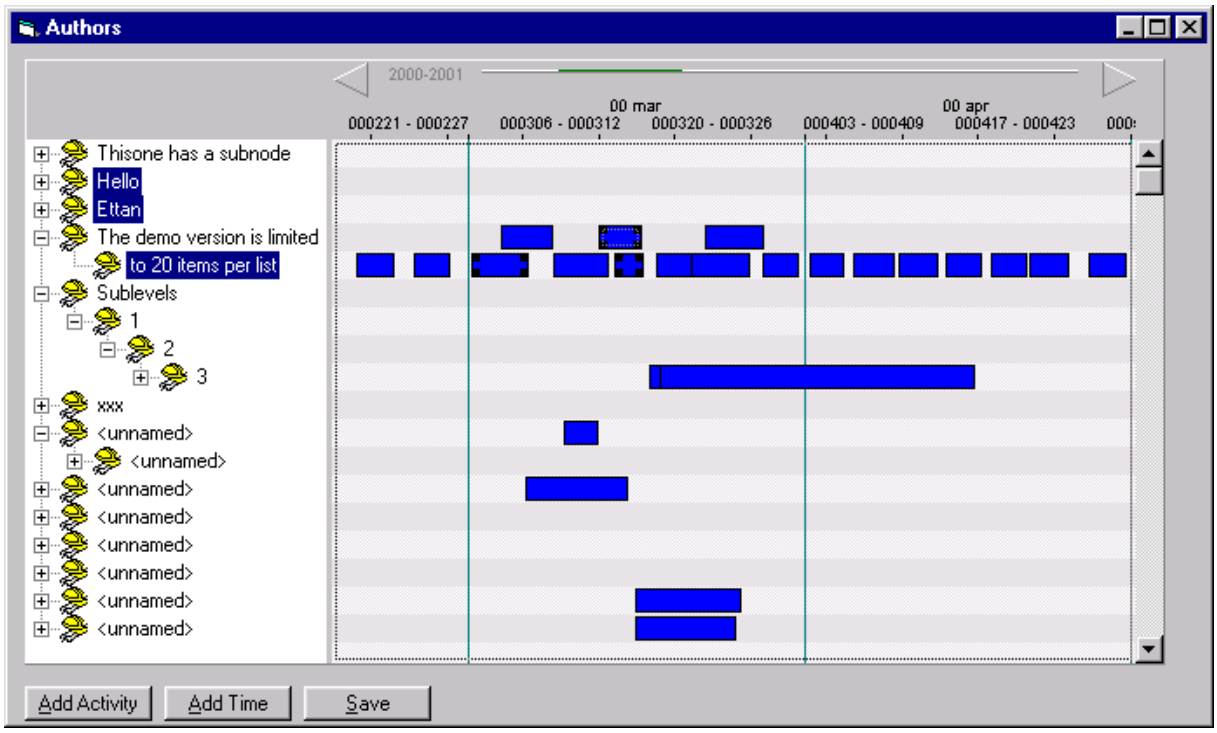
PhGantt

Introduction

This document is a hands on description of many of the current features of the phGantt component. The document is written for an intermediate or advanced visual basic programmer. The phGantt component is under constant development, do not hesitate to submit your thoughts about functionality to info@plexityhide.com.

Figure 1



Treeview

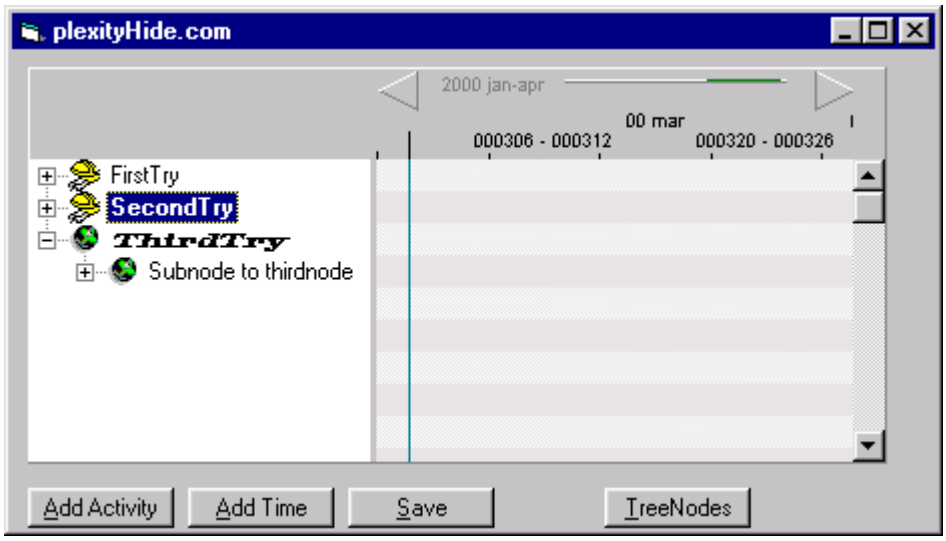
The treeview is a vital part of the phGantt component. The treeview let the programmer present stuff in a hierarchical manner, and that is often useful. Every row in the treeview constitutes a Gantt row, and consequently the Gantt rows will be visible or invisible according to the expanded state of the tree node.

Let us populate the tree:

```
Private Sub CmdTreeNodes_Click()
Dim newactivity As IphDataEntity_Tree

    Set newactivity = phGantX1.AddRootDataEntityTree
    newactivity.CanEdit = True
    newactivity.Text = "FirstTry"
    newactivity.ImageIndex = 0

    Set newactivity = phGantX1.AddRootDataEntityTree
```

```
    newactivity.CanEdit = True
    newactivity.Text = "SecondTry"
    newactivity.ImageIndex = 0
    newactivity.FontBold = True

    Set newactivity = phGantX1.AddRootDataEntityTree
    newactivity.CanEdit = True
    newactivity.Text = "ThirdTry"
    newactivity.ImageIndex = 1
    newactivity.FontItalic = True
    newactivity.FontName = "Wide Latin"

    Set newactivity = phGantX1.AddDataEntityTree(newactivity)
    newactivity.CanEdit = True
    newactivity.Text = "Subnode to thirdnode"
    newactivity.ImageIndex = 1


End Sub
```

Figure 2



You populate the tree by adding a DataEntity. DataEntity is a common object in plexityhide components; they hold the most basic information about what will be shown on the screen such as:

```
property Selected
property CanEdit
property OwningDataList
property UserReference
property Visible
property UserIntegerReference
```

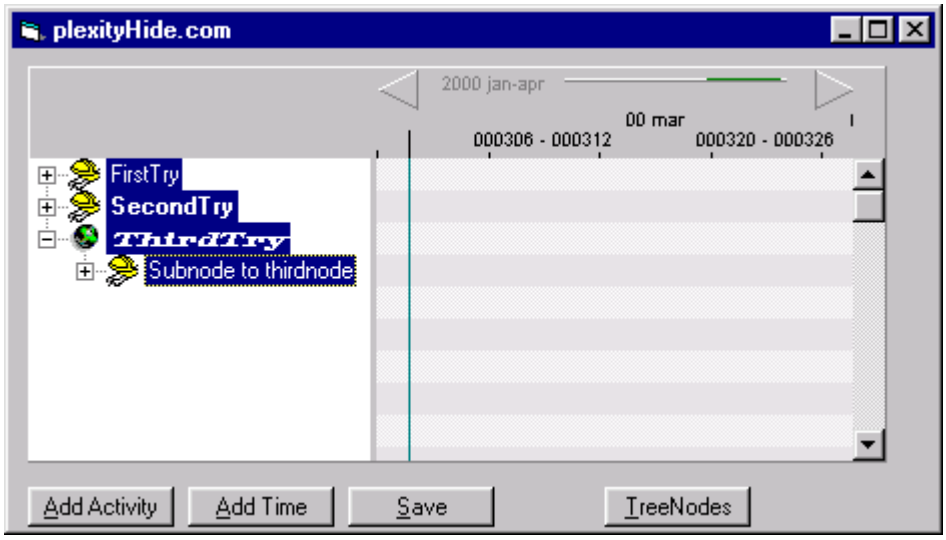
The DataEntity we use for the tree is of a special subclass with extra properties added typical for a tree node. Such as:

```
property Text
property ImageIndex
property SelectedIndex
property Expanded
property FontName
property FontBold
property FontItalic
property FontStrikeOut
property FontUnderline
```

You never instansiate the dataentities yourself, they are handed to you by the methods of the component. In the sample code above, we use two different methods because root nodes are created differently than the sub nodes. There are no other ways to create treenodes than the ones shown in the sample code above. Except from the creating methods there are no logical differences between a sub node and a root node, and there is no limit to the depth of sub nodes.

