Eclipse, Java and Web development - Free Tutorials

Using JFace Tables 3.3 API with Eclipse RCP - Tutorial

Clemens Muessener

<webmaster@vogella.de>

Additons by Hendrik Still

Version 0.5

15.12.2007

Abstract

This article is a How-to-Guide to build an Eclipse RCP application with a JFace table using the new 3.3 API. Different editors and label provider will be used for different data types. The article demonstrate also how to copy text into the system clipboard using SWT.

While we are creating an Eclipse RCP application the approach described here can also be used for building non RCP applications, plug-ins.

This article assumes you have already a good general knowledge about using Eclipse.

Table of Contents

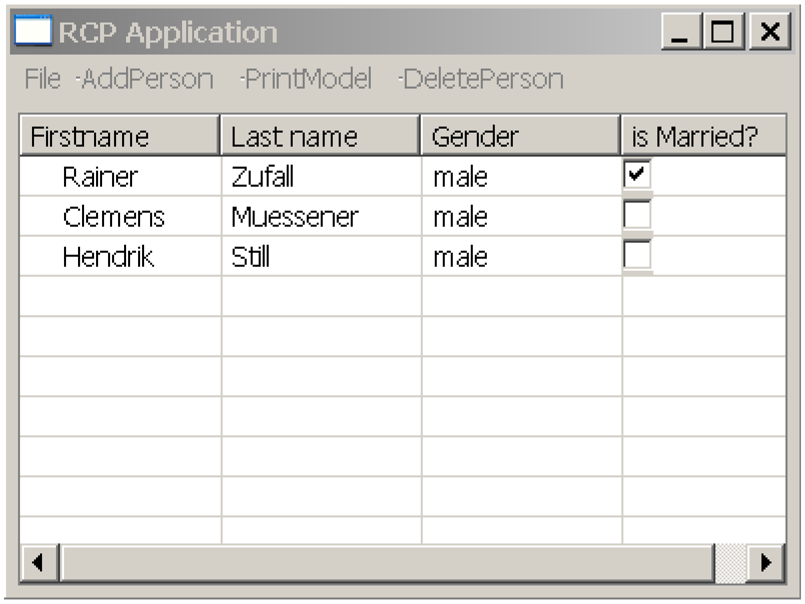
1. Scenario

In this tutorial we will build a Eclipse RCP application which will display a domain model in a JFace table. The domain model which is used represents simply the data of persons (first name, last name, gender and if he/she is married). Each person is displayed in one row.

The application will also be able to change the first name and last name like a text editor, the gender by a drop down list and the married status by a checkbox.

Via the menu it will be possible to add new persons to the table and to remove persons from the table.

The final application will look like this.



Firstname	Last name	Gender	is Married?
Rainer	Zufall	male	✓
Clemens	Muessener	male	☐
Hendrik	Still	male	☐

2. Create a project

Create a new RCP Project "TableTesting" (see Create your first Eclipse RCP application for details). Use the "RCP application with a view" as a template.

3. Create the model

Create a package tabletesting.model

Create a new class with the name "Person". This class contains the first name, last name, gender and married status for a person and represents the data model for this example.

```
package tabletesting.model;

public class Person {
	private String firstName;
	private String lastName;
	private boolean isMarried;
	private String gender;

	public Person(String firstName, String lastName, boolean isMarried,
			String gender) {
		this.firstName = firstName;
		this.lastName = lastName;
		this.isMarried = isMarried;
		this.gender = gender;
	}

	public String getFirstName() {
		return firstName;
	}

	public String getGender() {
		return gender;
	}

	public String getLastName() {
		return lastName;
	}

	public boolean isMarried() {
		return isMarried;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public void setGender(String gender) {
		this.gender = gender;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public void setMarried(boolean isMarried) {
		this.isMarried = isMarried;
	}

}
```

Now create a class, called ‘ControlPersons’. In this class we manage a collection of persons. This class is defined as Singleton (see The Singleton Pattern for details).

```
package tabletesting.model;

import java.util.ArrayList;
import java.util.List;

public class ControlPersons {

        private static ControlPersons content;
        private List<Person> personsList;

        private ControlPersons() {
                personsList = new ArrayList<Person>();
        }

        public synchronized static ControlPersons getInstance() {
                if (content != null) {
                        return content;
                } else {
                        content = new ControlPersons();
                        return content;
                }
        }

        public List<Person> getPersonsList() {
                return personsList;
        }
}
```

4. JFace viewer concept

The application you have created already contains a pre-defined view. This is because we used the template "RCP application with a view". In this view a content provider and a label provider are given and assigned to a table viewer.

The new 3.3 API allow you to assign label provider directly to columns. The predefined example uses one label provider for the whole table. We will change this later.

The content provider class is responsible for providing objects to the view. It can simply return objects as-is. The label provider defines how these objects are displayed, e.g. which part of the object is used to return the text which is displayed. The user interface is then re-presented by the viewer, e.g. in our example the table viewer.

This concept is very flexible as it allows to use the some content provider / label provider in different viewers without changing the underlying data structure. Have a look at class View.java to investigate the pre-defined coding.

