11 Chapter

Features and Searches

A point on a map representing New York City is an example of a Feature object. The search methods of the Layer object allow you to “mark” or choose Features that meet certain criteria. For example, you need to find all of the potential clients within 150 miles of New York City.Once you create this collection of data, you may cycle through the collected data, print it out, take averages, count how many met that criteria, save them to a file, or perform other tasks. In this chapter we will take a look at Features and searches.

The Feature Object

A map Feature is a geographic object on a map such as a point, line, or region. For example, a map of the World could contain regions as countries, lines as highways, and points as cities. In MapXtreme, a map Feature is represented as a Feature object. For example, the UK could be a Feature object of type region, the A10 highway a Feature object of type line, and London a Feature object of type point.

Anyone who has worked with databases is familiar with a record. A record is set of related columns of information. For example, a database of customers will have a record for each customer that includes columns for name, address, interest, etc. A Feature is simply a record that combines tabular data and geometric information. For example, the fileWorld.tab from the MapXtreme sample data is a MapInfo format database. For each country, there is a record. Each record includes several columns of tabular data as well as a reference to the geometric information that describes the shape and location of each country; this allows it to be displayed on the map. The tabular data is referred to as attribute data and the geometric data is referred to as the geometry. These two types of data make a Feature. The following illustration is a conceptual view of a Feature:

Country	Capital	Pop_1994	Gr_Rt	Pop_Male	Geometry
China	Beijing	1,136,429,638	2.2	584,836,207	
Mexico	Mexico City	81,249,645	2.2	39,893,969	
United States	Washington, D.C.	257,907,937	0.8	125,897,610	

Methods of the Feature Object

The Feature object has methods that obtain information about the tabular and geometric data. The following table lists these methods:

Method	Description
getAttribute	Gets the specified attribute given the column index.
getAttributeCount	Gets the number of attributes associated with this Feature.
getGeometry	Gets the associated Geometry or null if the Feature has no geometry.
getPrimaryKey	Gets a PrimaryKey object (unique ID) for this Feature. A null value will be returned if the Feature does not have a PrimaryKey.
getRaster	Will return the raster object associated with the Feature if one exists or null if the Feature does not have a Raster.
getRendition	Returns the rendition for this Feature. A null value will be returned if there is no rendition for the Feature.

Attributes

Each Feature can have one or more Attribute objects. Attribute objects represent a column of tabular data for the Feature. This object consists of the type and value of information. For example, an attribute may have a type of double and a value of 2.2 that represents growth rate.

Geometries

Each Feature can have a Geometry object. The Geometry object allows access to all of the geometric information for the Feature. The geometric information may be VectorGeometry or PointGeometry. The VectorGeometry is used for Features that ar polylines or regions. The PointGeometry is used for points.

Renditions

Each Feature can have a Rendition object. The Rendition object describes the display characteristics of the Feature. The Feature object can only get existing rendition information. It tells you how the Feature is being displayed. To change rendition

information for existing Features, you should use a theme, such as the OverrideTheme object.

Raster

Each Feature can have an MIRaster object. When a raster image is associated with a Feature, you can retrieve binary information that describes an image. When an object has a raster, it may also have a geometry in which case the geometry describes the bounds of the raster image.

Creating Features Using FeatureFactory

MapXtreme allows you to create, modify, or delete Features (points, lines, polylines, regions) and add them to an Annotation layer or JDBC layer. There are two ways to create new map Features. You can create a Feature using the FeatureFactory object, or create Features by retrieving existing Features using search methods of the Layer class. Once the Feature is created or retrieved, it is typically added to an Annotation layer or layer defined by a table in a JDBC database in order to be useful.

This section covers creating Features using the FeatureFactory. It is followed by a section on search methods that return Features in a FeatureSet. The chapter concludes with a section on editing Features.