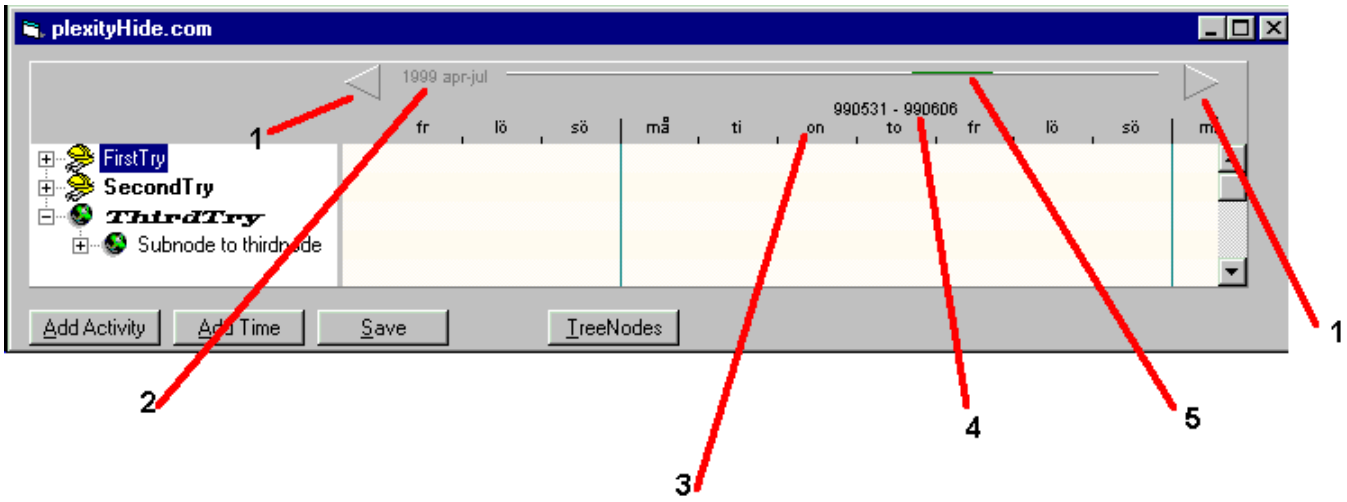
Special features

One special feature seen in the treeview which are not seen in many other components making use of the treeview is the multi select feature.



Date scale

The date scale component is of course crucial to the phGantt component, the scale should enable the user to zoom in or out and scroll back and fourth in time. The date scale is limited for practical reasons to a time span from year 1898 to year 4000, and the zooming mechanism shows time down to seconds and up to centuries.



1. These are buttons used to traverse in time, back and forth.

2. The scale presents the current span that is used in the indicator (5), the green mark in the indicator symbolizes the currently visible time window of the whole of the indicator.

3. The scale handles to kinds of presentation text, one small span text and one long span text. This is the small span text, in the sample the scale decided to use days as the small span, and consequently short names of days are presented.

4. This is the long span presentation, in the sample, it is weeks, and the weeks are presented as start of week to end of week dates.

Special features

All text that are used are taken from the national settings of the computer, in the sample above we use Swedish. First day of week is different in different countries, so the date scaler lets you specify which day that is first for you. The date and time presentations are properties of the scale and can be adjusted to your liking. The indicator functionality can be switched off.