5. ContentProvider

5.1. Build your ContentProvider

Create your own Content Provider. First create the package tabletesting.provider. Create a new class which is called "MyContentProvider" and implement the Interface "org.eclipse.jface.viewers.IStructuredContentProvider".

Here we use the classes "ControlPersons" and "Person" to save the data model. The main purpose of the class "MyContentProvider" is to read fill the data model and to return the data as an array.

Tip

It is important to note that within the content provider no assumption is made how and which data of the data model is displayed by the view. This is the responsibility of the label provider.

```
package tabletesting.provider;

import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.Viewer;
import tabletesting.model.ControlPersons;
import tabletesting.model.Person;

public class MyContentProvider implements IStructuredContentProvider {

        ControlPersons persons;

        public MyContentProvider() {
                persons = ControlPersons.getInstance();
                // Image here some fancy database access to read the persons and to
                // put them into the model
                Person person;
                person = new Person("Rainer", "Zufall", true, "male");
                persons.getPersonsList().add(person);
                person = new Person("Clemens", "Muessener", false, "male");
                persons.getPersonsList().add(person);
                person = new Person("Hendrik", "Still", false, "male");
                persons.getPersonsList().add(person);

        }

        @Override
        public Object[] getElements(Object inputElement) {
                return persons.getPersonsList().toArray();
        }

        @Override
        public void dispose() {
        }

        @Override
        public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {

        }

}
```

5.2. Use the content provider

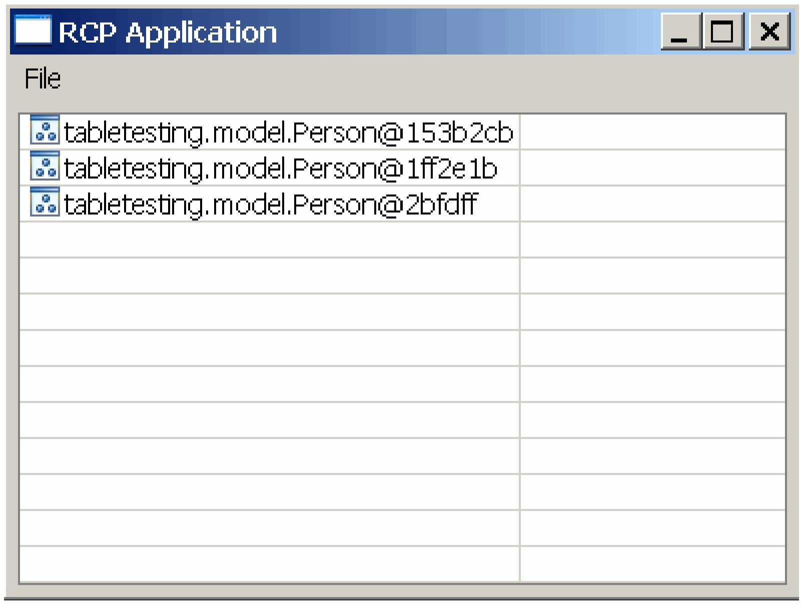
We have to change the method 'createPartControl' of class 'View'. Assign now your own content provider to your viewer. This way your data will be displayed in the viewer.

Tip

The content is set but table don't shows the fields of person, e.g. name, gender, etc., because we have to build our own label provider first. The standard label provider uses the method toString() for the content.

```
public void createPartControl(Composite parent) {
        viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL
                        | SWT.V_SCROLL | SWT.FULL_SELECTION);
        viewer.setContentProvider(new MyContentProvider());
        viewer.setLabelProvider(new ViewLabelProvider());
    viewer.getTable().setLinesVisible(true);
    viewer.setInput(getViewSite());
}
```

You can now delete the inter class "ViewContentProvider" of View. We don't need it any more. If you run the application the result should look similar to the following.



tabletesting.model.Person@153b2cb
tabletesting.model.Person@1ff2e1b
tabletesting.model.Person@2bfdff

6. LabelProvider for the columns

6.1. Build your Column LabelProvider

The class ‘View’ already contains a Label Provider from the example. Currently only the toString() method of the object person is used. We will replace this viewer label provider with label providers for the columns.

