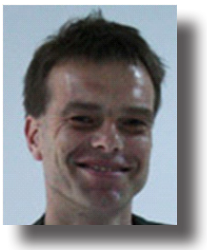
Integrating TatukGIS and PowerBuilder

Juergen Seelmann, Dr. Dirk Lorenzen and Joash Bolanio provide a detailed account of the TatukGIS and PowerBuilder integration process for a local revenue computing and accounting system called iTAX (integrated Tax Administration). The iTAX system stores, using a relational databank, and graphically displays all relevant (credit/ debit) data of taxpayers, and thus helps local government units monitor and control all tax transactions by its citizens and tax payers.

By Juergen Seelmann, Dr. Dirk Lorenzen and Joash Bolanio

Juergen Seelmann is senior technology adviser for GTZ. Juergen's main focus is on the development of Tax applications using Powerbuilder. He has 15 years of expertise with PB development using various databases such as Oracle, Sybase ASE, INGRES and Firebird. He can be reached at juergenseelmann@web.de.



Dr. Dirk Lorenzen is as a scientist for various German universities. Dirk's expertise is on a in-depth knowledge of the GIS tool TatukGIS. He can be reached at Dlorenzen@ecology.uni-kiel.de.

Joash Bolanio is senior developer for GFA. Joash is focused on Powerbuilder Development. He has 5 years of expertise in developing Tax Administration systems in the Philippines and in Tanzania. He can be reached at jb_bulls@yahoo.com.

The iTAX software was developed using PowerBuilder 10.5 and an Ingres database for data storage. The software for the development of GIS (Geographic Information System) functionalities within iTAX was the TatukGIS developer Kernel (ActiveX version). The major features of the system were:

- **Finding a lot by selecting the PIN_ID in the iTAX database,**
- **Zooming into the lot,**
- **Viewing the stored information of the selected and adjacent lots,**
- **Subdividing a lot and assigning the new PIN_IDs to the new polygons,**
- **Combining several lots, and**
- **Creating thematic maps.**

The TatukGIS Developer Toolkit

The TatukGIS Developer Kernel (DK) is a comprehensive GIS software development toolkit (SDK) for the development of custom stand-alone and embeddable applications using object-oriented languages. The DK toolkit is provided as four separate editions organized by development platform. The editionas are as follows:

1. **DK-VCL** : A native Delphi/C++ Builder VCL (visual component library) for use with CodeGear Delphi and C++Builder development environments.
2. **DK.NET** : A native .NET WinForms component built on managed code for use with C#, VB.NET and Microsoft Visual Studio. NET assemblies at 2.0 and higher.
3. **DK-ActiveX** : An ActiveX (OCX) control and supporting .NET via COM/Interop for use with Visual Basic, VB.NET, C#, Visual C++ and Microsoft Visual Studio.
4. **DK-CF** : A native .NET Compact Framework component for Microsoft Visual Studio and Windows Pocket PC, Windows Mobile and Windows CE development.

Each DK edition reflects the same, or very similar API, functionality and property sets. The code of an application developed with the DK.NET and DK-CF is exactly the same, so an application developed with the DK.NET can be recompiled with the DK-CF. The close similarity of the four DK editions makes porting a DK-based application between development environments, (e.g. Delphi, C++Builder, Visual Basic, Visual C++, C#, .NET, Compact Framework, etc.), a relatively straight forward task. In addition to most raster and shapefile data formats, the DK supports industry standard SQL database map layer formats. (A shapefile is a common digital map format that stores the topology and attribute

information - it consists at least three main files *.shp, *.shx and *.dbf file.)

DK based desktop applications can be deployed to final users free of additional costs or royalties. The deployment of DK based server applications requires the payment of a DK Server Deployment Royalty. Each DK license purchase includes a license of the TatukGIS Internet Server Developer Edition to enable DK licensed developers to work also with the Internet Server product.

