

Goals:

- Learn how to use Designer as a tool for rapid prototyping of the UI
- Know the Designer-to-IDE workflow
- Learn about many time saving features (Quick Value Set, Transform, Duplicate, Auto Columns, etc.)
- How to write event handlers and add behavior to the exported component classes

Assumption:

- It is assumed that you have set up XAMPP. Run XAMPP.
- \$XAMPP\$ refers to the XAMPP installation directory (e.g., c:\xampp).
- It is assumed that Ext Designer is installed.
- You have downloaded and unzipped moviestore.zip file in \$XAMPP\$\htdocs. You should have \$XAMPP\$\htdocs\exttraining\labs\moviestore\moviestore.xds file and many other files and subfolders.
- IDE of your choice is installed. You will use IDE, or text editors, to view and edit all file types other than .xds file.

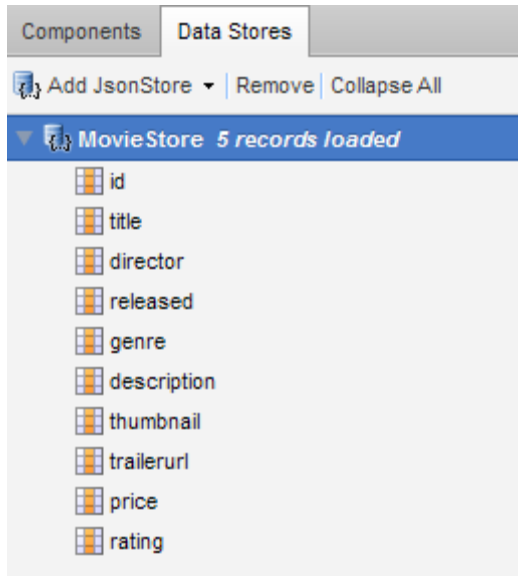
Step 1: Setup a Data Store

Goal:

Configure and display data from a Data Store in Ext Designer.

Hints:

- Open the `movies.json` file in an editor of your choice. Review the data content and its structure.
- Launch Ext Designer and open the ExDesigner project - `$XAMPP$\htdocs\exttraining\labs\moviestore\moviestore.xds`
- Add a new JsonStore and set the following config properties:
 - `jsclass:MovieStore`
 - `StoreId:MovieStore`
 - `url:db/movies.json`
 - Check `autoLoad`
 - `root:movies` (note that our JSON has `movies` member)
 - Add fields to the `MovieStore` that maps to the fields of `movies` content in JSON. Set proper field name, mapping, and type for each field. For the `released` field, select type as `date` and the `dateFormat` is 'Y-m-d'. You can add up to 5 fields at a time.



- Validate setup by loading the content from the data store. Right click on the MovieStore and select Load Data. If the data does not load, verify that properties are correct in Edit > Edit Preferences. The 'url' value from the Edit Preferences is prefixed to the data store url config property before retrieving the data.

Step 2: Display movie data in a Grid

Goals:

- Create GridPanel UI
- Read and display Movie data

Hints:

- **Create and configure the GridPanel component**
 - Drag and drop GridPanel from the Toolbox on the Design Canvas. Set its configuration to
 - height:500
 - Remove width and title.
 - jsClass:MovieGrid
 - userXType:moviegrid. By setting the userXType, you will be able to access the component with that name later on.
 - store:MovieStore.
 - Update columns using the *Auto Columns* feature. Right click on the grid and select Auto columns.
 - Hide the id column. Select id column under MovieGrid in Components Tab and check hidden property.
 - In Components tab, move the thumbnail column to **SECOND** in the order (below id column) and set the header property to Movie. You can either set

it in the Component Config, or by double clicking directly on the column header of the grid on the Design tab.

- Optionally, hide the trailerurl column, the user does not need to see it.
- Select the grid and test it in *Preview* mode. Test the many great features such as sorting and column hiding that you get automatically.