FeatureFactory Methods

The methods of the FeatureFactory object let you create new map Features that represent points, lines, polylines, and regions. They are:

- createPoint
- createPolyline
- createRegion
- createCircularRegion

These methods return stand-alone Feature objects. To create any Feature through the FeatureFactory you specify a Rendition, an array of Attributes associated with that Feature, the Geometry for the Feature, and the PrimaryKey

When creating a Feature to be added to a JDBC table layer, the Feature will ultimately be stored as a row in the database table. The array of Attributes provided to the FeatureFactory are the column values for this new row. These values must match the ordering of the column names retrieved from the layer'sTableInfo.

The Geometry for the Feature must be specified in the numeric coordinate system of MapJ. If the new Feature is being added to a JDBC table layer that has a different coordinate system, MapXtreme Java will handle any coordinate transformation that is required.

To create a point, specify the centerpoint, its Rendition, such as symbol size, font and color, and array of Attributes.

For describing the geometry of a circle, in addition to specifying rendition and attributes, you have additional parameters including resolution, and whether the circle is described using display or numeric coordinates. The resolution defines the number of nodes to use when making the approximating polygon and the first parameter controls whether the Feature is a display or numeric circle. The circle is either drawn in the coordsys of the screen (display) or of the map (numeric). The display version will generally look better (i.e., more like a circle) on the screen, whereas the numeric one may appear oblong since it is following the earth's curve.

Creating a region or polyline involves creating a double point array to describe the geometry.

Code Example

This example illustrates how to create each of the Feature types in the FeatureFactory: points, circles, lines, and polylines. To create a region, follow the polyline example to create an array of points that gets passed to createRegion().

```
// Get Feature Factory reference

FeatureFactory ff = map.getFeatureFactory();

// Set up Attribute object

Attribute Att[] = new Attribute[1];

Att[0] = new Attribute("Feature1");

// Set up rendition object

Rendition rend = new Rendition();

// For circles and regions, specify the edge and fill
   color.

rend.setValue(Rendition.STROKE, Color.cyan);
```

```
rend.setValue(Rendition.FILL, Color.green);
// For points, specify the symbol size, font, and color
rend.setValue(Rendition.SYMBOL_STRING, 52);
rend.setValue(Rendition.FONT_FAMILY, "MapInfo Shields");
rend.setValue(Rendition.SYMBOL_FOREGROUND, Color.green);
// For lines, specify the line color and width
rend.setValue(Rendition.STROKE, Color.cyan);
rend.setValue(Rendition.STROKE_WIDTH, 4);
// Set the center point Features
DoublePoint dp = new DoublePoint(-104, 45);
// Create Circular region
int circType=1;
int circRadius=25;
int circResolution=25;
Feature retFeature;
retFeature = ff.createCircularRegion(circType, dp,
   circRadius, LinearUnit.mile,circResolution, rend, Att,
   null);
// Create Point
retFeature = ff.createPoint(dp, rend, Att, null);
// Create PolyLine (or Region)
double pts[] = new double[6];
pts[0] = new double(-104); //x1
pts[1] = new double(45); //y1
pts[2] = new double(-102); //x2
pts[3] = new double(46); //y2
pts[4] = new double(-100); //x3
pts[5] = new double(45); //y3
retFeature = ff.createPolyLine(pts, rend, Att, null);
```

FeatureSet Collection

A FeatureSet is a collection of Features. In MapXtreme, the different layers that make up your map usually have the same Feature type within each layer. For example, the "World" layer has region Features to represent each country,the "US Highways" layer has line Features to represent major U.S. highways, and the "World Capitals" layer has point Features to represent each country's capital city. The search methods of the Layer object return a FeatureSet collection from a layer. The following methods allow you to manipulate the FeatureSet object:

Method	Description
dispose	Disposes the resources used by the FeatureSet. This must be called once you are done with the FeatureSet.
getNextFeature	Gets the next Feature in the set.
getRendition	Gets the base Rendition for all Features in this FeatureSet.
getTableInfo	Gets the TableInfo (metadata) describing this FeatureSet.
isRewindable	Determines the **rewindable** status for this object.
rewind	Rewinds the FeatureSet prior to the first Feature in the FeatureSet.