Implementation Of The "Basic" GIS Functionalities

The functionalities are implemented into PowerBuilder applications by ActiveX connections. TatukGIS internally uses a number of named constants. The first step to declare the variables declared as follows:

```
------------------------------------------------------------
Listing 1: Declaration of Variables
------------------------------------------------------------
//TatukGIS_DK.XGIS_ViewerMode.XgisDrag
CONSTANT INT gisSELECT                    = 0
CONSTANT INT gisDRAG                      = 1
CONSTANT INT gisZOOM                      = 2
CONSTANT INT gisEDIT                      = 3
CONSTANT INT gisZOOMEX                    = 4

//TatukGIS_DK.XGIS_ShapeType.XgisShapeTypeArc
CONSTANT INT gisSHAPETYPEUNKNOWN          = 0
CONSTANT INT gisSHAPETYPEDELETED          = 1
CONSTANT INT gisSHAPETYPEPOINT            = 2
CONSTANT INT gisSHAPETYPEMULTIPOINT       = 3
CONSTANT INT gisSHAPETYPEARC              = 4
CONSTANT INT gisSHAPETYPEPOLYGON          = 5

//TatukGIS_DK.XGIS_Lock.XgisLockExtent)
//CONSTANT INT gisLOCKNONE                = 0
CONSTANT INT gisLOCKEXTENT                = 1
//CONSTANT INT gisLOCKPROJECTION          = 2
//CONSTANT INT gisLOCKINTERNAL            = 3
//CONSTANT INT gisLOCKINTERNAL2           = 4
```

Connecting To The Database

The primary aim was to connect the data of the iTAX database to the digital map (shapefile). Each lot can be identified by a unique PIN-ID. Using this field, it is possible to link/join subsets of data of the Ingres databank to polygons of the shapefile. By doing this, modifications in the databank are immediately represented in the corresponding map. Furthermore, changes in attribute data only have to be done in the database and double storage of data is avoided. The following code segment shows how this is done in the PowerBuilder application.:

```
------------------------------------------------------------
Listing 2: Joining databank data to a shapefile by using a unique id (here: "prop_id")
------------------------------------------------------------
//This example generates a result set in a ResultSet
//object from an existing DataStore object. The Re-
//sultSet object is used to populate a new ADOResult-
//Set object. The GetRecordSet function on the ADORe-
//sultSet object is used to return an instance. ADO-
//Recordset has an OLEObject that can be used with
//ADO Recordset methods.

resultset lrs_resultset
ADOresultset lrs_ADOresultset
OLEObject loo_ADOrecordset
datastore ds_source
ds_source = create datastore
ds_source.dataobject = 'd_property_data_map'
ds_source.settransobject( sqlca )
ds_source.retrieve()

//Generate a result set from an existing DataStore
ds_source.GenerateResultSet(lrs_resultset)

//Create a new ADOResultSet object and populate it
//from the generated result set
lrs_ADOresultset = CREATE ADOResultSet
lrs_ADOresultset.SetResultSet(lrs_resultset)

//Pass the data in the ADOResultSet object to an
//OLEObject that you can use as an ADORecordset
loo_ADOrecordset = CREATE OLEObject
lrs_ADOresultset.GetRecordSet(loo_ADOrecordset)
ioo_layer.JoinADO = loo_ADOrecordset
ioo_layer.JoinPrimary = "prop_id"
ioo_layer.JoinForeign = "prop_id"
```

After doing this, it is possible to set the map extent to the extent of the selected lot. Please note the ioo_layer is an instance variable for the windows.

Displaying The Attributes

Because the viewer window is connected to the attributes control, attribute information of the selected lot is shown in the attribute control window. Showing attributes of current selected shape is as follows:

```
ole_GISAtt.ShowShape(GIS.Editor.CurrentShape)
```

After selecting a shape, it should be shown centered and can be adjusted in size. The following listing illustrates the zooming feature. Please note that zooming to selected lot makes it flash 4 times.

```
------------------------------------------------------------
Listing 3: Zooming in to the lot
------------------------------------------------------------
ls_layername = 'valencia'
ls_pinno = "046-20-013-02-046"
loo_layershape = ole_gis.Object.Get( ls_layername )

//Same code with above loo_layershape = ole_gis.-
//Object Items.item[0]
ll_rowcount = loo_layershape.GetLastUid()

For ll_row = 1 to ll_rowcount
    ls_fieldpinno = loo_layershape.GetField( ll_row ,
                    'pin_no' )
    If ls_fieldpinno = ls_pinno Then
      ioo_shape = loo_layershape.GetShape( ll_row )

      If Not IsNull( ioo_shape ) Then
        ioo_shape = ioo_shape.MakeEditable()
        ole_attrib.show()
        ole_attrib.Object.ShowShape( ioo_shape )
        ole_GIS.object.VisibleExtent =
            ioo_shape.Extent
        ioo_shape.flash( )
        il_currentselected = ll_row
      End If
    End If
Next
```

