

# **RCAGIS Technical Documentation**

Last Modified: 4/13/00

This document is designed to provide some basic information on how RCAGIS fundamentally works, and what some of the important things to look at are.

<b>CORE VISUAL BASIC PROJECTS FOR RCAGIS.....</b>	<b>2</b>
<b>DESCRIPTION OF THE RCAGIS CONFIGURATION DATABASE .....</b>	<b>3</b>
<b>HOW RCAGIS BUILDS AND EXECUTES AN INCIDENT QUERY .....</b>	<b>7</b>
<b>HOW RCAGIS HANDLES THEMES, PROJECTIONS, AND IMAGES.....</b>	<b>9</b>

## **Core Visual Basic Projects for RCAGIS**

The RCAGIS source code consists of one main project (rcagis.vbp) and several component projects, as listed below:

### **Core RCAGIS applications**

\mo_rcagis\rcagis.vbp	main rcagis app
\configuration\config.vbp	Administrator app
\createLegends\createLegends.vbp	Symbolizer app

### **Standalone application for importing FoxPro Databases into MS Access**

\Import\RCAGIS_Importer.vbp	RCAGIS Importer app
-----------------------------	---------------------

### **Controls used various places within RCAGIS**

\aeComponents\legendSR\MO2legend.vbp	legendSR ActiveX control
\crmgisMultiOption\crmgisMultiOption.vbp	MultiCheckOpt ActiveX Control
\VBUSplitter\VBUSplitterControl2.vbp	vbSplitter ActiveX Control

## **Description of the RCAGIS Configuration Database**

The RCAGIS Configuration Database is an MS Access database used by RCAGIS that stores information for practically everything RCAGIS does. Included in this database are SQL query definitions for crime types, default query wizard settings, additional detail information for lookup tables used by the query wizard, and other jurisdiction-specific information.

The Configuration Database is typically edited with the Administrator application designed for that purpose (config.vbp)

Note that there are a number of tables whose names are prefixed with “scas.” These table names are a legacy from the days when RCAGIS was going to be version 2 of the Spatial Crime Analysis System developed by the same team.

### **Lookup Tables used for wizard/query/filter screens**

These lookup tables are duplicated from the RCAS structure to allow more detailed captions for display in the wizards and to allow such things as reassignment of sort order. It also ensures a particular structure when we need to create query forms for tables which are truth tables in the RCAS structure. For instance, the MO table is a large collection of boolean fields the names of which represent the different MOs. Each incident, then, has a large series of True/False entries which describe the MO of the incident.

These tables are editable via the Edit Attributes menu option in the RCAGIS Administrator

*ageCode*  
*areaCode*  
*ArrestHistorySex*  
*AttributeTables*  
*bodyBuildCode*  
*complexionCode*  
*countyCode*  
*ethnicCode*  
*eyeCode*  
*glassesCode*  
*hairCode*  
*involvementCode*  
*locationCode*  
*markCode*  
*markLocationCode*  
*miscCode*  
*numWound*  
*paroleEye*

*paroleHair*  
*paroleOffenses*  
*paroleRace*  
*paroleReasmve*  
*paroleReleasedFrom*  
*paroleSupvcnty*  
*printCode*  
*scasMO*  
*scasMOA*  
*scasPOA*  
*scasProperty*  
*scasSuspectAge*  
*scasSuspectHeight*  
*scasSuspectRace*  
*scasSuspectSex*  
*scasSuspectWeapon*  
*scasSuspectWeaponFeature*  
*substanceInv*  
*tagCode*  
*teethCode*  
*vehColorCode*  
*vehCondition*  
*vehInvolvement*  
*vehMakeCode*  
*vehStyleCode*

*checkbox*

default values for checkboxes in the interface; here used only to specify the case status boxes should be checked by default in a query.

*fieldCaptions*

captions used in Administrator for prompting user to enter data source fields.

*fieldmap*

for each query type, a mapping of the fields RCAGIS will need to the field names used in the incident database. preset for RCAS database structure

*idFields*

small catalog of primary key fields in the three main tables of the incident database

*imgPrj*

available image catalogs and their attributes for display

## **IncidentRelated**

list of pages available for the advanced query wizard. right now this needs to be tightly integrated with the actual wizard form in the RCAGIS project, in terms of such things as step number, but it is hoped that in a future version, the wizard will be replaced with a more sensible object than the form which is currently handled by the VB Wizard Manager add-in.