First create the following class "CheckBoxLabelProvider" which will be used to display a checkbox in the table. In this class we generate the image for the checkbox and we set the right status.

```
package tabletesting.provider;

import org.eclipse.jface.resource.JFaceResources;
import org.eclipse.jface.viewers.ColumnLabelProvider;
import org.eclipse.jface.viewers.ColumnViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.GC;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Shell;

import tabletesting.model.Person;

public class MyCheckBoxLabelProvider extends ColumnLabelProvider {

        private static final String CHECKED_KEY = "CHECKED";
        private static final String UNCHECK_KEY = "UNCHECKED";

        public MyCheckBoxLabelProvider(ColumnViewer viewer) {
                System.out.println("Hello");
                if (JFaceResources.getImageRegistry().getDescriptor(CHECKED_KEY) == null) {
                        JFaceResources.getImageRegistry().put(UNCHECK_KEY,
                                        makeShot(viewer.getControl().getShell(), false));
                        JFaceResources.getImageRegistry().put(CHECKED_KEY,
                                        makeShot(viewer.getControl().getShell(), true));
                }
        }

        private Image makeShot(Shell shell, boolean type) {
                Shell s = new Shell(shell, SWT.NO_TRIM);
                Button b = new Button(s, SWT.CHECK);
                b.setSelection(type);
                Point bsize = b.computeSize(SWT.DEFAULT, SWT.DEFAULT);
                b.setSize(bsize);
                b.setLocation(0, 0);
                s.setSize(bsize);
                s.open();
                GC gc = new GC(b);
                Image image = new Image(shell.getDisplay(), bsize.x, bsize.y);
                gc.copyArea(image, 0, 0);
                gc.dispose();
                s.close();
                return image;
        }

        public Image getImage(Object element) {
                if (isChecked(element)) {
                        return JFaceResources.getImageRegistry().getDescriptor(CHECKED_KEY)
                                        .createImage();
                } else {
                        return JFaceResources.getImageRegistry().getDescriptor(UNCHECK_KEY)
                                        .createImage();
                }
        }

        // protected abstract boolean isChecked(Object element);

        @Override
        public String getText(Object element) {
                return "";
        }

        protected boolean isChecked(Object element) {
                return ((Person) element).isMarried();
        }
}
```

Now create class "MyColumnLabelProvider" which will create the columns and the individual column label providers.

```
package tabletesting.provider;

import org.eclipse.jface.viewers.ColumnLabelProvider;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.jface.viewers.TableViewerColumn;
import org.eclipse.swt.SWT;

import tabletesting.model.Person;

public class MyColumnLabelProvider {
        public void createColumns(TableViewer viewer) {
                String[] titles = { "Firstname", "Last name", "Gender", "is Married?" };
                int[] bounds = { 100, 100, 100, 100 };

                // Column 0: First Name
                int i = 0;
                TableViewerColumn column = new TableViewerColumn(viewer, SWT.NONE);
                column.getColumn().setWidth(bounds[i]);
                column.getColumn().setText(titles[i]);
                column.getColumn().setMoveable(true);
                column.setLabelProvider(new ColumnLabelProvider() {

                        public String getText(Object element) {
                                return ((Person) element).getFirstName();
                        }
                });

                // Column 1: Last Name
                i++;
                column = new TableViewerColumn(viewer, SWT.NONE);
                column.getColumn().setWidth(bounds[i]);
                column.getColumn().setText(titles[i]);
                column.getColumn().setMoveable(true);
```

```
		column.setLabelProvider(new ColumnLabelProvider() {

			public String getText(Object element) {
				return ((Person) element).getLastName();
			}
		});

		// Column 2: Gender
		i++;
		column = new TableViewerColumn(viewer, SWT.NONE);
		column.getColumn().setWidth(bounds[i]);
		column.getColumn().setText(titles[i]);
		column.getColumn().setMoveable(true);
		column.setLabelProvider(new ColumnLabelProvider() {

			public String getText(Object element) {
				return ((Person) element).getGender();
			}
		});

		// Column 3: Married
		i++;
		column = new TableViewerColumn(viewer, SWT.NONE);
		column.getColumn().setWidth(bounds[i]);
		column.getColumn().setText(titles[i]);
		column.getColumn().setMoveable(true);
		column.setLabelProvider(new MyCheckBoxLabelProvider(viewer));

	}
}
```

6.2. Use your Column LabelProvider

Change the ‘View’-Class

We have to change the method ‘createPartControl’ of class ‘View’ again because we want to register our own Label Provider to show the content right in the table.

```
public void createPartControl(Composite parent) {
	viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL
			| SWT.V_SCROLL | SWT.FULL_SELECTION);
	viewer.setContentProvider(new MyContentProvider());
	MyColumnLabelProvider columnsLabels = new MyColumnLabelProvider();
	columnsLabels.createColumns(viewer);
	// Set the header to visible
	viewer.getTable().setHeaderVisible(true);
	// Set the line of the table visible
	viewer.getTable().setLinesVisible(true);
	viewer.setInput(getViewSite());
}
```

Delete the internal class ‘ViewLabelProvider’ of class ‘View’. We don’t need it any more. Run your application and enjoy that your data is correctly displayed.

7. Create Cell Editors

A cell editor allows to define how the user can edit certain cells of a table. We will define the editors for our model and table.

The ‘first name’ and ‘last name’ column will be editable by a textfield. The ‘gender’ column will be editable by a drop down list and the ‘is married?’ column will be editable by a checkbox.