Gantt area

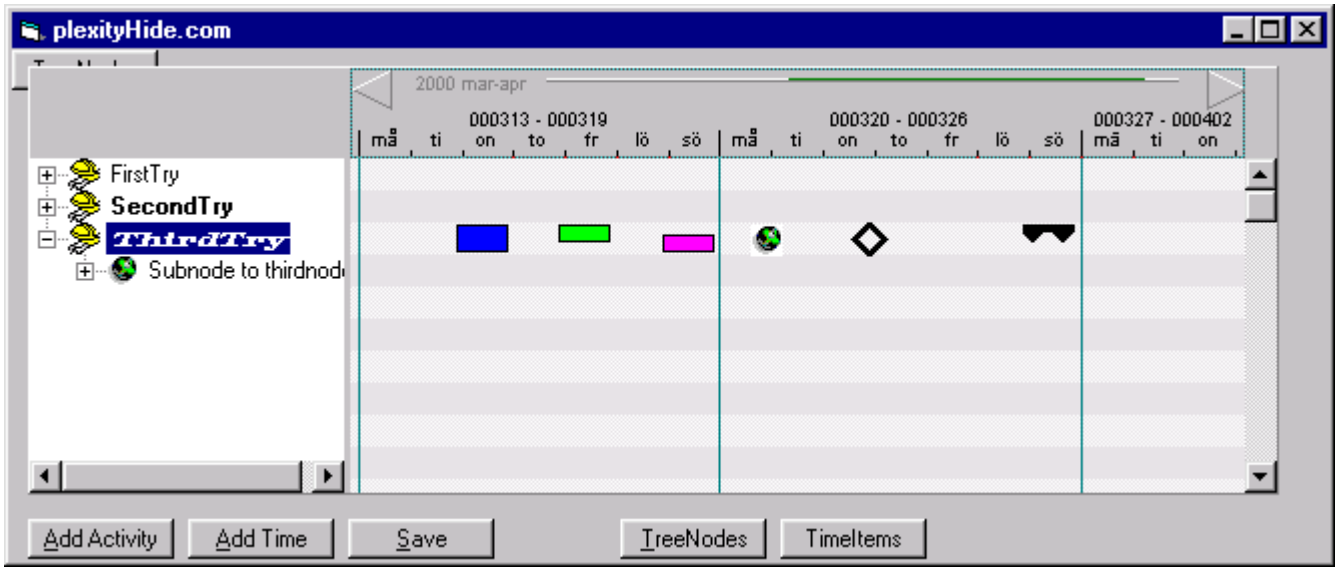
The Gantt area is divided into rows. One row corresponds to node in the treeview. The Gantt area is used to present things that has a representation in time, the time is taken from the current settings of the date scaler. Every Gantt row can have multiple layers, and the drawing order of these layers can be set. All time objects in the Gantt area are manipulated directly, the user can either move the whole object in time or resizing it by pulling the start or stop areas.

Let us populate a Gantt row:

```
Private Sub CmdTimeItems_Click()
Dim activity As IphDataEntity_Tree
Dim time As IphDataEntity_GantTime

    If Not (phGantX1.CurrentDataEntityTree Is Nothing) Then
        Set activity = phGantX1.CurrentDataEntityTree

        Set time = phGantX1.AddGantTime(activity, 0)
        time.Start = Now - 3
        time.Stop = Now - 2
        time.Color = ColorConstants.vbBlue
        time.Style = tsNormal

        Set time = phGantX1.AddGantTime(activity, 1)
        time.Start = Now - 1
        time.Stop = Now
        time.Color = ColorConstants.vbGreen
        time.Style = tsNormal
        time.BottomInset = 5

        Set time = phGantX1.AddGantTime(activity, 2)
        time.Start = Now + 1
        time.Stop = Now + 2
        time.Color = ColorConstants.vbMagenta
        time.Style = tsNormal
        time.TopInset = 5

        Set time = phGantX1.AddGantTime(activity, 0)
        time.Start = Now + 3
        time.Stop = Now + 4
        time.Style = tsImage
        time.FixedSize = 16
        time.ImageIndex = 1

        Set time = phGantX1.AddGantTime(activity, 0)
        time.Start = Now + 5
        time.Stop = Now + 6
        time.Style = tsRomb
        time.FixedSize = 16
        time.Color = ColorConstants.vbBlack

        Set time = phGantX1.AddGantTime(activity, 0)
        time.Start = Now + 8
        time.Stop = Now + 9
        time.Style = tsSpan
        time.Color = ColorConstants.vbBlack


    End If
```

The sample code adds time items to the currently active tree node. Many different styles of time drawing are available, and the use of color and inset from both the bottom and top makes the possibilities limitless. The use of fixed size allows

for time items that should not be stretched when the scale is zoomed. Items with fixed size are not resizable but still movable. There also is a drawing style that enables custom drawing.