### *jurisProjections*

list of jurisdictions available and their default projection ids

### *masterProjection*

default input and output projection IDs

### *paths*

list of paths used to specify file locations within the spatial data directory

### *projections*

projections which have been imported into the configuration database; stored as straight text inside a memo field, along with supplementary info (id and caption)

### *querymap*

list of stored queries in the RCAS database to be used if attributes are going to be brought in through an AddRelate instead of being explicitly exported into the result shapefile.

### *rcagisMisc*

miscellaneous data used throughout rcagis. easily accessible via the GetRcagiisMiscData and SetRcagisMiscData procedures.

### *scasAdvancedPanels*

legacy; not currently used

### *scasBaseQuery*

list of base queries used to generate queries to the incident database; editable via the Edit Base Queries option in RCAGIS Administrator

### *scascrim*

crime types and the SQL WHERE clauses necessary to query for them

### *scasCType*

legacy; not currently used

### *scasDateFormat*

format string used to query for date in a given database back end type; included for variant SQL dialects.

*scasga*

list of geographic areas available. includes jurisdiction id and display order (size attribute) as well as display caption

*scasgac*

available display captions for geographic areas

*scasgas*

table linking shift information from scasshif to geographic area info in scasga; contains paths for geographic area shapefiles

*scasjuri*

jurisdiction information (location, caption, rcas id)

*scasshif*

shift information by jurisdiction (caption, start/end time, etc.)

*scastj*

shapefiles used for themes, arranged by theme type and jurisdiction

*scastype*

available theme types

*tabArrestHistory*

list of attribute tables which will be used as tabs in the Arrest History query form

*tabParole*

list of attribute tables which will be used as tabs in the Parole query form

*tabVehicles*

list of attribute tables which will be used as tabs in the Vehicle query form

## How RCAGIS builds and executes an incident Query

RCAGIS was designed in such a way that one could basically remove the query wizard and replace it with something else and still end up with a working package.

The Query wizard basically collects a bunch of information, including the SQL string which will return a recordset to be exported into a shapefile, from the user and places it into a few invisible text fields on the main kiosk form. The frmMOVView form then knows how to take that information and make a map.

When you first run RCAGIS and go into the query wizard, the basic information regarding the query is collected and saved out into text fields on frmRcagisMain as follows. Some of this data is currently redundant and will probably have been cleaned up in a future release.

.txtAttrib      geographic area name

.txtAVSend

- 1)      jurisdictionID
- 2)      jurisdiction path
- 3)      jurisdiction shapefile
- 4)      geo area shapefile
- 5)      geo area field
- 6)      geo area value (in field)
- 7)      caption of geo area type
- 8)      query
- 9)      True (not currently used)
- 10)     False (not currently used)

.txtGApath      path to geo area shapefile

.txtDataPath    full path to geo area shapefile (yes, it's redundant)

.txtJuris        id number of jurisdiction (also redundant)

.txtQuery        SQL query string (also redundant)

.txtRecoveryQuery    SQL string for recoveries for current incident query (optional)

.txtSuspectQuery     SQL string for arrestees for current incident query (optional)

.txtTable        base name of shapefile (table containing attribute data)

To actually carry out the query, the boolean attribute DoQuery of frmMOVView is set to True and then frmMOVView is activated with a Show request. The Form\_Activate event then fires the various procedures which perform the query and set up the map display as necessary.

There are two main procedures behind this. One is MakeResultFromQuery, which handles the data aspects of the new query (running the query over ODBC, exporting the results to a shapefile, adding that shapefile as a layer to the map, and symbolizing it) and GetZoomExtent which handles the spatial aspects of the query, in particular figuring out what the visible extent should be and querying the report database as to whether there is a Crime Alert which should be displayed.