We have also provided the capability for displaying the attributes of the shape using a mouse click event. To do this, we implement the select function for "mouse-down event". The code segment is as follows:

```
------------------------------------------------------------
Listing 4: Display by clicking the shape with mouse
------------------------------------------------------------
ole_gis.object.mode = 0
loo_shape = ole_gis.object.locate(
            ole_gis.object. screentomap(
            loo_utils.point( ocx_x, ocx_y ) ),
            5 / ole_gis.object.zoom )

loo_shape.flash( )
long ll_rtu
ole_attrib.show()
ole_attrib.Object.ShowShape( loo_shape )
ole_GIS.object.VisibleExtent = loo_shape.Extent
```

Another feature that was implemented provides the ability to select polygons (lots) by doing a query on the attribute data (e. g. owner name, and market value).

```
------------------------------------------------------------
Listing 5: Selecting lots by doing a query
------------------------------------------------------------
Boolean wahl
String layer_name, ls_layername , ls_pinno
Integer z = 0
Long ll_row, ll_rowcount, ll_rtm
long ll_fieldmarketvalue, ll_marketvalue
OleObject loo_layershape
loo_layershape = Create OleObject
ll_rtm = loo_layershape.ConnectToNewObject(
          "TatukGIS_DK.XGIS_LayerSHP")

If ll_rtm <> 0 Then
    Messagebox("Connect...to: XGIS_LayerSHP",
                ll_rtm)

ole_gis.object.mode = gisSELECT
ll_rowcount = ole_gis.object.Items.Count - 1
ls_layername = 'valencia'
ll_marketvalue = long(sle_1.text)
loo_layershape = ole_gis.Object.Get( ls_layername )

//Reset MAP to original state.
loo_layershape.RevertAll()
ll_rowcount = loo_layershape.GetLastUid()

For ll_row = 1 to ll_rowcount
    ll_fieldmarketvalue = long(
    loo_layershape.GetField(ll_row , 'marketvalue' ) )

    If ll_fieldmarketvalue > ll_marketvalue Then
      ioo_shape = loo_layershape.GetShape(
                  ll_row)
      If Not IsNull( ioo_shape ) Then
        ioo_shape = ioo_shape.MakeEditable()
        ioo_shape.Params.Area.Color =
            RGB( 255,0,0) * 256 + Rand(256)
           //Rand(255) * 256 * 256 + Rand(255)
           //65536 * Blue+ 256 * Green+ Red
      End If
    End If
End For
ole_gis.object.update()
```

The result of the above query is illustrated in **Figure 1**. The selected lots (polygons) are colored in red.