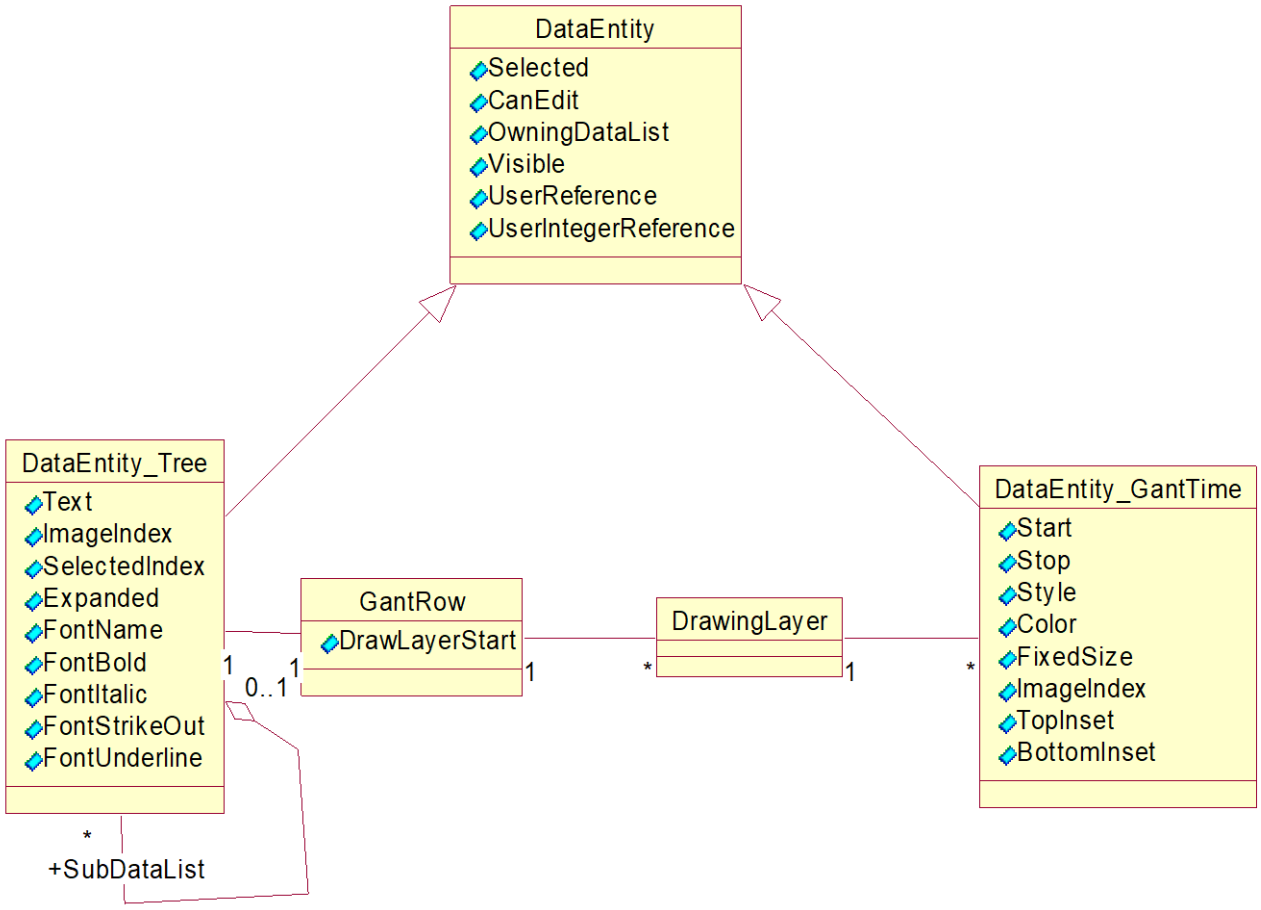
Special features

The use of drawing layers enables you to design your interface to show many different types of times on a single row and still control the appearance and drawing order. On every user interaction with a time item, an event is fired where the developer can react to the users new input. Whenever the user moves the mouse over the time items, an event is fired allowing the developer to show hints or more information about an item. Key navigation and multi select are also features of the Gantt area

The boolean property MoveInTimeWhenMoveRow lets you decide if a change of row also will be a change in time, or if the change of row operation will be time indifferent. The property MoveOver12Then24 is used if you want to make a distinction between move operations that are small (i.e. smaller than 12 hours) that should be interpreted as a change of start and stop within the same day, and large move operations that should be interpreted as a change of day but keep start and stop time the same on that new day.

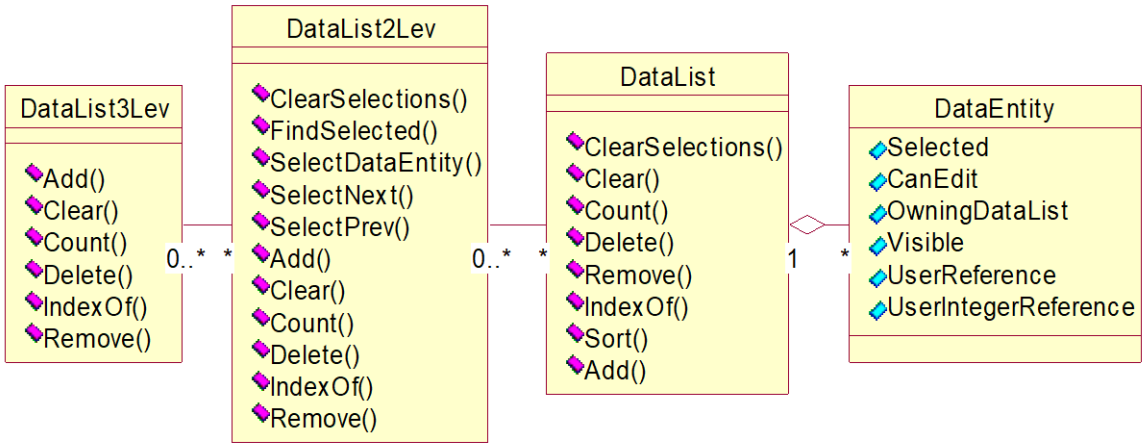
Conceptual model of dataEnteties in the phGantt