Create a new class with the name ‘MyEditingSupport’ in package tabletesting.provider.

```
package tabletesting.provider;

import org.eclipse.jface.viewers.CellEditor;
import org.eclipse.jface.viewers.CheckboxCellEditor;
import org.eclipse.jface.viewers.ColumnViewer;
import org.eclipse.jface.viewers.ComboBoxCellEditor;
import org.eclipse.jface.viewers.EditingSupport;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.jface.viewers.TextCellEditor;
import org.eclipse.swt.SWT;

import tabletesting.model.Person;

public class MyEditingSupport extends EditingSupport {

        private CellEditor editor;
        private int column;

        public MyEditingSupport(ColumnViewer viewer, int column) {
                super(viewer);

                String[] gender = new String[2];
                gender[0] = "male";
                gender[1] = "female";

                // Create the correct editor based on the column index
                switch (column) {
                case 2:
                        editor = new ComboBoxCellEditor(((TableViewer) viewer).getTable(),
                                        gender);
                        break;
                case 3:
                        editor = new CheckboxCellEditor(null, SWT.CHECK);
                        break;
                default:
                        editor = new TextCellEditor(((TableViewer) viewer).getTable());
                }
                this.column = column;
        }

        @Override
        protected boolean canEdit(Object element) {
                return true;
        }

        @Override
        protected CellEditor getCellEditor(Object element) {
                return editor;
        }

        @Override
        protected Object getValue(Object element) {
                Person person = (Person) element;

                switch (this.column) {
                case 0:
                        return person.getFirstName();
                case 1:
                        return person.getLastName();
                case 2:
                        if (person.getGender().equals("male")) {
                                return 0;
                        } else {
                                return 1;
                        }
                case 3:
                        return person.isMarried();
                default:
                        break;
                }
```

```
			return null;
		}

		@Override
		protected void setValue(Object element, Object value) {
			Person pers = (Person) element;

			switch (this.column) {
			case 0:
				pers.setFirstName(String.valueOf(value));
				break;
			case 1:
				pers.setLastName(String.valueOf(value));
				break;
			case 2:
				if (((Integer) value) == 0) {
					pers.setGender("male");
				} else {
					pers.setGender("female");
				}
				break;
			case 3:
				pers.setMarried((Boolean) value);
				break;
			default:
				break;
			}

			getViewer().update(element, null);
		}

}
```

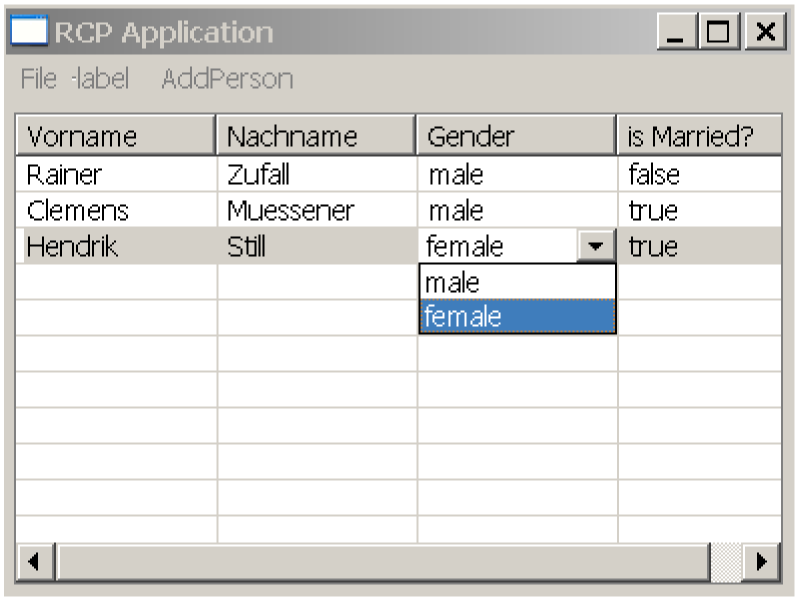
Explanation of the the main methods of ‘MyEditingSupport’:

- ‘getCellEditor’ : In this method you return the celleditor you want to use (e.g. Texteditor). I work with a ‘switch-case-command’ to split my columns and give every column its own celleditor.
- ‘setValue’: Receives the new value the user gives. Here you have to set the new value to the object which is given, too.
- ‘getValue’: Receives the object which was changed and returns the value for the table. Here you have to return the new value of the object.

Now assign the editors in your column label provider class "MyColumnLabelProvider".

```
package tabletesting.provider;

import org.eclipse.jface.viewers.ColumnLabelProvider;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.jface.viewers.TableViewerColumn;
import org.eclipse.swt.SWT;

import tabletesting.model.Person;

public class MyColumnLabelProvider {
		public void createColumns(TableViewer viewer) {
			String[] titles = { "Firstname", "Last name", "Gender", "is Married?" };
			int[] bounds = { 100, 100, 100, 100 };

			// Column 0: First Name
			int i = 0;
			TableViewerColumn column = new TableViewerColumn(viewer, SWT.NONE);
			column.getColumn().setWidth(bounds[i]);
			column.getColumn().setText(titles[i]);
			column.getColumn().setMoveable(true);
			column.setLabelProvider(new ColumnLabelProvider() {
				public String getText(Object element) {
					return ((Person) element).getFirstName();
				}
			});
			column.setEditingSupport(new MyEditingSupport(viewer, i));

			// Column 1: Last Name
			i++;
			column = new TableViewerColumn(viewer, SWT.NONE);
			column.getColumn().setWidth(bounds[i]);
			column.getColumn().setText(titles[i]);
			column.getColumn().setMoveable(true);
			column.setLabelProvider(new ColumnLabelProvider() {

				public String getText(Object element) {
					return ((Person) element).getLastName();
				}
			});
			column.setEditingSupport(new MyEditingSupport(viewer, i));

			// Column 2: Gender
			i++;
			column = new TableViewerColumn(viewer, SWT.NONE);
			column.getColumn().setWidth(bounds[i]);
			column.getColumn().setText(titles[i]);
			column.getColumn().setMoveable(true);
			column.setLabelProvider(new ColumnLabelProvider() {

				public String getText(Object element) {
					return ((Person) element).getGender();
				}
			});
			column.setEditingSupport(new MyEditingSupport(viewer, i));

			// Column 3: Married
			i++;
			column = new TableViewerColumn(viewer, SWT.NONE);
			column.getColumn().setWidth(bounds[i]);
			column.getColumn().setText(titles[i]);
			column.getColumn().setMoveable(true);
			column.setLabelProvider(new MyCheckBoxLabelProvider(viewer));
			column.setEditingSupport(new MyEditingSupport(viewer, i));

	}
}
```