## How RCAGIS Handles Themes, Projections, and Images

One of the major design goals for RCAGIS was to be able to display thematic data originating from different sources within the group of jurisdictions that would be using the package in a way that made sense and would hopefully reduce confusion for users. For instance, suppose two jurisdictions each wanted to include a point shapefile containing school locations for their users. These shapefiles might have different fields, use different codes to represent different types of schools, and be stored in different projections. These parameters lead us to a list of ways to reduce cognitive load on the user:

- Provide a method of treating all shapefiles which comprise a group as a single entity. This would:
  - Load all necessary shapefiles that comprise a group such as “Schools.”
  - Present all shapefiles as a single theme to the user
  - Allow the user to operate on those shapefiles as if it were just one – changing sort order or visibility.
- Provide administrative types to persistently symbolize those shapefiles so that the user could see a consistent set of symbology across jurisdictions.
- Provide a method to make sure that the different shapefiles would line up with each other despite being in different projections. Further the projections would need to be changeable so that raster data stored in different projections would line up with vector data as necessary.

The implementation of this wish list was largely carried out in two ways. The first was to use the concept of a theme – a group of layers representing similar features – as the foundation; a projection dictionary to store information about the projections used within the overall installation; and an implementation of an image catalog system to keep track of what imagery is to be displayed given a map extent. All of this was done through a number of different Visual Basic code modules, Visual Basic class modules, and ActiveX controls.

```
\miscCode\legendPersistence.bas  
\miscCode\scaleRenderer.cls  
\miscCode\scaleRenderers.cls  
\aeComponents\legendSR\legendSR.vbp  
\mo_rcagis\frmLayerPropAE.frm, .frx  
\miscCode\MapTip.cls
```

The first step of implementing themes was to create a way to load and save legends for individual shapefiles. These legends are stored as plain text in files with an *.rvl* extension, typically using the same prefix as the shapefile they are to accompany. Because it was

desirable to be able to symbolize a shapefile differently at different scales (for instance, providing less detail on a road layer at small scales), the legend files were designed so that they could store multiple legends, each with a specified minimum and maximum display scale.

MapObjects uses renderer objects to symbolize map layers. Each renderer has a set of attributes, some of which apply to all renderers (e.g. color) while others apply only to particular type of renderer (e.g. DotValue). Legend files may store any of the types of (non-custom) renderers included in MapObjects 1.2, as well as the LabelPlacer object, and may store different types of renderers for use at different scales. Legend files can also store alternate captions to use for display in an ArcExplorer-styled legend. For instance, if an attribute in the database reads "W", one might wish to assign the word "Water" for a more meaningful display. Finally, legend files can also store an arbitrary string which allows specification of a theme type; this is what allows shapefiles to be grouped into themes as described above.

In order to actually handle the multiple renderers that might be used with a single layer in a map, the ScaleRenderer object was created. The ScaleRenderer is basically a collection of renderers, along with details about the scales at which each is to be used, and containing information about the layer being symbolized – to which jurisdiction, if any, it belongs; the type of layer (streets, schools, land use, etc.), and whether or not that jurisdiction is currently visible on the map display.

Another object, the ScaleRenderers collection, was also written to handle the overhead of keeping track of the multiple ScaleRenderer objects that would be needed for a map. The ScaleRenderers collection holds a ScaleRenderer for each layer loaded into a map. For any given layer, it can return the correct renderer to use for the current display scale. It can also return subsets, such as a reference to all of the loaded layers of a given type. In order to be able to identify which ScaleRenderer in the collection is to be used for a given layer in a map, the layer's tag is specified as the full path and filename of the shapefile. This same string is then used as the key for that layer's ScaleRenderer within the ScaleRenderers Collection.

To simplify use of the ScaleRenderer and its collection, a method of loading a shapefile with its attendant legend file into a map layer was written. The programmer simply passes the ScaleRenderers collection, along with a few other necessary things such as the map layer (to ensure its tag is correct) and the shapefile's path and filename, and a new ScaleRenderer is added to the collection. The full path of the shapefile drawn within a layer is used as a unique key for that layer's ScaleRenderer in the ScaleRenderer's collection, making it possible to retrieve

```
dim myLayer as MapObjects2.MapLayer
Set myLayer = loadSymbolizedShapefile2(myDataConnection,
    shapefilePathName, myScaleRenderersCollection,
    [defaultLegendPath], [defaultLegendFile.rvl],
    [alternateLayerName])
```

Using the ScaleRenderers collection is fairly simple. In a map's BeforeLayerDraw event, one cycles through the layers, using the layer tag as a means to get to the ScaleRenderer containing symbology information for the layer and thereby retrieve the correct renderer for the current scale.