The DataEntity has two properties that are left up to you as a developer to make use of. They are UserReference and UserIntegerReference. You can assign values to them to help you identify what a particular dataentity symbolizes for you.



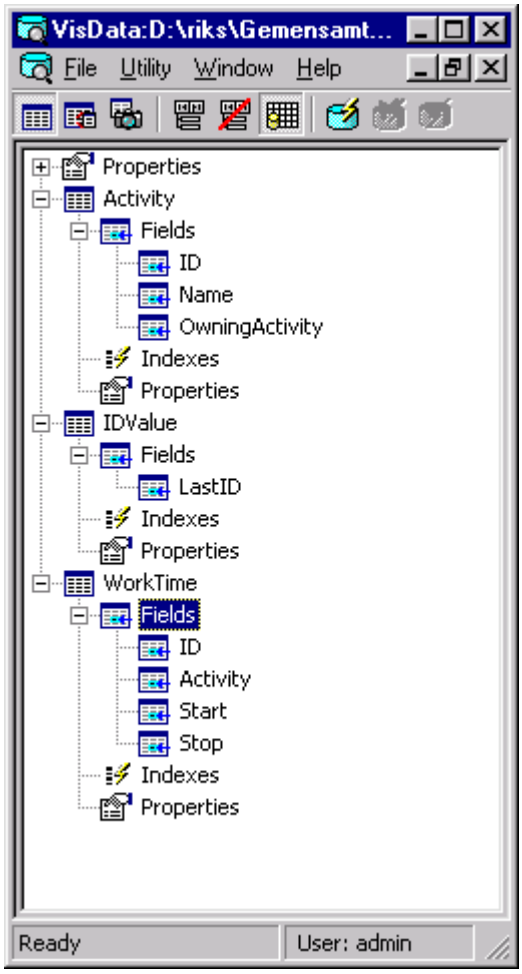
Model of DataEntity functionality

The dataEntities are managed by a datalist, all DataEntity objects are owned by exactly on DataList. There is also functionality to build collecetions of DataLists, and even collections of collections of DataLists.



Load data with ADO into the phGantt

We need an Access database for this sample:



The database has three tables.

The Activity-table is data that we will display in the treeview. It has three properties ID, Name and OwningActivity. ID is what is called the primary key; Name is the attribute we want to communicate in each treenode. And finally OwningActivity is a foreign key; we use this to point out any owning activity's primary key. If there is no owning activity we will put a null key here, the sample uses –1 as the null key.

The IDValue table is just as a persistent global variable to hold the last unique ID we used for new rows for our objects. It only has on field, LastID. LastID will be adjusted by our sample every time we want a new unique ID.

The WorkTime table has four fields. First the primary key ID. Then a foreign key Activity that points out the primary key off the activity row that this particular work item belongs to. Then we have attributes for start and stop; these are timestamp in the database and will hold the start and stop of a particular time item in the Gantt component.

```
Dim adoIDValueRS As Recordset
  Dim db As Connection

Private Sub Form_Load()
  Set db = New Connection
```

```
  db.CursorLocation = adUseClient
  db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data Source=.\GanttDemoData.mdb;"

  Set adoIDValueRS = New Recordset
  adoIDValueRS.Open "select lastid from idvalue", db, adOpenStatic, adLockOptimistic

    phGantX1.LoadAndAddIconToList ".\IconForTree.ico"
    phGantX1.LoadAndAddIconToList ".\IconForTree2.ico"

Dim activity As IphDataEntity_Tree
 LoadTreeItemsOwnedBy activity, -1

End Sub
```