Run now your application. You should now be able to modify the content of the JFace table.



Vorname	Nachname	Gender	is Married?
Rainer	Zufall	male	false
Clemens	Muessener	male	true
Hendrik	Still	female	true

8. Print the model content

This chapter add the functionality to print the content of the domain model to the console. This way it is possible to verify that the changes on the JFace table are updating the model.

Create one actions as extensions to org.eclipse.ui.actionSets. Create the action "TableTesting.PrintModel" with class 'tabletesting.actions.PrintModel'. (see See Usage of Actions (Menu and Toolbar) for details).

Make sure that the menupath is set in the extension, otherwise the action will not be visible in the application. Modify the Code of the class "tabletesting.actions.PrintModel" like this:

```
package tabletesting.actions;

import java.util.Iterator;
import java.util.List;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

import tabletesting.model.ControlPersons;
import tabletesting.model.Person;

public class PrintModel implements IWorkbenchWindowActionDelegate {

        @Override
        public void dispose() {
        }

        @Override
        public void init(IWorkbenchWindow window) {
        }

        @Override
        public void run(IAction action) {
                List<Person> persons = ControlPersons.getInstance().getPersonsList();
                for (Iterator<Person> iterator = persons.iterator(); iterator.hasNext();) {
                        Person person = (Person) iterator.next();
                        System.out.println(person.getFirstName() + " "
                                        + person.getLastName() + " " + person.getGender() + " "
                                        + String.valueOf(person.isMarried()));
                }
        }
        @Override
        public void selectionChanged(IAction action, ISelection selection) {
        }
}
```

After finishing this you will be able to print the current state of your domain model to the console via the menu.

9. Modify the model - Add / Remove a Person

This chapter shows how to add a new Person to your Table and how to remove a selected one.

You have to add a new Method to class ‘View’ to get the viewer in your actions.

```
public TableViewer getViewer()
        {
                return viewer;
        }
```

Create two actions as extensions to org.eclipse.ui.actionSets. Create the action "TableTesting.AddPerson" with class "tabletesting.actions.AddPerson" and the action "TableTesting.DeletePerson" with class "tabletesting.actions.DeletePerson". (see See Usage of Actions (Menu and Toolbar) for details).

Create a dialog to maintain the data for the additional person. Create package "tabletesting.dialog" and the following class "AddDialog".

```
package tabletesting.dialog;

import org.eclipse.jface.dialogs.IMessageProvider;
import org.eclipse.jface.dialogs.TitleAreaDialog;
import org.eclipse.jface.resource.JFaceResources;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;

import tabletesting.model.Person;

public class AddDialog extends TitleAreaDialog {

        private Text text1;
        private Text text2;
        private Person person;
        private Button button1;
        private Combo combo1;

        public Person getPerson() {
                return person;
        }

        public AddDialog(Shell parentShell) {
                super(parentShell);
                // TODO Auto-generated constructor stub
        }
```

```
	@Override
	protected Control createContents(Composite parent) {
		// TODO Auto-generated method stub
		Control contents = super.createContents(parent);
		setTitle("Add a new Person");
		setMessage("Please enter the data of the new person",
				IMessageProvider.INFORMATION);
		return contents;
	}

	@Override
	protected Control createDialogArea(Composite parent) {
		GridLayout layout = new GridLayout();
		layout.numColumns = 2;
		parent.setLayout(layout);
		Label label1 = new Label(parent, SWT.NONE);
		label1.setText("First Name");
		text1 = new Text(parent, SWT.BORDER);
		Label label2 = new Label(parent, SWT.NONE);
		label2.setText("Last Name");
		text2 = new Text(parent, SWT.BORDER);
		Label label3 = new Label(parent, SWT.NONE);
		label3.setText("Gender");
		GridData gd = new GridData(GridData.HORIZONTAL_ALIGN_END);
		gd.horizontalSpan = 2;
		combo1 = new Combo(parent, SWT.READ_ONLY);
		combo1.add("male");
		combo1.add("female");
		button1 = new Button(parent, SWT.CHECK);
		button1.setText("Is married?");
		button1.setLayoutData(gd);
		return parent;

	}

	@Override
	protected void createButtonsForButtonBar(Composite parent) {
		((GridLayout) parent.getLayout()).numColumns++;

		Button button = new Button(parent, SWT.PUSH);
		button.setText("OK");
		button.setFont(JFaceResources.getDialogFont());
		button.addSelectionListener(new SelectionAdapter() {
			public void widgetSelected(SelectionEvent e) {
				if (text1.getText().length() != 0
						&& text2.getText().length() != 0
						&& combo1.getItem(combo1.getSelectionIndex()).length() != 0
					person = new Person(text1.getText(), text2.getText(),
							button1.getSelection(), combo1.getItem(combo1
									.getSelectionIndex()));
					close();

				} else {
					setErrorMessage("Please enter all data");
				}
			}
		});
	}

}
```