Movies	title	director	released	genre	description	trailerurl	price	rating
inception.jpg	Inception	Christopher Nolan	07/16/2010	Action	In a world wher...	http://www.yout...	15.95	9.00
godfather.jpg	The Godfather	Francis Ford Co...	03/24/1972	Action	The aging patria...	http://www.yout...	16.95	8.00
americanbeauty...	American Beauty	Sam Mendes	10/01/1999	Romance	Beauty is in the ...	http://www.yout...	12.95	7.00
toystory3.jpg	Toy Story 3	Lee Unkrich	06/10/2010	Animation	The toys are mis...	http://www.yout...	13.95	9.00
darkknight.jpg	The Dark Knight	Christopher Nolan	07/18/2008	Thriller	Batman, Gordon...	http://www.yout...	14.95	9.00

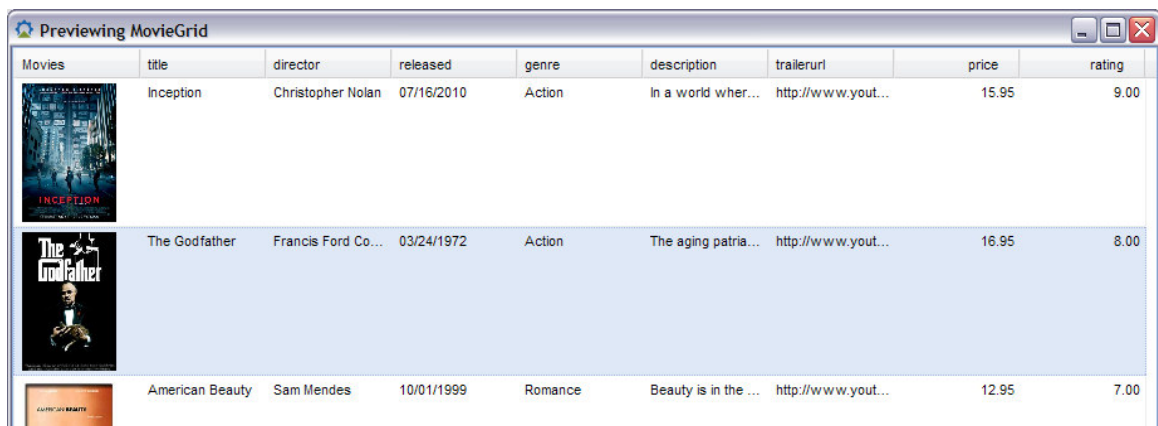
➤ Render image for Movie column using a template


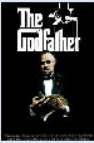

- To show movie image, transform the `Movie` column (mapped to `thumbnail`) to a `TemplateColumn`. Right click on the `thumbnail` column in Components panel and select `Transform > TemplateColumn`. This will give you a `'tpl'` property in the Component Config for that column. Set the value of `tpl` as follows.

```

```

- Test the grid in *Preview* mode.



Movies	title	director	released	genre	description	trailerurl	price	rating
	Inception	Christopher Nolan	07/16/2010	Action	In a world wher...	http://www.yout...	15.95	9.00
	The Godfather	Francis Ford Co...	03/24/1972	Action	The aging patria...	http://www.yout...	16.95	8.00
	American Beauty	Sam Mendes	10/01/1999	Romance	Beauty is in the ...	http://www.yout...	12.95	7.00

Step 3. Show a Movie Trailer

Goals:

- Display a movie trailer for a selected movie in a Window UI

Hints:

➤ Create and configure TrailerWindow component

- Drag and drop a Window from the ToolBBox to the Design Canvas. Set its config as follows:
 - Check auto-scroll.
 - title:Movie Trailer
 - jsClass:TrailerWindow.
 - Set layout type fit.
 - Remove the width and height configurations.
- To play movie, set following template (tpl configuration):

