Nat Table

This tutorial shows how to get started with Nebula NatTable. It shows how to get NatTable installed into the IDE and explains the basic concepts. It is also explained how to put data in a NatTable instance and how to create a NatTable with various compositions.

• N(ot) a(nother) t(able)

• Framework for building tables/grids/trees

• Designed to handle large datasets

• Provides a lot of functionality out of the box

How we create a table using in TableViewer

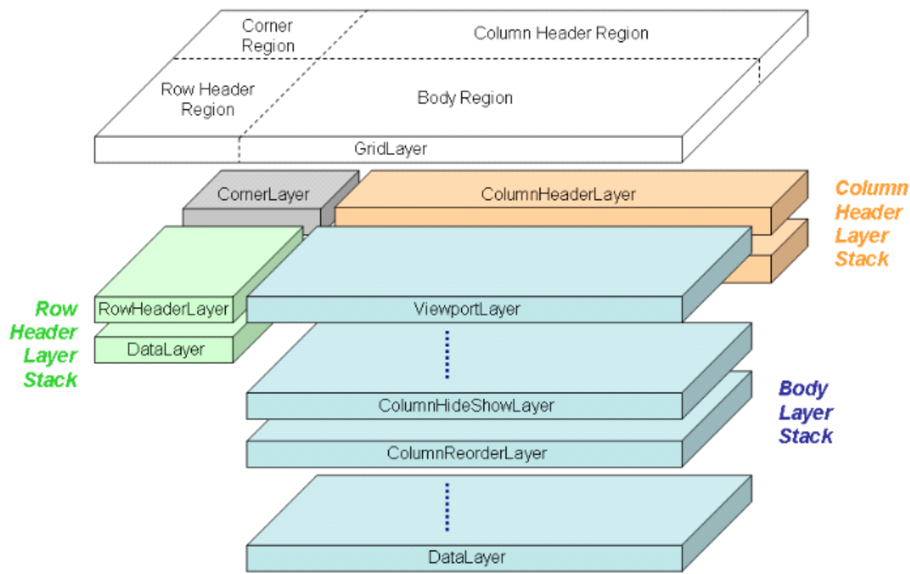
```
TableViewer viewer = new TableViewer(parent, SWT.BORDER|SWT.H_SCROLL|SWT.V_SCROLL);
TableViewerColumn column = new TableViewerColumn(viewer, SWT.NONE);
column.getColumn().setText("Firstname");
column.getColumn().setWidth(100);
column.setLabelProvider(new CellLabelProvider() { ... });
column.setEditingSupport(new MyEditingSupport());
```

- Created column by column
- Several objects for every column – Column object, LabelProvider, ContentProvider(EditingSupport) and setInput for tableViewer.

Before going to start creating nattable we need to understand the architecture and layers of the NatTable

If Nattable was not installed in your eclipse version,then you should download it from internet and install it into eclipse.
A NatTable instance is typically build out of several layers. A layer is a rectangular region of cells and has methods to access columns, rows, width and height. All layers implement the ILayerinterface. Every basic feature is implemented in a layer and can be added to a NatTable instance by adding the layer to a layer stack or a layer composition. For example, to add the ability to select cells in a table you need to add the SelectionLayer.

ANCIT
CONSULTING



We have to add "org.eclipse.nebula.widgets.nattable.core" in dependency

Example Code:

```
String[] propertyNames = { "name", "address", "contact" };

IColumnPropertyAccessor<Contact> columnPropertyAccessor = new ReflectiveColumnPropertyAccessor<Contact>(
        propertyNames);
IDataProvider bodyDataProvider = new ListDataProvider<Contact>(addressBook.getContacts(),
        columnPropertyAccessor);

final DataLayer bodyDataLayer = new DataLayer(bodyDataProvider);

final NatTable natTable = new NatTable(parent, SWT.NO_REDRAW_RESIZE | SWT.DOUBLE_BUFFERED | SWT.BORDER,
        bodyDataLayer);

GridDataFactory.fillDefaults().grab(true, true).applyTo(natTable);
```

- DataLayer – IDataProvider (e.g. ListDataProvider)
- IColumnAccessor (e.g. ReflectiveColumnPropertyAccessor)
- No JFace databinding support out of the box
- GlazedLists support

Working With Example

In this section you will find some examples on how to use the NatTable.

