkit.Odbc;
CacheParameters cp = new CacheParameters(CacheOption.All);
cp.StorageType = CacheStorageType.Geopackage;    //Cache Type as Geoackage.
tis.CacheSettings = cp;
```

Multiple RDB tables are cached in the same GeoPackage Cache database. The benefit for using the GeoPackage for the cache is that MDAL will use fewer temporary files which is important for environments where the number of available file handles becomes limiting.

ⓘ   Note: When using GeoPackage as a cache format, MDAL will store information that is not supported by the standard (such as coordinate system information and styles information). It is advisable not to use Cache file directly.

Create and Save GeoPackage file programmatically

A GeoPackage table can be created using the TableInfoGeopackage API. This will allow you to create and save a .gpkg file along with its Tab file at the specified path.

```
TableInfoGeopackage tig = new TableInfoGeopackage("GeoPackageTest");
tig.Temporary = true;
Column col = new Column();
tig.Columns.Add(ColumnFactory.CreateIndexedIntColumn("ROWNUM"));
tig.Columns.Add(ColumnFactory.CreateStringColumn("StrName", 40));
tig.Columns.Add(ColumnFactory.CreateIndexedDoubleColumn("Doubleval"));
CoordSysFactory csf = new CoordSysFactory();
CoordSys wgs84 = csf.CreateLongLat(DatumID.WGS84);
tig.Columns.Add(ColumnFactory.CreateFeatureGeometryColumn("GEO", wgs84));
tig.Columns.Add(ColumnFactory.CreateStyleColumn());
tig.DatabasePath = _tempPath + "GeoPackageTest.gpkg";
tig.DatabaseTableName = "GeoPackageTest";
tig.TablePath = _tempPath + "GeoPackageTest.TAB";
Table _miTable = Session.Current.Catalog.CreateTable(tig);
```

The above code will create a GPKG database along with its TAB file and will save that file at TablePath. If GPKG database already exists then the table will be added to the existing Database.

Once the table gets created Features can be inserted, updated and deleted from the table. MDAL can only create GeoPackage tables for which EPSG code and OGC "Well Known Text" description is available for the Coordinate System. The list of supported Coordinate Systems in MDAL can be found in "MapInfoCoordinateSystemSet.xml" at installation path "C:\Program Files\MapInfo\Professional".

ⓘ Note: As per GeoPackage Specification Requirements - " Every feature table or view in a GeoPackage SHALL have a column with column type INTEGER and PRIMARY KEY AUTOINCREMENT column constraints".

Due to the rule above when a new GeoPackage Table is created, an ID column is added to the column list by default. So the column order of the newly created table may differ from the order supplied.

The GeoPackage standard does not support capture of styling information with the data. As a result, MDAL will apply a default styling to the data. Users should prefer to use layer style overrides and/or themes on their GeoPackage feature layers to display the data in the desired way.

MapInfo Pro will store default styling in the .TAB file which MDAL will recognize and apply it on the GeoPackage table.