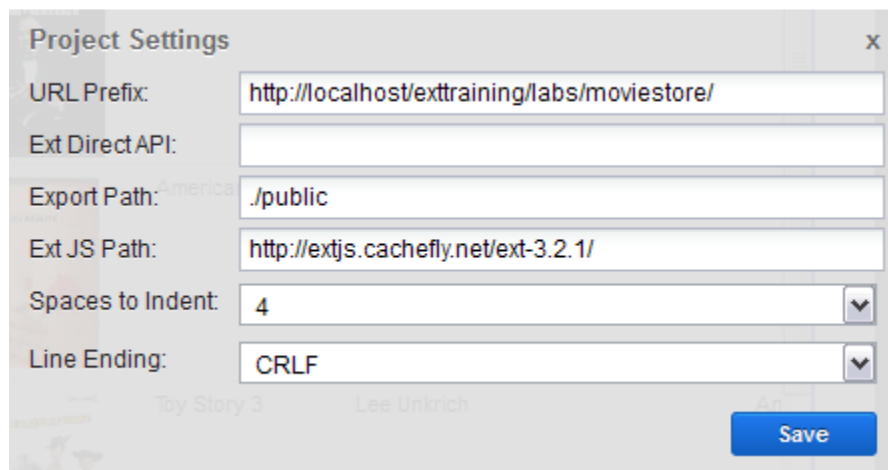
```
<object width="425" height="353"><param name="movie" value="{trailerurl}"></param><param name="wmode" value="transparent"></param><embed src="{trailerurl}" type="application/x-shockwave-flash" wmode="transparent" width="425" height="353"></embed></object>
```

➤ Create event handler to open the TrailerWindow

- To show the hand cursor for mouse over on the Movie column of the MovieGrid component, set the following css property:

```
cursor: pointer;
```

- Save and Export the project. The *Project Settings* dialog describes the export path for this project. You can access this dialog via *Edit > Edit Preferences* menu.



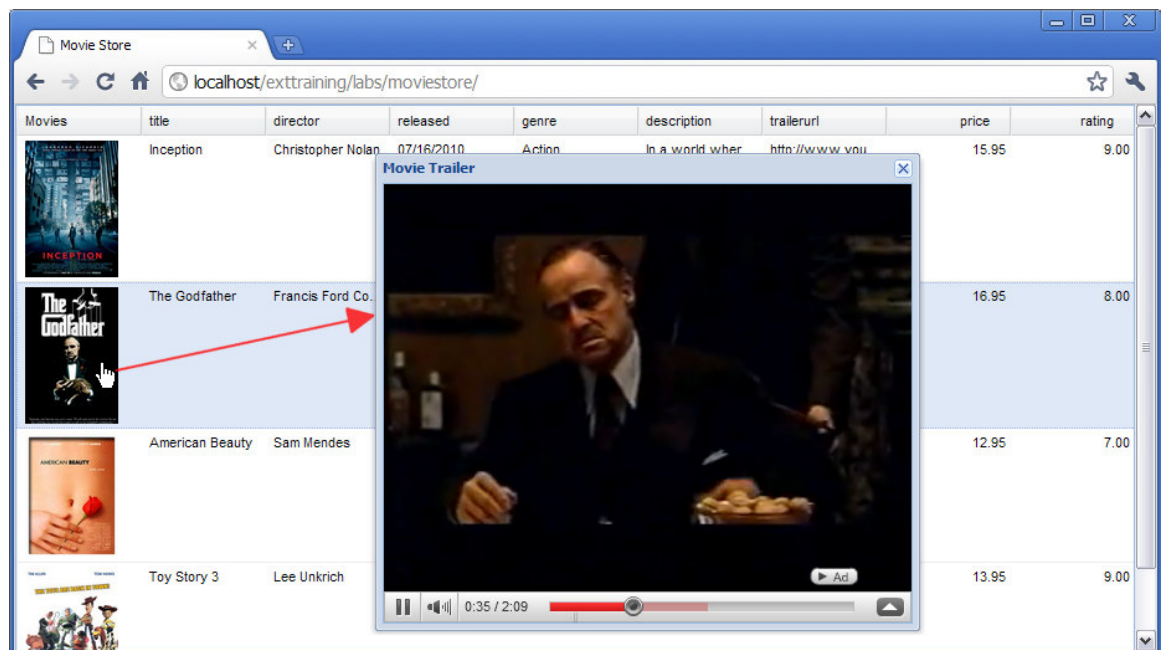
- Review the exported artifacts under the \$XAMPP\$\htdocs\exttraining\labs\moviestore\public folder.
- In an IDE of your choice, edit *MovieGrid.js* to add a handler for `cellclick` event.

```
MovieGrid = Ext.extend(MovieGridUi, {
    initComponents: function() {
        MovieGrid.superclass.initComponents.call(this);

        this.on('cellclick', this.onCellClick, this);
    },

    onCellClick: function(grid, rid, cid) {
        //console.log("inside onCellClick: ", rid, cid);
        if (cid == 1) {
            var trailer = new TrailerWindow();
            trailer.show();
            var rec = grid.getStore().getAt(rid);
            trailer.update(rec.data);
        }
    }
});
Ext.reg('moviegrid', MovieGrid);
```

- In the `$XAMPP$\htdocs\exttraining\labs\moviestore\index.html` file, uncomment the section which refers to .js files for MovieStore, MovieGrid and TrailerWindow components. The `application.js` declared the start component.
- Test your solution by loading <http://localhost/exttraining/labs/moviestore> page in a browser.



Step 4: Add new Movie data

Goals:

In this step, you will learn to add new records to a data store. Remember that the data store is only on the client side in the browser memory. A php is provided to upload the thumbnail image for the movie to the server.

- Create a form for entering Movie data
- Add a toolbar button to open the form

Hints:

➤ Create a new Movie entry form

- Add a new Window container with component configuration `jsClass: AddMovieWindow` and `layout: fit`. Set the title, width and height preferences.
- Add a `FormPanel` and configure it with the URL `db/saveimage.php`. Set some padding.
- Make the `AddMovieWindow` container a modal dialog.
- Add a toolbar and position at the bottom.
- Add `Save` and `Cancel` buttons to the toolbar. Also, set `autoRef` property.
- To enable form validation, configure the Form with `monitorValid` option and the `Save` button with `formBind` option.
- Add set of form fields for new Movie entry. Also, make sure to set `id` property for each name field.
- Transform the `Description` field to `TextArea`.
- Transform the `Price` field to `NumberField`.
- Transform the `Rating` field to `NumberField`.
- Transform the `Thumbnail` field to `Hidden` and set the `inputType` to `file`.
- Configure the form with a default option where every field requires an input [`Defaults={"allowBlank": false}`]. For the form fields (like `title`, `director`, `genre`, and `price`), specify that it requires input parameter (`allowBlank: false`).

➤ Add a toolbar button to launch the form

- In order to launch the form, add a toolbar (along with the `New Movie` button) at the top of the `MovieGrid` panel using drag and drop.
- Configure `btnNew` value for `autoRef` property of `New Movie` button.
- Export the project and review exported artifacts.
- In `MovieGrid.js`, add a function to open the `Add new Movie` data form when someone clicks on the `New Movie` button.

```
MovieGrid = Ext.extend(MovieGridUi, {
```

```
initComponent: function() {
    MovieGrid.superclass.initComponent.call(this);

    this.on('celldclick', this.onCellClick, this);

    this.btnNew.on('click', this.onBtnNewClick, this);

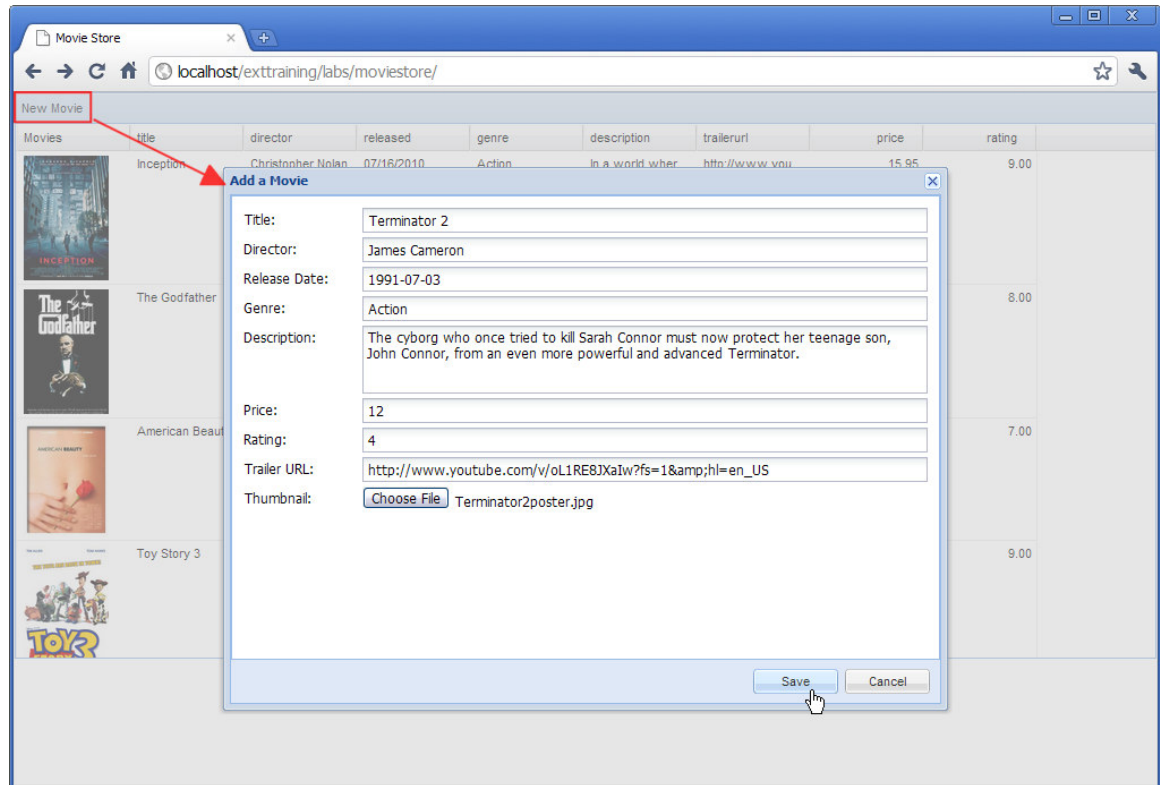
},

onCellClick: function(grid, rid, cid) {

    if (cid ==1) {
        var trailer = new TrailerWindow();
        trailer.show();
        var rec = grid.getStore().getAt(rid);
        trailer.update(rec.data);
    }
},

onBtnNewClick: function() {
    var win = new AddMovieWindow();
    win.show();
}
});
Ext.reg('moviegrid', MovieGrid);
```

- In the *index.html* file, uncomment the section, which refers to .js files for AddMovieWindow components.
- Test your solution.



➤ **Add handler to insert a new movie data**

- To access the form reference, configure the Add new Movie form with `autoRef=form` option.
- Since the form allows one to choose an image file, enable the `fileUpload` option on the form.
- In the `AddMovieWindow.js`, add following functions
 - A function to close the window on click event of the Cancel button.
 - A function to submit the form data on click event of the Save button.
 - If the request is handled successfully, then close the form and refresh the MovieGrid by reloading the MovieStore.
 - In case of failure, notify user about the failure.

```
AddMovieWindow = Ext.extend(AddMovieWindowUi, {
    initComponents: function() {
        AddMovieWindow.superclass.initComponent.call(this);

        this.btnCancel.on('click', this.close, this);
        this.btnSave.on('click', this.onBtnSaveClick, this);
    },

    onBtnSaveClick: function() {
```



```
var rec = this.record;

var title = Ext.getCmp('title').getValue();
var director = Ext.getCmp('director').getValue();
var released = Ext.getCmp('released').getValue();
var genre = Ext.getCmp('genre').getValue();
var description = Ext.getCmp('description').getValue();

var thumbnailFilePath = Ext.getCmp('thumbnail').getValue();
var lastPathDelimiter = thumbnailFilePath.lastIndexOf("\\");
var thumbnail = thumbnailFilePath.substring(lastPathDelimiter+1);

var price = Ext.getCmp('price').getValue();
var rating = Ext.getCmp('rating').getValue();

var successCb = function(f, a) {
    var store = Ext.StoreMgr.lookup('MovieStore');
    store.add(new store.recordType({
        id: store.getCount(),
        title: title,
        director: director,
        released: released,
        genre: genre,
        description: description,
        thumbnail: thumbnail,
        trailerurl: "",
        price: price,
        rating: rating
    }));

    this.close();
};

if (rec) {
    this.form.form.submit({
        method: 'PUT',
        url: '/saveimage.php',
        success: successCb,
        failure: function() {
            alert('uhh something went wrong - editing a movie.');
```

```
        failure: function() {
            alert('uhh something went wrong - adding new movie.');
```

- Export the project and test your solution. You can test by adding data for some movies. Observe that the data is immediately shown in the grid. However, movie data is not saved on the server except for the image. If you refresh the page on the browser then you will notice that it shows the original set of movies from the JSON file.