In the sample code we create a new database connection and hold in a global variable called dB. We open the database. We also create a recordset that reads from the idvalue table and hold it for later use. We also make the Gantt aware of the icons that we intend to use in the tree. Then we call a sub named LoadTreeItemsOwnedBy and this sub is responsible for loading treenodes on one level , in this case the root level.

```
Private Sub LoadTreeItemsOwnedBy(activity As IphDataEntity_Tree, ownerid As Integer)
Dim newactivity As IphDataEntity_Tree
Dim aRS As Recordset
Dim selectstat As String

    selectstat = "select id,name,owningactivity from activity where owningactivity="
    selectstat = selectstat + Str(ownerid)
    Set aRS = New Recordset
    aRS.Open selectstat, db, adOpenStatic, adLockReadOnly
    While Not aRS.EOF
        If activity Is Nothing Then
            Set newactivity = phGantX1.AddRootDataEntityTree
        Else
            Set newactivity = phGantX1.AddDataEntityTree(activity)
        End If

        InitActivity newactivity, aRS
        LoadTreeItemsOwnedBy newactivity, newactivity.UserIntegerReference
        aRS.MoveNext
    Wend
End Sub
```

The sub above opens the activity table and reads all rows that has a owningactivity matching the supplied ownerid, for root nodes this will be –1. We iterate over the RecordSet until it is empty. For each row, we do three things. First, we check if we have a supplied activity that should own the one read, if not then it is a root activity being read. We then create a DataEntity to symbolize the new activity and hand it to a sub called InitActivity that will move values from the current row of the record set to the DataEntity. Last, we do a recursive call to the same sub that we are in to fetch and instansiate the subnodes of the newly created activity.

```
Private Sub InitActivity(activity As IphDataEntity_Tree, aRS As Recordset)
    activity.UserIntegerReference = aRS.Fields.Item("id").Value
    activity.Text = aRS.Fields.Item("Name").Value
    activity.CanEdit = True

    LoadTimeItemsOwnedBy activity
End Sub
```

In the Sub above, we initialize the activity by taking values from the current row of the recordset and putting it to the DataEntity item. Note that we also take the primary key from the record set and put it in one of the user controlled fields of the DataEntity, we need this later when it is time to update the database with changes made to the DataEntity.

At the very end, we call the Sub LoadTimeItemsOwnedBy that will load all the work time items for the activity,

```
Private Sub LoadTimeItemsOwnedBy(activity As IphDataEntity_Tree)
Dim newtimeItem As IphDataEntity_GantTime
Dim aRS As Recordset
Dim selectstat As String

    selectstat = "select id,activity,start,stop from worktime where activity="
    selectstat = selectstat + Str(activity.UserIntegerReference)
    Set aRS = New Recordset
    aRS.Open selectstat, db, adOpenStatic, adLockReadOnly
    While Not aRS.EOF
        Set newtimeItem = phGantX1.AddGantTime(activity, 0)
        InitTimeItem newtimeItem, aRS
        aRS.MoveNext
    Wend

End Sub
```

The Sub above is almost analogous to the one loading in the activities. We send a SQL statement that gives us all time items owned by a particular activity. We then iterate over the rows and add a Gantt time for each row. Then we call a new sub; InitItem. InitItem will take the values from the current row of the supplied recordset and put them into the new time item.

```
Private Sub InitTimeItem(newtimeItem As IphDataEntity_GantTime, aRS As Recordset)
    newtimeItem.CanEdit = True
    newtimeItem.Start = aRS.Fields.Item("start").Value
    newtimeItem.Stop = aRS.Fields.Item("stop").Value
    newtimeItem.UserIntegerReference = aRS.Fields.Item("id").Value
    newtimeItem.Color = ColorConstants.vbBlue
End Sub
```

The sub above moves values from the recordset to the time item. Note that the primary key also is taken from the recordset and put into the userIntegerReference property, it will be needed when we save data back to the db.

Ok, our work here is done. We have populated the Gantt with data from a database. However, what if it was empty from the start, then we wont get any output... I guess we need to save stuff in there too...

Save data with ADO from the phGantt

We want to save data that now resides in the phGantt component. Fine, then we must iterate over it, lets start with the root tree nodes.

```
Private Sub cmdSave_Click()
Dim activity As IphDataEntity_Tree

    Set activity = Nothing
    SaveActivity phGantX1.RootDataEntitiesTree, activity


End Sub
```

