CONSUMING an EMF MODEL in UR ECLIPSE UI

PART 6 – Displaying EMF Model on a TableViewer

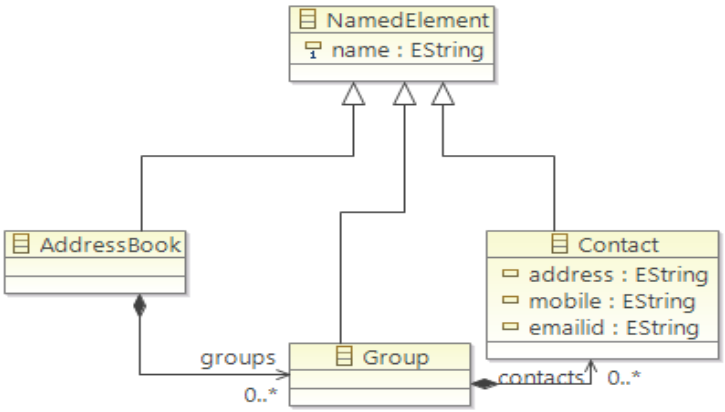
From Part 1 to Part 5 we have been discussing on Edit Plugins support for displaying and editing of EMF Models on a TreeViewer. In this tutorial we are about to discuss the use of the Edit Plugin to display the EMF Model on a TableViewer.

```
public class CustomerItemProvider
    extends ItemProviderAdapter
    implements
        IEditingDomainItemProvider,
        IStructuredItemContentProvider,
        ITreeItemContentProvider,
        IItemLabelProvider,
        IItemPropertySource {
```

Typically any EMF generated Item Provider would look like the above shown screenshot. It extends from ItemProviderAdapter and implements IEditingDomainItemProvider, IStructuredItemContentProvider, ITreeItemContentProvider, IItemLabelProvider, IItemPropertySource

Looking at the interfaces IStructuredItemContentProvider, it is clear that this ItemProvider is ready to work for a TableViewer as a ContentProvider but as a LabelProvider ?? The answer is, it is not. IItemLabelProvider supports only getText which is good for a single column table or a list or a treeviewer but not for a multi column tableviewer.

This requires a bit of a customization to the edit plugin to support Tableviewer with custom columns on it. In this tutorial we would learn what is required to customize the edit plugin to work with a tableviewer to display the relevant column information.



Author : Annamalai C
Date : August 20th, 2014

ANCIT CONSULTING

Now the example that I am developing is, whenever a Group is selected in the EMF generated Editor, it needs to display Contacts in a tabular format in my custom view.

1. Create a New Plugin [org.ancit.examples.emf.edit.extension]
2. Add dependancies [org.ancit.examples.emf.edit] to the newly created plugin
3. We need to extend 2 Classes. The ItemProviderAdapterFactory and the ItemProvider itself.
4. Create a new Class called as ExtendedAddressBookItemProviderAdapterFactory which extends AddressBookItemProviderAdapterFactory.
5. Create a Default Constructor for the new Class.
6. To the list of supportedTypes, add ItableItemLabelProvider as shown below

```
public class ExtendedAddressBookItemProviderAdapterFactory extends
        AddressbookItemProviderAdapterFactory {

    public ExtendedAddressBookItemProviderAdapterFactory() {
        supportedTypes.add(ITableItemLabelProvider.class);
    }
```

7. Now override createContactAdapter() method and add the following code snippet

```
@Override
public Adapter createContactAdapter() {
    if(contactItemProvider == null) {
        contactItemProvider = new ExtendedContactItemProvider(this);
    }
    return contactItemProvider;
}
```

The above codesnippet is about instead of returning the generated ContactItemProvider which doesnt support ItableItemLabelProvider, we return our extended version of ContactItemProvider which implements ItableItemLabelProvider.

8. Create a new Class called as ExtendedContactItemProvider which extends ContactItemProvider and implements ITableItemLabelProvider.

```
public class ExtendedContactItemProvider extends ContactItemProvider implements ITableItemLabelProvider{
```

9. Override getColumnText() and provide your implementation to handle various columns. The method would look like as shown below

```
@Override
public String getColumnText(Object object, int columnIndex) {
    if (object instanceof Contact) {
        Contact contact = (Contact) object;
        switch (columnIndex) {
        case 0:
            return contact.getName();
        case 1:
            return contact.getAddress();
        case 2:
            return contact.getMobile();
        case 3:
            return contact.getEmailid();
        default:
            break;
        }
    }

    return super.getColumnText(object, columnIndex);
}
```

Author : Annamalai C
Date : August 20th, 2014

ANCIT CONSULTING

10. Now that the AdapterFactory and ItemProvider has been extended, we are ready to use it in our TableViewer. Therefore create a new plugin [org.ancit.examples.emf.ui]

11. In the ComposedAdapterFactory instead of adding AddressbookItemProviderAdapterFactory you need to add ExtendedAddressbookItemProviderAdapterFactory and the view class would look like as shown below.

```
public class EMFTableView extends ViewPart implements ISelectionListener {

    private ComposedAdapterFactory adapterFactory;
    private TableViewer tViewer;

    public EMFTableView() {
        adapterFactory = new ComposedAdapterFactory();
        adapterFactory.addAdapterFactory(new ResourceItemProviderAdapterFactory());
        adapterFactory.addAdapterFactory(new ExtendedAddressBookItemProviderAdapterFactory());
        adapterFactory.addAdapterFactory(new ReflectiveItemProviderAdapterFactory());
    }

    @Override
    public void createPartControl(Composite parent) {
        tViewer = new TableViewer(parent);
        tViewer.setContentProvider(new AdapterFactoryContentProvider(adapterFactory));
        tViewer.setLabelProvider(new AdapterFactoryLabelProvider(adapterFactory));

        getSite().getPage().addSelectionListener(this);
    }

    @Override
    public void setFocus() {
        // TODO Auto-generated method stub

    }

    @Override
    public void selectionChanged(IWorkbenchPart part, ISelection selection) {
        if (selection instanceof IStructuredSelection) {
            IStructuredSelection sSelection = (IStructuredSelection) selection;
            Object firstElement = sSelection.getFirstElement();
            if (firstElement instanceof Group) {
                Group group = (Group) firstElement;
                tViewer.setInput(group);
            }
```

12. We are good to run and test. You need to add Columns in the desired order to the TableViewer which I havnt done in the above shown code.