In order to create renderers, a single GUI is provided in frmLayerPropAE.frm. This VB form, a rewrite of frmLayerProp which shipped as part of the MOVView demo in the MapObjects 1.2 distribution, provides an interface not only to the different types of renderers available, but also to the enhanced functionality provided by the ScaleRenderer, including the assignment of alternate labels for attributes. Other enhancements to the Layer Properties editor include the ability to assign custom class breaks, add and remove values from the list of attribute values to be displayed, and setting of layer type for inclusion in a theme. Additionally, a number of bugs, particularly in the creation of class breaks, were fixed.

Because one of the goals of the theme architecture was to cut down on clutter in a legend display, a fairly extensive rewrite was performed on ESRI's ArcExplorer legend ActiveX control, the source code to which was included in the MapObjects 2 distribution. The ArcExplorer legend is a fairly simple control with many useful features. When associated with a map control, it will display a list of the layers loaded into that map. Map Layers include a small set of symbols representing the settings of the renderer or symbol used to draw the layer. An optional checkbox gives the user control over whether a layer is visible. Users may reorder the layers of a map by dragging and dropping the items listed within the legend. A single layer may be specified as "active" by clicking on it; the legend can then be queried for the active layer so that operations can be carried out on the specified layer.

The new version, legendSR, is backwardly compatible with the old legend, but now contains a property for storing a reference to a ScaleRenderers collection. If this property has been set, the legend will ensure that its display reflects what is currently displayed in the map window based on scale. Further, to help in situations in which two jurisdictions might use different shapefiles with varying legends for the same theme, the legend will display the symbology for the shapefile which is currently in the center of the map display. LegendSR also has options for whether or not to display images in the legend and whether images should be always kept as the bottom layer of a map or can have their draw order changed by the user.

If a ScaleRenderer contains alternate captions for the values a renderer will display, the legendSR will use the alternate captions. Theme names are used to identify the item in the legend, rather than the name of the individual layer. And of course, the legendSR ensures that when themes in the legend are dragged up or down, all of the layers with that type will move up or down in the map draw order so that they remain together.

Further support of the Themes architecture was written into a new version of ESRI's MapTip class, also distributed as part of the MOVView sample in MapObjects 1.2. MapTips allow the user to specify an attribute field of interest; hovering over a feature on

the map will pop up a small label showing the value of the attribute field for that record. The new version of the MapTip object is capable of displaying MapTips for multiple shapefiles which have been grouped into a theme. The only limitation to this is that the user may only specify one attribute name on which to search. However, if the shapefiles share field names, such as "Name," this will not cause problems. But if a Streets theme has two shapefiles, one where street name is stored as the attribute "street" and another where it is called "streetname," the user must select the correct attribute for the jurisdiction in which they are showing tips.

### **\miscCode\ProjectionDictionary.cls**

One of the requirements for RCAGIS was that the software be able to display image layers – or more specifically, to be able to take advantage of the aerial digital orthophotos and other raster data available to some members of the consortium.

MapObjects image layers can not be projected. In order to display shapefiles on top of a georeferenced image, one needs to specify the coordinate system of the map display as well as the coordinate system of the shapefile, if it differs.

Ideally, all shapefiles used in RCAGIS could be converted to a common projection, which would obviate the need to project shapefiles for display. However, unless any image layers, such as aerial photos, that participatory jurisdictions wished to use were all in the same projection, it would still be necessary to specify input and output projections for the displayed shapefiles and map control, respectively.

One of the new features of MapObjects 2 is the ability to handle multiple projected and unprojected coordinate systems. By breaking it down such that a map object has an display projection and shapefiles have, in effect, input projections, it becomes possible to display shapefiles with two different projections in a third projection. RCAGIS takes advantage of these projection capabilities, allowing jurisdictions to maintain their data in a locally prescribed projection, yet still view their data alongside that from other jurisdictions or sources, while allowing these data to be viewed over an image layer with an unchangeable projection.