In order to minimize memory allocations, MapXtreme Java may reuse the same internal memory when returning a Feature from the getNextFeature method. If you need to hold on to all or parts of a Feature beyond the next call to getNextFeature, make your own copy of the object(s) you would like to persist. This means that FeatureSets can only be traversed in a forward direction, and that once you pass a Feature you cannot return to it.

Some FeatureSets may be rewindable, which means that the FeatureSet can be reset to its first Feature. Whether a FeatureSet returned from a search method is rewindable is an implementation detail of each Data Provider. If a FeatureSet is not rewindable you can create a rewindable FeatureSet from a non-rewindable one, then the FeatureSet can once again be traversed.

Here is an example of rewinding a FeatureSet:

```
If(!fs.isRewindable() )
{
   fs = new RewindableFeatureSet(fs);
}
```

When you are done using FeatureSets the dispose method should always be called.

Searching

One of the most powerful capabilities of MapXtreme is searching. Searching allows you to retrieve specific data according to geographic information. For example, if you were looking for all of the cellular towers within a 25 mile radius, you would perform a search. Searches are methods of the Layer object. They return FeatureSet objects. A fundamental function of MapXtreme is selecting Features on the map, so that you can perform additional tasks on them. Users can click on the map to select one or more Features (points, lines, regions, etc.). Search results are often interpreted as selections.

The following methods of the Layer object provide various ways to search a Layer and return a FeatureSet collection.

- searchAll
- searchWithinRadius
- searchWithinRegion
- searchWithinRectangle
- searchAtPoint
- searchByAttribute
- searchByPrimaryKey

All searches are passed the names and query parameters of the columns to be returned. The names of the columns you want returned from any search should be put into a vector object.

The following is an example of creating a vector of column names for all columns in the specified table:

```
//Assume myLayer is a Layer object.
TableInfo myTableInfo = myLayer.getTableInfo( );
Vector columnNames = new Vector( );
int columnCount = myTableInfo.getColumnCount( );
String col;
for (int j=0; j<columnCount; j++)
{
col = myTableInfo.getColumnName(j);
columnNames.addElement(col);
}
```

The characteristics of the Features returned from a search on a Layer depend on several optional parameters. By default, a Feature's associated Geometry, Rendition, PrimaryKey, preferred label position, and raster data are returned with any query. If you wish to limit the information that is returned for a Feature, use the QueryParams class. This will improve query performance.

The QueryParams class also specifies the SearchType for the query.The Features returned from a query are dependent on the search type specified as part of the query. Queries using a search type of *mbr* return Features whose minimum bounding rectangle intersects the search region. This search type is least restrictive and returns the maximum number of Features. Queries using a search type of *partial* return Features that intersect the search region. Queries using a search type of *entire* return Features that are completely contained within the search region. It is the most restrictive search type. If you don't use QueryParams, the SearchType defaults to *mbr*.

The following is an example of creating a QueryParams object:

```
QueryParams qp = new
QueryParams(bGeometry,bRendition,bPrimarykey,bLabelPoint,
   bRasterInfo,SearchType.entire);
```

This example shows how the QueryParams object limits the information returned in a search:

```
// find all Features entirely within a given

// search region, return a single Attribute column

// and no Rendition information

Vector cols = new Vector();

cols.addElement("County");

Feature searchFeature // =
   mapj.getFeatureFactory().createRegion(points, rend,
   attribs,null);

QueryParams queryParams = new QueryParams(true, false,
   true,true, true, SearchType.entire);

FeatureSet fs = layer.searchWithinRegion(cols,
   searchFeature.getGeometry(), queryParams);
```

Search Methods

This section defines each search method available and code to demonstrate its use.