We call a sub called SaveActivity that apparently does all the work. It looks like this:

```
Public Sub SaveActivity(aphDataList As IphDataList, aOwner As IphDataEntity_Tree)
Dim activity As IphDataEntity_Tree
Set aRS = New Recordset
Dim upsertstat As String

        For i = 0 To aphDataList.Count - 1
            Set activity = aphDataList.Items(i)

            owningactivity = -1
            If Not aOwner Is Nothing Then
                owningactivity = aOwner.UserIntegerReference
            End If
            If activity.UserIntegerReference = 0 Then
                id = GetNewID
                upsertstat = "insert into activity (id,name,owningactivity)"
                upsertstat = upsertstat + " values(" + Str(id) + ",'"
                upsertstat = upsertstat + activity.Text + "'," + Str(owningactivity) + ")"
                activity.UserIntegerReference = id
            Else
                upsertstat = "update activity set name='"
                upsertstat = upsertstat + activity.Text + "' where id="
                upsertstat = upsertstat + Str(activity.UserIntegerReference)
            End If

            Set aRS = New Recordset
            aRS.Open upsertstat, db, adOpenStatic

            SaveTimeItems activity ' Save corresponding time items

            SaveActivity activity.SubDataList, activity ' traverse owned nodes
        Next

End Sub
```

The sub above receives a list of dataentities, which it iterates. For each DataEntity, it does four things. The first thing is to decide if the DataEntity is a subnode or not, if it is a sub node we need to know because we want to put the foreign key to the owner in the database. The second thing we do is to check if the UserIntegerReference is zero or not. If it is zero we take this as a sign that this particular item is newly created in this session and is not represented in the database yet. If this is the case we must use an insert statement instead of an update statement. We also need to get a new unique identity to put in the primary key field. The primary key is fetched by the sub GetNewID, described further down this document.

The third thing we do is to execute the compiled SQL statement. Moreover, the fourth thing is to save the corresponding time items. And the fifth thing is to call our self in a recursive loop to do the same thing for our sub nodes.

```
Public Function GetNewID()
    If adoIDValueRS.RecordCount = 0 Then
        adoIDValueRS.Close
        adoIDValueRS.Open "insert into IDValue (lastid) values (1)", db
        GetNewID = 1
    Else
        adoIDValueRS.Close
        adoIDValueRS.Open "update IDValue set lastid=lastid +1 ", db
        adoIDValueRS.Open "select lastid from IDValue", db

        GetNewID = adoIDValueRS.Fields.Item("LastID").Value

    End If
```

```
End Function
```

The sub above makes sure that we always get a unique id for our newly created items. This is important if we are in a multi user environment. There are many ways to retrieve unique keys, many databases has built in support for this operation.

We also called the sub SaveTimeItems earlier, so let us look on that:

```
Public Sub SaveTimeItems(activity As IphDataEntity_Tree)
Set aRS = New Recordset
Dim upsertstat As String
Dim starttime, stoptime As String
Dim gantRow As IGantRow
Dim ganttime As IphDataEntity_GantTime
    Set gantRow = phGantX1.RowList.FindRowFromTreeNode(activity)


        For i = 0 To gantRow.DataLists.DataList(0).Count - 1
            Set ganttime = gantRow.DataLists.DataList(0).Items(i)
            starttime = "'" + Str(ganttime.Start) + "'"
            stoptime = "'" + Str(ganttime.Stop) + "'"
            activityid = activity.UserIntegerReference


            If ganttime.UserIntegerReference = 0 Then
                id = GetNewID
                upsertstat = "insert into worktime (id,activity,start,stop) values("
                upsertstat = upsertstat + Str(id) + "," + Str(activityid) + ","
                upsertstat = upsertstat + starttime + ","
                upsertstat = upsertstat + stoptime + ")"
                ganttime.UserIntegerReference = id
            Else
                upsertstat = "update worktime set activity=" + Str(activityid)
                upsertstat = upsertstat + ",start=" + starttime + ",stop=" + stoptime
                upsertstat = upsertstat + " where id=" + Str(ganttime.UserIntegerReference)
            End If

            Set aRS = New Recordset
aRS.Open upsertstat, db, adOpenStatic
        Next
End Sub
```

As you can see this sub is almost the same as the one saving the activities, and there is really nothing to add.

When you run the sample code you will notice that all the formatting of treenodes and color and drawing styles of time items are lost when we save and reload. This would be a good exercise to extend the database tables to allow them to hold this formatting information and both save it and load it.

References

Tidomat is working on products using plexityHide components.

Swedish television broadcasting cooperation has built program planning and project planning systems using phGantt and phSchema.

The Swedish parliament builds a workflow system for their production of documents. PlexityHide components are used to visualize planning and current state of documents.

Boldsoft are framework builders, one of a kind. PlexityHide support this framework, both phGantt and phSchema can be delivered Bold-Aware.

Openinfo is the software-consulting firm of Stockholm that first started the development of the plexityHide time components.

Buy plexityHide components at ComponentSource.