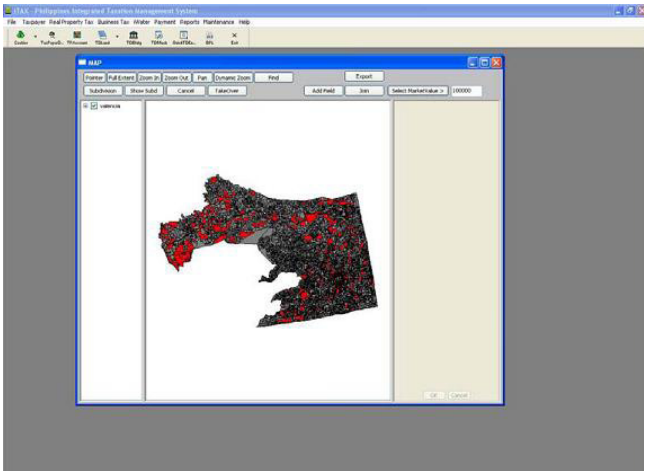


Figure 1: Result of the Query

Subdivision And Consolidation

Properties can get subdivided or properties get consolidated by combining several lots into one lot. This can be done by using the topology functions of TatukGIS. The following example shows how the subdividing function is implemented in the iTAX application:

```
------------------------------------------------------------
Listing 5: Subdividing a lot using the topology function of TatukGIS. Events/Functions used: OnMouseDown, ue_showsubdivide and ue_save
------------------------------------------------------------
Subdivide: OnMouseDown Event:
   Case 'subdvision'
         ioo_ptg = Ole_GIS.Object.ScreenToMap(
                   ioo_utils.point( ocx_x, ocx_y ) )
          ioo_shpArc.Lock( gisLOCKEXTENT )
          ioo_shpArc.AddPoint( ioo_ptg )
          ioo_shpArc.Unlock()
          Ole_GIS.Object.Update()

Show Subdivision: ue_ShowSubdivide
event ue_showsubdivide();
long ll_row , ll_rowcount, ll_color
string ls_modify
ioo_layerObj.RevertAll()
ioo_shape_list = ioo_topology_obj.SplitByArc(
                    ioo_shpPolygon, ioo_shpArc, True)

If Not isNull( ioo_shape_list ) Then
   ll_rowcount = ioo_shape_list.Count - 1
   For ll_row = 0 To ll_rowcount
       ll_color = Rand(255) * 256 * 256 + Rand(255)
                  * 256 + Rand(256)
       //65536 * Blue+ 256 * Green+ Red
       ioo_shape_list.Item(ll_row).Params.Area.Color
                = ll_color
       ioo_shape_list.Item(ll_row).Params.Labels.
       Value = string( ll_row + 1 )
       ioo_shape_list.Item(ll_row).Params.Labels.
       Font.Size = 45
       ioo_layerObj.AddShape(
                ioo_shape_list.Item(ll_row))
       dw_detail.setItem( ll_row + 1 , 'taxtrans_id',
                ll_row + 1 )
       ls_modify += " if( taxtrans_id =" + string(
       ll_row + 1 ) + " , " + string( ll_color ) + " , "
   Next
       ls_modify += " rgb(255,255,255) " + fill( ')',
                ll_rowcount + 1 )
       dw_detail.Modify("pinno.Background.
                Color='0~t " + ls_modify + " ' ")
       dw_detail.Modify("taxtrans_id.Background.
       Color='0~t " + ls_modify + " ' ")
       dw_detail.Modify("tdno.Background.Color=
                '0~t " + ls_modify + " ' ")
       dw_detail.Modify("owner_name.Background.
                Color='0~t " + ls_modify + " ' ")
End If

//topology_obj = Nothing
//GIS.CtlUpdate()

Ole_GIS.Object.Update()
cb_save.Enabled = True
end event
Save Subdivision: ue_save
event ue_save();
long ll_rtu, ll_row, ll_rowcount, ll_rtm,
     ll_rowcountshp, ll_rowshp
string ls_layername1
OleObject  loo_lo, loo_shp, loo_shppoly,
           loo_layershape
ls_layername1 = 'valencia'
loo_shp = Create OleObject
ll_rtu = loo_shp.ConnectToNewObject("TatukGIS_
DK.XGIS_ShapePolygon")

If ll_rtu <> 0 Then
   Messagebox("Connect... to: loo_shp", ll_rtu)

loo_shpPoly = Create OleObject
ll_rtu = loo_shpPoly.ConnectToNewObject(
        "TatukGIS_DK.XGIS_ShapePolygon")
```

```
If ll_rtu <> 0 Then
    Messagebox("Connect... to: loo_shpPoly", ll_rtu)

loo_layershape = Create OleObject
ll_rtm = loo_layershape.
ConnectToNewObject("TatukGIS_DK.XGIS_LayerSHP")

If ll_rtm <> 0 Then
   Messagebox("Connect... to: XGIS_LayerSHP", ll_rtm)

//loo_shpPoly.Name = "poly"
//ioo_layerObj.Name = "Splits"
loo_shpPoly = ole_GIS.Object.Items.Item(1).Create-
Shape( gisSHAPETYPEPOLYGON )
loo_shpPoly.AddPart()
loo_layershape = ole_gis.Object.Get( ls_layername1 )
loo_shp = loo_layershape.GetShape(
            il_currentselected ) // ll_row )
loo_shp = loo_shp.MakeEditable

If ioo_shpArc.Intersect(loo_shp) And
   loo_shp.Uid = il_currentselected Then
   loo_shp.Delete()
   //GIS.Refresh()
   //setsort
   dw_detail.setsort( 'taxtrans_id ASC' )
   dw_detail.sort( )
   ll_rowcountshp =  ioo_shape_list.count - 1

   For ll_rowshp = 0 To ll_rowcountshp
       //TODO: save the New PIN..include prop id.
       ioo_shape_list.Item( ll_rowshp ).SetField("pin",
              dw_detail.getItemString( ll_rowshp + 1,
              'pinno' ) )
       ioo_shape_list.Item( ll_rowshp ).SetField(
             "prop_id", dw_detail.getItemString(
              ll_rowshp + 1, 'prop_id' ) )
       loo_layershape.AddShape( ioo_shape_list.Item(
              ll_rowshp ) )
       //loo_layershape.object.SetField( "pin_no"
    Next
    Ole_GIS.Object.Update()
End If

loo_layershape.SaveAll()
ole_GIS.Object.Delete("Splits")
ole_GIS.Object.Delete("Arc")
ole_GIS.Object.Update()
is_flag = 'point'
end event
```