searchAll

Returns a FeatureSet collection with all Features from the layer. Use this search if your application requires you to loop through an entire layer.

```
//Assume columnNames is a vector of the columns to be
   returned.

//Assume qp is the QueryParams object.

//Assume myLayer is a Layer object.

try

{

   FeatureSet = myLayer.searchAll(columnNames,qp);

}

catch(exception e)

{

   e.printStackTrace();

}
```

searchWithinRadius

Returns a FeatureSet collection made up of Features within a specified distance of a point object. This search can be used to locate the nearest dealer to a given location, or it could return the number of customers within a certain radius of a store.

```
//Assume columnNames is a vector of the desired columns
   to
//be returned.
//Assume qp is the QueryParams object.
//Assume myLayer is a Layer object.
DoublePoint dblPt = new DoublePoint(-
   73.889444,42.765555);
double dRadius = 10.03;
try
{
     FeatureSet = myLayer.searchWithinRadius
   (columnNames,dblPt,dRadius,LinearUnit.mile,qp);
}
catch(exception e)
{
   e.printStackTrace();
}
```

searchWithinRegion

This search method returns a FeatureSet collection made up of Features within the geometry of a Feature. Use this method to return the number of customers in a specific region, such as postal code, or return the Features that fall within a region created with the FeatureFactory.

```
private boolean layerSearchWithinRegion()
{
 //Assume columnNames is a vector of the columns to be
   returned.
 //Assume qp is the QueryParams object.
 //Assume myLayer is a Layer object.
 //Assume vGeom is a VectorGeometry of TYPE_REGION
 try
{
     fs =
   myLayer.searchWithinRegion(columnNames,vGeom,qp);
    }
 catch(Exception e)
   {
     e.printStackTrace();
     return false;
}
 return true;
}
```

searchWithinRectangle

This search method returns a FeatureSet collection within bounds of specified rectangle. This method could be used to search within the given map window or to pretest a zoom level to see if it will incorporate certain points of interest.

```
//Assume columnNames is a vector of the columns to be
  returned.

//Assume qp is the QueryParams object.

//Assume myLayer is a Layer object.

DoubleRect dRect = new DoubleRect(-74.092662,42.765555,-
  73.668898,42.856420);

try

{

 FeatureSet =

   myLayer.searchWithinRectangle(columnNames,dRect,qp);

}

catch(exception e)

{

 e.printStackTrace();

}
```

searchAtPoint

This search method returns a FeatureSet collection that is made up of Features at a specified point. This method could be used to test for all objects intersecting a certain point. It could be used to test if a given location falls within a certain coverage area.

```
//Assume columnNames is a vector of the columns to be
   returned.

//Assume qp is the QueryParams object.

//Assume myLayer is a Layer object.

DoublePoint dp = new DoublePoint(12.3456,-67.890)

try

{

 FeatureSet = Layer.searchAtPoint(columnNames,dp,qp);

}

catch(exception e)

{

 e.printStackTrace();

}
```

searchByAttribute

This search method returns a FeatureSet collection whose Attribute matches the given attribute. This method could be used to select all Features with a common piece of attribute information. For example if you had a table of world countries that included a column of currency type, you could do a searchByAttribute to return all of the countries that use the Euro.

```
//Assume columnNames is a vector of the columns to be
   returned.
//Assume qp is the QueryParams object.
//Assume myLayer is a Layer object.
//Assume attr is the attribute that to search against.
//Assume colName is the column to search against.
try
{
  FeatureSet fs =

   myLayer.searchByAttribute(columnNames,colName,attr,qp)
   ;
}
catch(Exception e)
{
 e.printStackTrace();
 return false;
}
```

searchByPrimaryKey

This search method returns a FeatureSet Collection with PrimaryKeys that match the PrimaryKeys in a given array of PrimaryKeys. Use this search if you have previously returned a FeatureSet. If you want to use the information in the future, but not want to hold onto the entire FeatureSet, you can just store the PrimaryKey. When you want the same FeatureSet back, use the **searchByPrimaryKey** method.

```
private boolean layerSearchByPrimaryKey()
{
 // Assume columnNames is a vector of the columns to be
   returned.
 // Assume qp is the QueryParams object.
 // Assume myLayer is a Layer object.
 // Assume attr is the attribute that to search against.
 // Assume colName is the column to search against.
 // Assume pk is a PrimaryKey from a previous search.
 PrimaryKey[] arraypk = new PrimaryKey[1];
 arraypk[0] = pk;
 try
 {
 FeatureSet fs =
   lyr.searchByPrimaryKey(columnNames,arraypk,qp);
 }
 catch(Exception e)
    {
     e.printStackTrace();
     return false;
 }
 return true;
 }
}
```