Modify the Code of the class ‘AddPerson’ like this:

```
			package tabletesting.actions;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

import tabletesting.View;
import tabletesting.dialog.AddDialog;
import tabletesting.model.ControlPersons;
import tabletesting.model.Person;

public class AddPerson implements IWorkbenchWindowActionDelegate {

        private ControlPersons persons;
        private Person newPers;
        private TableViewer viewer;
        private View view;
        private IWorkbenchWindow window;

        @Override
        public void dispose() {
                this.viewer = view.getViewer();

        }

        @Override
        public void init(IWorkbenchWindow window) {
                this.window = window;
                IWorkbenchPage page = window.getActivePage();
                this.view = (View) page.findView(View.ID);
                this.viewer = this.view.getViewer();
        }

        @Override
        public void run(IAction action) {
                persons = ControlPersons.getInstance();
                AddDialog dialog = new AddDialog(window.getShell());
                dialog.open();
                if (dialog.getPerson() != null) {
                        persons.getPersonsList().add(dialog.getPerson());
                }
                this.viewer.refresh();

        }

        @Override
        public void selectionChanged(IAction action, ISelection selection) {

        }

}
```

When you finished this successful you are able to add new persons in your application. In our example the added person is hard-coded.

We will implement that a person can be deleted from the list after someone selected this person via a double click. To do this you have to add a double-click listener to the table. Within this message you will remember the selected data and a method will be available to to give this selection to the delete method.

The whole "View" class looks now like that:

```
package tabletesting;

import org.eclipse.jface.viewers.DoubleClickEvent;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.ITableLabelProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.ViewPart;

import tabletesting.model.Person;
import tabletesting.provider.MyColumnLabelProvider;
import tabletesting.provider.MyContentProvider;

public class View extends ViewPart {
        public static final String ID = "TableTesting.view";
        private TableViewer viewer;
        private Person selectedPerson;

        class ViewLabelProvider extends LabelProvider implements
                        ITableLabelProvider {
                public String getColumnText(Object obj, int index) {
                        return getText(obj);
                }

                public Image getColumnImage(Object obj, int index) {
                        return getImage(obj);
                }

                public Image getImage(Object obj) {
                        return PlatformUI.getWorkbench().getSharedImages().getImage(
                                        ISharedImages.IMG_OBJ_ELEMENT);
                }
        }

        public void createPartControl(Composite parent) {
                viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL
                                | SWT.V_SCROLL | SWT.FULL_SELECTION);
                viewer.setContentProvider(new MyContentProvider());
                MyColumnLabelProvider columnsLabels = new MyColumnLabelProvider();
                columnsLabels.createColumns(viewer);
                // Set the header to visible
                viewer.getTable().setHeaderVisible(true);
                // Set the line of the table visible
                viewer.getTable().setLinesVisible(true);
                viewer.setInput(getViewSite());
                viewer.addDoubleClickListener(new IDoubleClickListener() {
                        @Override
                        public void doubleClick(DoubleClickEvent event) {
                                IStructuredSelection sel = (IStructuredSelection) event
                                                .getSelection();
                                selectedPerson = (Person) sel.getFirstElement();
                        }
                });

        }

        /**
         * Passing the focus request to the viewer's control.
         */
        public void setFocus() {
                viewer.getControl().setFocus();
        }

        public TableViewer getViewer() {
                return viewer;
        }

        public Person getSelectedPerson() {
                return selectedPerson;
        }
```

Modify the Code of the class "TableTesting.DeletePerson" like this:

```
package tabletesting.actions;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

import tabletesting.View;
import tabletesting.model.ControlPersons;
import tabletesting.model.Person;

public class DeletePerson implements IWorkbenchWindowActionDelegate {
        private ControlPersons persons;
        private Person selectedPerson;
        private View view;
        private TableViewer viewer;

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

        }

        @Override
        public void init(IWorkbenchWindow window) {
                IWorkbenchPage page = window.getActivePage();
                view = (View) page.findView(View.ID);
                viewer = view.getViewer();

        }

        @Override
        public void run(IAction action) {
                selectedPerson = view.getSelectedPerson();
                persons = ControlPersons.getInstance();
                persons.getPersonsList().remove(selectedPerson);
                this.viewer.refresh();

        }

        @Override
        public void selectionChanged(IAction action, ISelection selection) {
        }

}
```

Please try to delete entries of the table by double-clicking a row and run "Delete Action".