In MapObjects 2, an optional projection file, with the extension *.prj*, would be stored with each shapefile. When loading a ShapeFile into a layer via the `DataConnection.FindGeoDataset` method, MO2 assumes that a projection file, if it exists, should be loaded and assigned to the layer. Based on our testing, it seems that after this, no sort of comparison is then run to see if the input projection of the shapefile is the same as the output projection of the map on which that layer is being drawn. As a result, MO2 first unprojects the points in the file and then reprojects them to the output projection. This can result in much slower feature drawing than if the layer's `CoordinateSystem` property is set to nothing when the underlying data for the layer is in the same projection as the output projection.

To ensure that ShapeFiles are only projected when necessary, RCAGIS stores projections in its internal configuration database, and specifies the projection to be used for each given shapefile. The primary advantage to this is that projections are only assigned to input layers if their projection is different from the output projection. In other words, if a shapefile has been stored as UTM coordinates and the map will be projected to UTM, the layer's CoordinateSystem property is set to nothing. This yields a noticeable improvement in map display speed.

In order to handle the projections that would be necessary within RCAGIS, an object called the ProjectionDictionary was created.

Rather than force RCAGIS administrators to define a projection for every shapefile utilized within a jurisdiction, defaults can be assigned to jurisdictions. For instance, if the bulk of the data used by a jurisdiction is in Maryland State Plane Feet, then that can be assigned as the default projection for the jurisdiction and all shapefiles associated with that jurisdiction will be assumed to be stored in that coordinate system unless otherwise specified. Individual projections may be specified for themes and geographic areas.

The projections themselves are stored within the configuration database, rather than as individual .prj files. The ProjectionDictionary object automates the tasks of importing projection files into the database and retrieving them from the database as coordinate system objects. Internally, the database stores the contents of projection files in a memo field. When a request is made to an instance of a ProjectionDictionary to retrieve a projection which has not yet been stored as a coordinate system object within the instance, the dictionary will write out a temporary projection file containing the text describing the projection; use MapObjects methods to open that file as a coordinate system, and then remove the scratch file. Subsequent requests for that same projection will be able to use a reference to the now-loaded object.

In many ways, the ProjectionDictionary simply acts as an interface to data stored within the configuration database. It provides a single object that allows storing and retrieving projections, conversion of Arc/Info projections to MapObjects projections, and exportation of shapefiles into new projections.

### **`\miscCode\clsRcagisImage.cls`**

The handling of raster data in RCAGIS is done with a custom object which both creates image catalogs (shapefiles containing polygons demarcating the extent of each image file and the paths to those images) as well as the methods for retrieving single or multiple image filenames so that MO2 image layers can be created and added to a map. All image files for a single catalog (e.g. a set of aerial photos for a jurisdiction) must be contained in a single directory. Each catalog must be in its own directory.

The two most important requests for the RcagisImage object are:

```
CreateIndexFile(inPath)
```

where `inPath` is the full path to the directory containing the images to be indexed

```
GetImages(ByRef inShape As Object, _  
    Optional jurisID As Variant, _  
    Optional sPath As Variant) As Variant
```

where `inShape` is a `MapObjects2 Point` or `Rectangle`. Either `JurisID` or `sPath` must be provided. If `JurisID` is given, the shape will be intersected with the current library for that jurisdiction and a variant array of images representing that intersection will be returned. If `sPath` is provided, the shape will be intersected with the image library in that path, if one exists. If no images are found, a 0 is returned.

## Self-Adjusting Interfaces

One of the keys to the data-driven design of RCAGIS is the ability of the wizard interface to update itself given changes in the configuration database. Most of this is made possible by the `crmgisMultiChkOpt` ActiveX control. This control, which can be dropped on a form, organizes control arrays of checkboxes and option buttons and provides horizontal and vertical sliders, a toggle switch for checkbox arrays, and several methods for getting and setting the status of the items in the control array. Examples of its use can be found in `frmWizard`'s "LoadX" procedures, amongst other places.

**`\crmgisMultiOption\crmgisMultiOption.vbp`**