Searching Layers Defined by SQL Queries

MapXtreme Java executes user-defined SQL queries without making any modifications to the query.Referred to as "pass-through" queries, MapXtreme will execute them as written and retrieve all the Features into the layer. Note that the query could return many undisplayed Features, for example, when zoomed in on a densely featured layer.

Pass-through queries are intended for advanced users of MapXtreme Java who need complex queries to construct layer data and understand how to include the appropriate limiting conditions.

QueryBuilder Interface

To assist power users with the limitations posed by pass-through queries, MapXtreme Java provides an interface that allows you to write your own call back objects to create modified query strings when rendering or performing searches on layers defined by pass-through queries. A QueryBuilder object is given to a pass-through layer, which invokes its methods when needed.

During map rendering, if MapXtreme Java encounters a layer defined by a pass-through query containing a QueryBuilder, the QueryBuilder method **queryInRectangle** is invoked to provide the query string that is passed to the Renderer. The QueryBuilder is provided with all the data needed to construct a new query string that contains the limiting geometric condition that limits the Features returned to only those visible in the display viewport. If that layer does not have a QueryBuilder, it will likely cause significant inefficiency when rendering it as many more Features may be returned than are displayed. (You can determine the number of Features returned that were not rendered by running your application with verbose turned on.)

Searching a pass-through layer by invoking any of the search methods requires the query to be modified by adding a where clause and/or changing columns in a select clause. Each search method invokes its counterpart method on the QueryBuilder interface and uses the new query string to perform the search. Without the QueryBuilder, a pass-through layer search will throw an exception. A QueryBuilder is required for searchWithin, searchAt, and searchBy.Only searchAll method does not require a QueryBuilder.

To set the QueryBuilder on a layer object, follow this example:

```
Layer.setQueryBuilder(QueryBuilder myQB);
```

QueryBuilder Considerations

- The QueryBuilder interface is a power user Feature and should only be used when the table definition of a layer is not sufficient.
- QueryBuilder references are not stored with the Map Definition. They must be restored after the Map Definition is loaded.
- The QueryBuilder interface can only be used in client-side applications. QueryBuilder objects are not sent to the server.
- You can use the same QueryBuilder reference for more than one layer.
- Queries returned from a QueryBuilder are executed exactly the same as all pass-through queries.
- Using a QueryBuilder does not change any data that defines the layer.The returned query is executed once and discarded; it does not replace the original query from the TableDescHelper that was used to construct the Layer object.
- The data returned by the QueryBuilder query must have the same primary key definition, dimension, coordinate system, and spatial column (if any) as originally identified in the TableDescHelper. (This is a limitation on the QueryBuilder that will be relaxed in later releases.)

Example Code

Provided in the \sampleapps\QueryBuilders directory of MapXtreme Java is an implementation of OracleQueryBuilder that Oracle users can use as a starting point.

You will also find in sample code for IdentityQueryBuilder, which returns the original input query unchanged. This is useful as a base class for new QueryBuilder development.