10. Copy JFace viewer data to the system clipboard

This chapter shows how to copy the table data to the system clipboard via shortcut Cntr+C.

Create a new action "Copy" with the class "tabletesting.actions.Copy".

```
package tabletesting.actions;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.dnd.Clipboard;
import org.eclipse.swt.dnd.TextTransfer;
import org.eclipse.swt.dnd.Transfer;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

import tabletesting.View;
import tabletesting.model.Person;

public class Copy implements IWorkbenchWindowActionDelegate {

        private TableViewer viewer;
        private View view;
        private Clipboard cb;

        @Override
        public void dispose() {
                this.viewer = view.getViewer();

        }

        @Override
        public void init(IWorkbenchWindow window) {
                IWorkbenchPage page = window.getActivePage();
                this.view = (View) page.findView(View.ID);
                this.viewer = this.view.getViewer();

                this.cb = new Clipboard(Display.getDefault());

        }

        @Override
        public void run(IAction action) {

                IStructuredSelection selection = (IStructuredSelection) view
                                .getViewer().getSelection();
                List<Person> persons = new ArrayList<Person>();

                for (Iterator<Person> iterator = selection.iterator(); iterator
                                .hasNext();) {
                        Person person = (Person) iterator.next();
                        persons.add(person);

                }

                String saveString = "";
                for (Person person : persons) {
                        saveString += personToString(person);
                }
                TextTransfer textTransfer = TextTransfer.getInstance();
                cb.setContents(new Object[] { saveString },
                                new Transfer[] { textTransfer });
        }

        @Override
        public void selectionChanged(IAction action, ISelection selection) {
                // TODO Auto-generated method stub

        }

        private String personToString(Person person) {
                return person.getFirstName() + "\t" + person.getLastName() + "\t"
                                + person.getGender() + "\t" + person.isMarried()
                                + System.getProperty("line.separator");
        }

}
```

11. Multiple Editor

This chapter shows how to implement the functionality to select several entry of a table and edit common attributes via the right mouse click. Only if the element are the same they can get changed.

Create a new dialog "tabletesting.dialog.EditDialog"

```
package tabletesting.dialog;

import org.eclipse.jface.dialogs.IMessageProvider;
import org.eclipse.jface.dialogs.TitleAreaDialog;
import org.eclipse.jface.resource.JFaceResources;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;

public class EditDialog extends TitleAreaDialog {

        private Text text1;
        private Text text2;
        private Button button1;
        private Combo combo1;
        private String firstName;
        private String lastName;
        private String gender;
        private int isMarried;

        public EditDialog(Shell parentShell) {
                super(parentShell);
        }

        @Override
        protected Control createContents(Composite parent) {
                Control contents = super.createContents(parent);
                setTitle("Edit a Person");
                setMessage("Please enter the new data for the person",
                                IMessageProvider.INFORMATION);
                return contents;
        }

        @Override
        protected Control createDialogArea(Composite parent) {
                GridLayout layout = new GridLayout();
                layout.numColumns = 2;
                parent.setLayout(layout);
                Label label1 = new Label(parent, SWT.NONE);
                label1.setText("First Name");
                text1 = new Text(parent, SWT.BORDER);
                if (firstName != null) {
                        text1.setText(firstName);
                } else {
                        text1.setEnabled(false);
                }
                Label label2 = new Label(parent, SWT.NONE);
                label2.setText("Last Name");
                text2 = new Text(parent, SWT.BORDER);
                if (lastName != null) {
                        text2.setText(lastName);
                } else {
                        text2.setEnabled(false);
```

```
		}
		Label label3 = new Label(parent, SWT.NONE);
		label3.setText("Gender");
		GridData gd = new GridData(GridData.HORIZONTAL_ALIGN_END);
		gd.horizontalAlignment = GridData.FILL;
		gd.horizontalSpan = 2;
		combo1 = new Combo(parent, SWT.READ_ONLY);
		combo1.add("male");
		combo1.add("female");
		if (gender == null) {
			combo1.setEnabled(false);
		} else {
			combo1.setText(gender);
		}

		button1 = new Button(parent, SWT.CHECK);
		button1.setText("Is married?");
		button1.setLayoutData(gd);
		if (isMarried != 0) {
			button1.setSelection((isMarried == 1));
		} else {
			button1.setEnabled(false);
		}
		return parent;

	}

	public void setPersonData(String firstName, String lastName, String gender,
			int isMarried) {
		this.firstName = firstName;
		this.lastName = lastName;
		this.gender = gender;
		this.isMarried = isMarried;

	}

	public String getFirstName() {
		return firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public String getGender() {
		return gender;
	}

	public int getIsMarried() {
		return isMarried;
	}

	@Override
	protected void createButtonsForButtonBar(Composite parent) {
		((GridLayout) parent.getLayout()).numColumns++;
		Button button = new Button(parent, SWT.PUSH);
		button.setText("OK");
		button.setFont(JFaceResources.getDialogFont());
		button.addSelectionListener(new SelectionAdapter() {
			public void widgetSelected(SelectionEvent e) {
				firstName = text1.getText();
				lastName = text2.getText();
				if (gender != null)
					gender = combo1.getItem(combo1.getSelectionIndex());
				isMarried = (button1.getSelection() ? 1 : 2);

				close();
			}
		});
	}
```

```
}
```