Step 5: Remove the selected Movie data

Goals:

- Enable/disable the `Delete Movie` button based on the row selection
- Delete a specific Movie entry

Hints:

➤ **Add Delete Movie button**

- Add a `Delete Movie` button to the `MovieGrid` toolbar (after the `New Movie` button).
- Configure `btnDelete` value for `autoRef` property of `Delete Movie` button.
- Since the delete movie action depends on the selection of a movie, make the `Delete Movie` disabled by default.

➤ **Enable/disable the Delete Movie button based on selection**

- Add `RowSelectionModel` to `MovieGrid` component and choose the `singleSelect` configuration for enabling the single selection.
- In `MovieGrid.js`, add function to handle selection change.
 - If a movie entry has been selected then enable the `Delete Movie` toolbar buttons.
 - If nothing (movie entry) has been selected then disable the `Delete Movie` toolbar buttons.

```
MovieGrid = Ext.extend(MovieGridUi, {
    initComponents: function() {
        MovieGrid.superclass.initComponent.call(this);

        this.on('cellclick', this.onCellClick, this);
```

```
this.btnNew.on('click', this.onBtnNewClick, this);

this.getSelectionModel().on('selectionchange',
    this.onSelChange, this);

},

onCellClick: function(grid, rid, cid) {

    if (cid == 1) {
        var trailer = new TrailerWindow();
        trailer.show();
        var rec = grid.getStore().getAt(rid);
        trailer.update(rec.data);
    }
},

onBtnNewClick: function() {
    var win = new AddMovieWindow();
    win.show();
},

onSelChange: function(sm) {
    var rec = sm.getSelected();
    this.btnDelete.setDisabled(!rec);
}
});
Ext.reg('moviegrid', MovieGrid);
```

- Export the project and test your solution.

➤ **Remove a selected movie**

- In `MovieGrid.js`, add function to delete a specific movie entry on click event of the `Delete Movie` toolbar button. The function shall
 - First prompt a confirmation dialog for removing the selected movie.
 - Remove the currently selected `Movie` entry from the `MovieStore`, if user confirms, otherwise ignores the request.

```
MovieGrid = Ext.extend(MovieGridUi, {
    initComponents: function() {
        MovieGrid.superclass.initComponent.call(this);

        this.on('celldclick', this.onCellClick, this);

        this.btnNew.on('click', this.onBtnNewClick, this);
```

```
        this.getSelectionModel().on('selectionchange', this.onSelChange, this);

        this.btnDelete.on('click', this.onBtnDeleteClick, this);

    },

    onCellClick: function(grid, rid, cid) {

        if (cid ==1) {
            var trailer = new TrailerWindow();
            trailer.show();
            var rec = grid.getStore().getAt(rid);
            trailer.update(rec.data);
        }
    },

    onBtnNewClick: function() {
        var win = new AddMovieWindow();
        win.show();
    },

    onSelChange: function(sm) {
        var rec = sm.getSelected();

        this.btnDelete.setDisabled(!rec);
    },

    onBtnDeleteClick: function() {

        var rec = this.getSelectionModel().getSelected();
        var dbstore = this.store;

        Ext.Msg.show({
            title: 'Confirm - Delete Movie?',
            buttons: Ext.MessageBox.YESNO,
            msg: 'Do you want to remove the movie - ' + rec.data.title + '?',
            fn: function(btn) {
                if(btn == 'yes') {
                    dbstore.remove(rec);
                }
            }
        });
    }

});
```

```
Ext.reg('moviegrid', MovieGrid);
```

- Test your solution. Observe that the deleted movie is immediately removed from the grid. However, the data is not deleted on the server. If you refresh the page in the browser then it will show the original set of movies based on the JSON file.