The following code is a portion of the OracleQueryBuilder sample that shows one way to construct a query for a query at point. Depending on the purpose of the search and the geometry in the layer, you may want to change the search geometry to a small rectangle, circle or region. For example, if the search geometry is a small rectangle, the mouse click does not have to fall exactly on the Feature, but within the rectangle in order to select it.

```
// Construct a query string to be used when executing a
   query at point.

// @return The SpatialQueryDef defining the new query and
   its metadata

public SpatialQueryDef queryAtPoint(MapJ mapj, Layer
   layer, SpatialQueryDef queryDef, String[] columnNames,
   QueryParams queryParams, DoublePoint point)

throws Exception {

//build SELECT clause

TreeSet selectCols = findRequiredColumns(queryDef,
   columnNames, queryParams);

String selectClause = buildSelectClause(selectCols);

//get SRID

String srid = "NULL";

if (m_bUseSRID) {

int id =
   OracleSRID.getSRIDFromCS(queryDef.getSpatialQueryMetaD
   ata().getCoordSys());

srid = String.valueOf(id);

}

//build WHERE clause

String spatialColumn =
   queryDef.getSpatialQueryMetaData().getGeometryColumn()
   ;

StringBuffer whereClause = new StringBuffer();

whereClause.append("WHERE MDSYS.SDO_RELATE(" +
   spatialColumn + ", MDSYS.SDO_GEOMETRY(1, " + srid + ",
   MDSYS.SDO_POINT_TYPE(0, 0, NULL),
   MDSYS.SDO_ELEM_INFO_ARRAY(1, 1, 1),
   MDSYS.SDO_ORDINATE_ARRAY(" + point.x + ", " + point.y
   + ")), 'mask=ANYINTERACT querytype=WINDOW') =
   'TRUE'");

//construct final query

StringBuffer newQuery = new StringBuffer();
```

```
newQuery.append(selectClause + " FROM (" +
   queryDef.getQuery() + ") " + whereClause);

SpatialQueryDef result = new
   SpatialQueryDef(newQuery.toString(), m_outMetaData);

 if(layer.isVerbose()) {

System.out.println("QueryBuilder new queryAtPoint: " +
   newQuery);

}

return result;
}
```

Feature Editing

MapXtreme allows you to add, modify, or delete Features (points, lines, regions, etc.) that make up an Annotation layer or a layer populated from a table in a JDBC data source. This is done using the Layer class methods:

- addFeature
- addFeatureSet
- replaceFeature
- removeFeature

Features that are added to a Layer can be created in the FeatureFactory or can be the result of performing a search (searchWithin, searchBy, etc.) See page 166 for more on the FeatureFactory or page 170 for a discussion on searching a layer.

Feature editing can only be performed in a client-side application. Be sure that any layers that you plan or may need to edit are created using the LocalDataProviderRef.

Although the interfaces for editing Annotation layers and JDBC layers are the same, the behavior of these two types of layers is different and is addressed in the following sections.

Editing an Annotation Layer

Annotation layers contain Features that can be used to mark or place emphasis on certain areas of the map. Annotation layers are not associated with any persistent data source so changes to Annotation layers are only reflected in the current MapJ. Editing annotation layers only changes the image rendered for that layer

PrimaryKey of Annotation Layer Features

Because the Features that populate the Annotation Layer can come from many different sources, their PrimaryKeys may be defined differently or may not be unique. To overcome this, the Annotation Layer assigns a new PrimaryKey to each new Feature. The key has a single integer attribute that starts numbering at 1 and increments sequentially for each new Feature. Even in cases where the inserted Feature already has a PrimaryKey, a new PrimaryKey is assigned to it. The return value of the **addFeature** method is the new PrimaryKey so it is always possible to obtain the new key if needed.

CoordSys of Annotation Layer Features

For Annotation layers, the Features are expected to be in the numeric coordinate system of the MapJ object. When using the FeatureFactory to create Features, it is necessary to specify the input ordinate arrays in the MapJ numeric CoordSys. When taking Features from a FeatureSet returned by a search method, the Feature is already in the MapJ numeric Coordsys.

Editing a JDBC Table Layer

JDBC layers can be defined by either a database table name or a database pass-through query. However, only those layers defined by a database table name can be edited (i.e., Features added, replaced or deleted) since the change to the layer is actually a change to the source database table.