Creating a basic grid

In this example we will create a basic grid that will show objects of type Contact. You will need to the follow these steps to setup the grid:

- Assemble the Layer stacks depending on the features you wish to enable. Every region has a layer stack backing it.

In this example we will create a grid with the following features enabled.

ANCIT CONSULTING

- Reorder columns
- Hide columns
- Scrolling
- Selection

Plugging data

The primary interface for providing data to NatTable is `IDataProvider`. The most common way of providing data to the table is to use a List data structure. This list contains an object for each row in the table. Each property of the row object is represented in a column.

In this example we are using ecore model/domain mode named Contact to represent the data in a row.

```
AddressBook addressBook = AddressbookFactory.eINSTANCE.createAddressBook();
addressBook.setName("Phone Book");

Contact contact = AddressbookFactory.eINSTANCE.createContact();
contact.setName("Rohit");
contact.setAddress("Bangalore");
contact.setContact("8974562541");

addressBook.getContacts().add(contact);
```

If you are using a List as your data structure, you can use the `ListDataProvider` out of the box. In this case our data provider will look like so

```
String[] propertyNames = { "name", "address", "contact" };
IColumnPropertyAccessor<Contact> columnPropertyAccessor = new ReflectiveColumnPropertyAccessor<Contact>(
        propertyNames);
IDataProvider bodyDataProvider = new ListDataProvider<Contact>(addressBook.getContacts(),
        columnPropertyAccessor);
```

The `ReflectiveColumnPropertyAccessor` uses standard java getter methods to read data from the row object. If you wish to fetch data from your row object in specific ways, you can plugin a custom `IColumnPropertyAccessor` here.

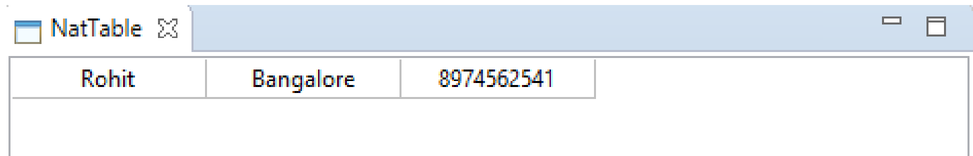
Setting up the body region

The data provider should be wrapped up by the `DataLayer`. `The DataLayer` is always the lowermost layer in the stack. It is responsible for providing data to the grid.

```
String[] propertyNames = { "name", "address", "contact" };

IColumnPropertyAccessor<Contact> columnPropertyAccessor = new ReflectiveColumnPropertyAccessor<Contact>(
        propertyNames);
IDataProvider bodyDataProvider = new ListDataProvider<Contact>(addressBook.getContacts(),
        columnPropertyAccessor);

final DataLayer bodyDataLayer = new DataLayer(bodyDataProvider);

final NatTable natTable = new NatTable(parent, SWT.NO_REDRAW_RESIZE | SWT.DOUBLE_BUFFERED | SWT.BORDER,
        bodyDataLayer);

GridDataFactory.fillDefaults().grab(true, true).applyTo(natTable);
```

ANCIT CONSULTING

Run the Application :



ABOUT ANCIT:

ANCIT Consulting is an Eclipse Consulting Firm located in the "Silicon Valley of Outsourcing", Bangalore. Offers professional Eclipse Support and Training for various Eclipse based Frameworks including RCP, EMF, GEF, GMF. Contact us on annamalai@ancitconsulting.com to learn more about our services.

ANCIT
CONSULTING