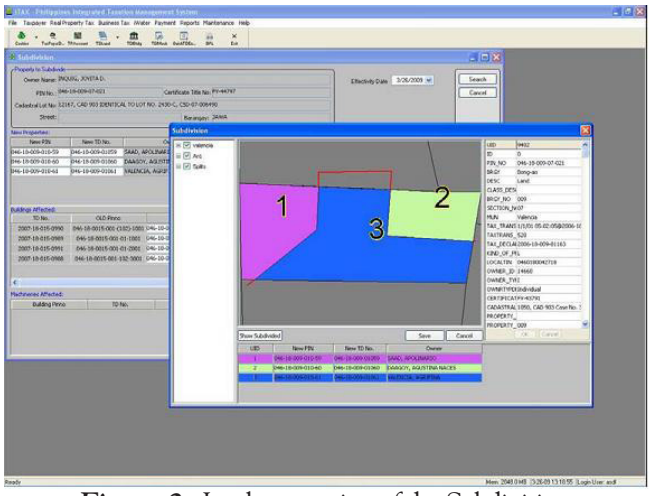


Figure 2: Implementation of the Subdivision

Automatic Creation Of Thematic Maps

The TatukGIS developer kernel provides all functionalities to produce layouts of maps by programming all needed steps (including legend, north arrow, scale bar, text, etc.). So the idea is to implement predefined or very easy to use queries on the data of the databank, join the results to the attribute table of the shapefiles and layout the whole map just "on mouseclick". **Figure 3** shown below illustrates the schematic concept of this idea:

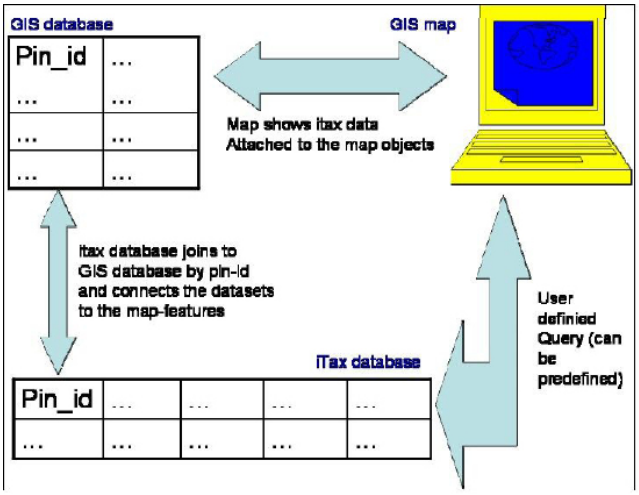


Figure 3: Creating thematic maps automatically by using query functions and joining the results to the shapefiles

Besides the time saving effect, the automation process will minimize failures and always ensure that the most current data is used for map production. Another advantage of this approach is that the layout of the final maps can be defined by the programmers so that all "result-maps" have a unique design and can be easily identified a "iTAX-maps" by the contemplators. ■