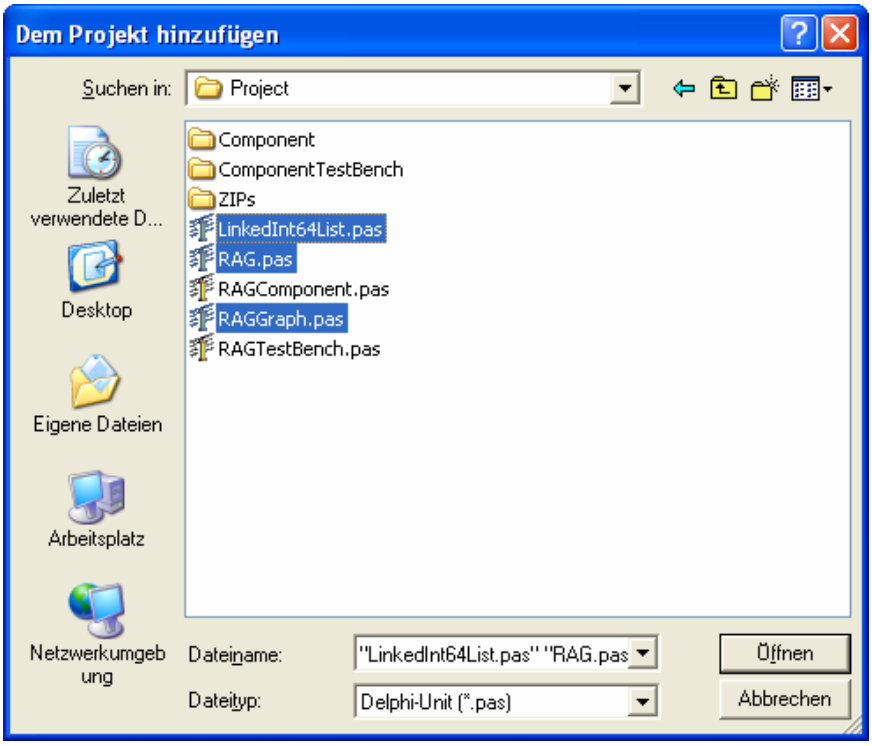
Tutorial 1: First steps

Welcome to my tutorial which is going to teach you how to integrate the RAG Component in a Delphi project and how to use it. I hope I will have enough time to write some more tutorials to get deeper into the RAG Component. So let's get busy!

Project Setup

First you should start Delphi and if there's no blank project you should also create one. After you've done that press the `SHIFT + F11` Keys to get "Add Project Items" Dialog. Add the following four files to your project: `LinkedInt64List.pas`, `RAGGraph.pas` and `RAG.pas` (see figure 1).



Now that we've added the files to our project we can do some form design which is necessary to display a RAG graph. So open the form designer and place an Image Control on the form. Let's choose a width of 350 pixels and a height of 200 pixels (you can choose a different size, too). Locate the Image Control in the left upper corner of the form and rename it to `imgGraph`. Please add a Button Control and name it `btnAddRandomValue`. Place it somewhere and set a caption like "Add Random Value".

Coding

The UI is complete as far, so we can open the source editor of the form. The first thing we have to do here is to add the RAG Units to the `uses` list: `LinkedInt64List, RAG, RAGGraph`. After that it should look like in figure 2:

```
interface

uses
  Windows, Messages, SysUtils, Variants, Class
  Dialogs, LinkedInt64List, RAG, RAGGraph;

type
  TForm1 = class(TForm)
```

After we've referenced the RAG Units we can declare private variable called `_rag` and another variable called `_graph`:

```
...
  private
    _rag : TRAG;
    _graph : TRAGGraph;
...
```

Now create the `FormCreate` procedure and add the following code:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  _rag := TRAG.Create(imgGraph.Width, imgGraph.Height);
  _graph := TRAGGraph.Create(175);
end;
```

In the first line we create a new `TRAG` object with the width and the height of the graph as constructor parameters. In the second line a new `TRAGGraph` object is created with the length as constructor parameters. The length of a graph indicates how many values in this graph can be stored. A length of 175 values means that the graph is actually 175 pixels wide. But this looks very strange so we set the `XScalar` property to 2. This causes the graph to scale its width from 175 pixels to 350 pixels and increases the distance between two points to 2 pixels.

```
...
  _rag.XScalar := 2;
```

In order to customize the appearance of our graph we add some further code to the `FromCreate` procedure:

```
...
  _rag.GridStepHorz := 10;
  _rag.GridStepVert := 20;
  _rag.GridColor := clGreen;
  _rag.BackColor := clBlack;
```

The `GridStepHorz` property retrieves or sets the horizontal grid step and the `GridStepVert` property does this in vertical direction. Just for fun I set the `GridColor` property of the `_rag` variable to `clGreen` and the `BackColor` to `clBlack`. But at the moment the `_rag` variable does not contain any graphs. We change this by adding the following code to the FormCreate procedure.

```
...
  _graph.Color := clLime;
  _rag.AddGraph(_graph);
```

The first line just changes the line color of the graph to clime but the second line is really important. Here we add the graph to the list of graphs managed by the RAG Component. That means the `_rag` variable does all the calculating and drawing of every graph

contained. The last thing we have to do in the `FormCreate` procedure is to add code for the first rendering of the RAG:

```
...
  _rag.RedrawGraphs(true);
  imgGraph.Picture := _rag.Picture;
```

Everytime you call the `RedrawGraphs` procedure of the `_rag` variable the graphs are rendered onto the picture encapsulated by the `Picture` property. In order to display this picture we assign it to the Image Control.

Now the initializing of the RAG Component is done, so that we can focus on adding values to the RAG. Remember that we've added a button to the form and add the `Click` procedure. For the beginning we just add random values to the graph:

```
...
  _rag.AddValues([Random(100)]);
  _rag.RedrawGraphs(true);
  imgGraph.Picture := _rag.Picture;
```

In the first line the value gets added to the graph. Notice that we call the AddValues method from the `_rag` object and pass an array. The reason is the movement of the grid. If we pass all the values synchronous to the RAG, it is abled to move the grid to the left. Just start the project and keep pressing the add value button about 176 times…

The last thing we will do in this tutorial is to add support for changing dynamically the graph type. For this purpose add a ComboBox to the form and give it four items: "Lines", "Area", "Bars (Absolute)", Bars (Relative)" and add the Change procedure. Jump to the Change procedure and add the following code:

```
procedure TForm1.cboGraphTypeChange(Sender: TObject);
begin
  if cboGraphType.ItemIndex = 0 then
  begin
    _graph.GraphType := gtLines;
  end
  else if cboGraphType.ItemIndex = 1 then
  begin
    _graph.GraphType := gtArea;
  end
  else if cboGraphType.ItemIndex = 2 then
  begin
    _graph.GraphType := gtBarsAbsolute;
  end
  else if cboGraphType.ItemIndex = 3 then
  begin
    _graph.GraphType := gtBarsRelative;
  end;

  _rag.RedrawGraphs(true);
  imgGraph.Picture := _rag.Picture;
```

```
end;
```

We just check the selected item of the combo box and assign the new graph type. After we've done this, the graph gets recalculated and redrawn.

Now we're already done with this tutorial. As I already said, I hope I will have enough time to write more tutorials. In order to close this tutorial a small screenshot of the sample application: