Virtualload in the phGant

Background and introduction

Virtual load allows the phGant to handle unloaded rows. Before virtual load was supported all rows in the phGant was instantiated and loaded prior to the first draw operation. To instantiate rows is resource consuming, and users that needed to use the phGant as a "window" to very large datasets suffered from this memory and time consumption.

Virtual load basically allows the developer to tell the phGant about the complete amount off rows in very large or gigantic datasets without being punished in extreme use of memory or time. The Virtual load functionality loads and unloads phGantRows as they are needed. The need for a particular row is based on that it is on screen or that it has state that the programmer do not want to persist in order to recreate at a later time (typically selections). A row can also be needed if it is part of a tree hierarchy and one of its child nodes are needed.

Add unloaded and Purge

All concerned lists in the phGant (except the RowList, RowList is managed by the grid, and only for loaded rows) have been given the ability to accept unloaded rows and to purge unused rows. The key methods are:
AddUnloaded(x) – Tells the list to add unloaded rows to the list. These rows are never loaded unless they are needed.
Purge() - Tells the list that it should scan for unused rows and try to unload them (this is done automatically for the grid).

Load on demand

All concerned lists in the phGant (except the RowList) have been given a load on demand behaviour. This means that if you were to ask for an item that is not loaded yet, using the Items property, it will be loaded in order to give you the result. In order to avoid unwanted loading of items new properties has been added:
IsLoaded(x) – Answers the question if a given index is loaded or not.
LoadedCount() - Tells you how many items that are currently loaded.
Loaded[x] – Gives you one of the loaded items, will never trigger load on demand.

Developer interaction

The virtual load functionality has been added to the phGant and phGrid and as such it needs developer interaction when loading and unloading rows. We also need for the developer to say if it is ok to purge a row or if we should keep it for some reason. Also when an operation that requires the complete amount of rows, like sorting, is performed we need to give the developer a choice of solving this outside the control. The key events are:

PhGant.OnVirtualLoad_Grid – The control is in need of an unloaded row, you should fill it with the correct data.
PhGant.OnVirtualUnLoad_Grid – The control has detected that this row is not needed, you should answer if it is OK to unload, and possibly save any state changes back to the dataset.
PhGant.OnColumnSort_Grid – The grid header has been clicked, you can choose what the reaction should be. The default response is to load all rows and resort them.

Limitations

The virtual load is implemented as a loaded window in an unloaded list. This means that the loaded items are always in contact with each other. If the first item is loaded and not allowed to unload, and the last item is required, then all items in between will be loaded automatically. The developer should be aware of this implementation in order to avoid forcing the phGant to load to many rows.

If the users make use of the multi select function, and marks the first row and tries to select all rows by using shift and ctrl-end, an internal safe-catch will limit the selection to the last row and 500 hundred rows above. If this safe-catch did not exist, all rows would be needed in order to mark them as selected.

Examples and reference

```
void __fastcall TGanttView::phGant1VirtualLoad_Grid(
      TBabGridCell *theCaller, TAxisContentItem *theAxisItem,
      bool theRowNotColumn)
{
  if (theRowNotColumn) //Only for rows, in this example we do not show virtual cols
  {
    int fixed = theAxisItem->AxisContentList->FixedItems;
    int i = theAxisItem->ListIndex() - fixed;
    VirtuelTestData *p = virtuelTestDatas[i];


    // Init the cells in the grid
    phGant1->Grid->SetText(phGant1->Grid->Cell[1][i+fixed], IntToStr(p->id));
    phGant1->Grid->SetText(phGant1->Grid->Cell[2][i+fixed], IntToStr(p->date));


    // Add a time item to the gantRow
    TphDataEntity_GantTime *timeitem = phGant1->GridNode_AddGantTime(
                phGant1->Grid->AxisContentItemToGridTreeNode(theAxisItem),0);

    timeitem->Start = (double) p->date;
    timeitem->Stop=(double) (p->date+5);
    timeitem->Style =gtsPipe ;
    timeitem->Color = clYellow;
  }
}


void __fastcall TGanttView::phGant1VirtualUnLoad_Grid(
      TBabGridCell *theCaller, TAxisContentItem *theAxisItem,
      bool theRowNotColumn, bool UnloadPerform, bool &CanUnload)
{
  if (theRowNotColumn) // Only for rows
  {
    CanUnload=true;  // Important answer, if we say FALSE the row will not unload
    if (UnloadPerform)
    {
      // Save any state-changes if it is not already done
```

```
    }
  }
  else
  {
    CanUnload=false; //Since we do not want virtual columns we never unload them
  }
}


void __fastcall TGanttView::phGant1ColumnSort_Grid(TphGrid *aGrid,
      TphGrid_Column *aColumn, bool aUpNotDown, bool &GoOnInternalSort)
{

   GoOnInternalSort=false; // Prevent default sorting (load and compare all rows)
   aGrid->MainGrid->AxisContentListY->UnloadAll(); // On next Paint, all visible data
will be reloaded, Great if we have re-shuffled our dataset

   /* Resort all content here
   switch (aColumn->Index) {
      case 0:    ListView1->CustomSort(CustomSortCompareId, sortDirections[Column-
>Index]); break;
      case 1:    ListView1->CustomSort(CustomSortCompareSize, sortDirections[Column-
>Index]); break;
   }*/
}
```

```
  TAxisContentList=class // Used by grid to hold x and y axis
  public
    constructor create(theBabGridCell:TBabGridCell);
    destructor  Destroy;override;
    function    getGridCell:TBabGridCell;
    procedure   Drop(i:integer);
    procedure   Clear;
    procedure   ClearKeepFixedItems;
    procedure   ClearVisibles;
    function    InsertNew(i: integer; theValue: TObject):TAxisContentItem;
    function    AddNew(theValue: TObject):TAxisContentItem;
    function    IsLoaded(x: Integer):Boolean;
    procedure   AddUnloaded(x: Integer);
    procedure   UnloadAll;
    procedure   Purge;
    procedure   Move(curIndex,newIndex:integer);
    function    Count:Integer;
    function    VisibleCount:Integer;
    function    LoadedCount:Integer;

    property    Items[i:integer]:TAxisContentItem
    property    Visibles[i:integer]:TaxisContentItem
    property    Loaded[i:integer]:TaxisContentItem
    property    HiddenRowCount:integer read GetHiddenRowCount;
    property    FixedItems:Integer read fFixedItems write fFixedItems;
    property    LoadOperationInFront:Boolean read fLoadOperationInFront;
  end;


  TphDataList=class(TphExposable) // Used by Tree and time items lists
  public
    destructor  Destroy;override;
    function    FindDataEntity(theFindFunc:TFindFunc):TphDataEntity;
    procedure Sort(theDescending:Boolean);
    procedure ClearSelections;
    function  Add:TphDataEntity;
    function  Insert(theIndex:Integer):TphDataEntity;
    procedure AddUnloaded(x:Integer);virtual;
    function  IsLoaded(x: Integer): Boolean;
    procedure UnloadAll;
```

```
  procedure Purge;
  procedure Clear;
  function  Count:Integer;
  function  LoadedCount:Integer;
  procedure Delete(theIndex:integer);
  procedure Remove(theDataEntity:TphDataEntity);
  function  IndexOf(theDataEntity:TphDataEntity):integer;
  function  VisibleCount:integer;

  property  Items[index: integer]: TphDataEntity;
  property  Loaded[index: integer]: TphDataEntity;
  property  ListController:TphListController;
end;
```