The changes made to the table must be in accordance with any constraints defined in the table's schema definition. For example, certain columns may be required to be non-null, unique, have numeric values within a certain range, have numeric values greater that zero, or have string values within a certain length, etc. Violation of these constraints will cause the database to throw an Exception.

Additionally, you must have permission to make changes to the database table.

Persistence of Changes to JDBC Table Layer

A successful change to a JDBC table layer results in a change in the source database table. The change will then be visible in the MapJ the next time the layer data is refreshed from the database. MapXtreme treats each Feature edit request as a separate transaction and will immediately commit the change to the database after each request has successfully completed (or will immediately rollback if a change fails). When the request is addFeatureSet, a commit (or rollback) occurs for each individual Feature in the set.

Assignment of PrimaryKey for JDBC Layer Features

Certain conditions exist, depending on the database, regarding the assignment of the PrimaryKey when adding and replacing Features in JDBC layers. The following sections describe those conditions for Oracle8*i*, SpatialWare for Oracle, IUS with SpatialWare DataBlade, and DB2 with SpatialWare Extender.

Oracle8*i* with Spatial Option

The following conditions hold for PrimaryKeys of the Feature being edited. In short, Oracle8*i* assigns the PrimaryKey or uses the one provided.

- The number of columns allowed for the PrimaryKey is 1...n.
- Columns can be of type integer or type character.
- PrimaryKey is required when using addFeature, optional for replaceFeature.
- If you provide a PrimaryKey for a Feature update, the value in the database is overwritten. If you do not provide a PrimaryKey, the old one is kept.
- The value in the PrimaryKey identified for the Feature is assigned to the corresponding column in the database exactly as provided in the Feature.
- PrimaryKey is returned on a successful call to addFeature.

SpatialWare for Oracle and DB2 with SpatialWare Extender

The behavior regarding PrimaryKeys is handled the same way for both SpatialWare for Oracle and DB2 with SpatialWare Extender. In short, the PrimaryKey is set by MapXtreme.

- The number of columns allowed for the PrimaryKey is 1...n.
- Columns can be of type integer only.
- PrimaryKey is not required when using addFeature or replaceFeature. Even if provided, the values in the PrimaryKey for the Feature are ignored.
- The values for the column(s) that comprise the key identified for the table in the TableDescHelper or discovered by MapXtreme if none were identified are automatically set to one greater than the maximum value for the column(s).
- PrimaryKey is returned on a successful call to addFeature.

IUS with SpatialWare DataBlade

The following conditions hold for PrimaryKeys of the Feature being edited. In short, IUS only recognizes SW_MEMBER as the PrimaryKey.

- The number of columns allowed for the PrimaryKey is 1...n.
- Columns can be of type integer only.
- PrimaryKey is not required when using addFeature or replaceFeature. Even if provided, the values in the PrimaryKey for the Feature are ignored and the column(s) that comprise the key identified for the table in the TableDescHelper or discovered by MapXtreme if none were identified in the TableDescHelper are ignored.

- The column SW_MEMBER is automatically set to one greater than the maximum value for that column in the table. The column(s) that comprise the PrimaryKey, if different from SW_MEMBER are not set at all.
- The PrimaryKey is not returned from addFeature as it may not refer to the same column(s) as the input PrimaryKey.

Transformation of Coordinate System for JDBC Layer Features

It is not necessary that the coordinate system of the Feature be the same as that of the JDBC database table where it is to be added. MapXtreme expects the Feature to be in the MapJ numeric coordinate system and will handle all required transformations.

Saving Rendition for JDBC Layer Features

MapXtreme can only save the Feature rendition if the rendition column in the database table has been previously identified. This must be done in the TableDescHelper used to create the layer.

In addition, because MapXtreme renditions contain constructs that cannot be represented in the format RenditionType.mapbasic, MapXtreme can only save renditions in the format RenditionType.mapxtreme. It can, however, read renditions in the format RenditionType.mapbasic. If the table rendition column is in format RenditionType.mapbasic, the Feature rendition is not saved. For addFeature, this means the column will be null. For replaceFeature, the column will retain its previous value.