Create a new action Edit with the class tabletesting.actions.Edit

```
package tabletesting.actions;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

import tabletesting.View;
import tabletesting.dialog.EditDialog;
import tabletesting.model.Person;

public class Edit implements IWorkbenchWindowActionDelegate {

        private TableViewer viewer;
        private View view;
        private IWorkbenchWindow window;

        @Override
        public void dispose() {
                this.viewer = view.getViewer();
        }

        @Override
        public void init(IWorkbenchWindow window) {
                this.window = window;
                IWorkbenchPage page = window.getActivePage();
                this.view = (View) page.findView(View.ID);
                this.viewer = this.view.getViewer();

        }

        @Override
        public void run(IAction action) {

                IStructuredSelection selection = (IStructuredSelection) view
                                .getViewer().getSelection();
                List<Person> persons = new ArrayList<Person>();

                for (Iterator<Person> iterator = selection.iterator(); iterator
                                .hasNext();) {
                        Person person = (Person) iterator.next();
                        persons.add(person);

                }
                String tempFirstName = null, tempLastName = null, tempGender = null;
                int tempIsMarried = 0;
                int i = 0;

                for (Person person : persons) {
                        i++;
                        if (i == 1) {
                                tempFirstName = person.getFirstName();
                                tempLastName = person.getLastName();
                                tempGender = person.getGender();
                                tempIsMarried = person.isMarried() ? 1 : 2; // if married 1 if
                                // not 2
                        } else {
                                if (!person.getFirstName().equals(tempFirstName)) {
                                        tempFirstName = null;
                                }
                                if (!person.getLastName().equals(tempLastName)) {
                                        tempLastName = null;
                                }
                                if (!person.getGender().equals(tempGender)) {
                                        tempGender = null;
                                }
                                if ((person.isMarried() ? 1 : 2) != tempIsMarried) {
                                        tempIsMarried = 0;
                                }
                        }
                }

                EditDialog dialog = new EditDialog(window.getShell());
                dialog.setPersonData(tempFirstName, tempLastName, tempGender,
                                tempIsMarried);
                dialog.open();

                for (Person person : persons) {
                        if (tempFirstName != null) {
                                person.setFirstName(dialog.getFirstName());
                        }
                        if (tempLastName != null) {
                                person.setLastName(dialog.getLastName());
                        }
                        if (tempGender != null) {
                                person.setGender(dialog.getGender());
                        }
                        if (tempIsMarried != 0) {
                                person.setMarried((dialog.getIsMarried() == 1 ? true : false));
                        }
                }
                this.viewer.refresh();

        }

        @Override
        public void selectionChanged(IAction action, ISelection selection) {
        }

}
```

12. Thank you

Thank you for practicing with this tutorial.

13. Links and Literature

13.1. Source Code

http://www.vogella.de/code/codeeclipse.html Source Code of Examples

13.2. JFace Resources

http://www.eclipse.org/articles/Article-Table-viewer/table_viewer.html Building and delivering a table editor with SWT/JFace

http://wiki.eclipse.org/index.php/JFaceSnippets JFace snippets, e.g. small code examples

13.3. Other Eclipse resources

http://www.eclipse.org Eclipse.org Homepage

http://www.vogella.de Articles about Java and Eclipse

http://www.vogella.de/articles/Eclipse/article.html Using Eclipse as IDE - Tutorial

http://www.vogella.de/articles/RichClientPlatform/article.html Eclipse Rich Client Platform Tutorial - A Hands-on-Guide

http://www.vogella.de/articles/EclipseDataBinding/article.html Eclipse Databinding

http://www.vogella.de/articles/EclipseMicrosoftIntegration/article.html Eclipse Microsoft Integration - Tutorial

http://www.vogella.de/articles/EclipseCodeAccess/article.html A guide to access the Eclipse Sources

http://www.vogella.de/articles/EclipseJFreeChart/article.html Using JFreeChart with an Eclipse RCP application.

http://www.vogella.de/articles/EclipseJFaceTable/article.html Using the JFace 3.3 API to create a table in an Eclipse RCP application.

http://www.vogella.de/articles/EclipseDataToolsPlatform/article.html Database Access with the Eclipse Data Tool Platform (DTP) - Tutorial

http://www.vogella.de/articles/EclipseTPTP/article.html Tutorial for testing Web applications with the Eclipse Test and Performance Tools Platform (TPTP)

http://www.vogella.de/articles/EclipseWTP/article.html Using Eclipse WTP to build servlets, manage database access for webapplication and to handle webservices.

http://www.vogella.de/articles/EclipseEMF/article.html Building a HTML Website with the Eclipse Modeling Framework (EMF) and Java Emitter Template (JET) - Tutorial

http://www.vogella.de/articles/EclipseBIRT/article.html Using the Eclipse Business Intelligence and Reporting Tools (BIRT) for creating interactive reports with Plain Old Java Objects.

Bookmark this page:

Digg this site del.icio.us

This site has been accessed 20358 times.