Download

Sencha SDK

`http://www.sencha.com/products/extjs/download/ext-js-4.2.0`

Sencha Cmd

`http://www.sencha.com/products/sencha-cmd/download`

Application Structure

`extjs` contains the Ext JS 4 SDK files

`app.js` contains your application's logic

```
appname/
  app/
    namespace/
      Class1.js
      Class2.js
      ...
  extjs/
  resources/
    css/
    images/
    ...
  app.js
  index.html
```

index.html:

```
<head>
  <link rel="stylesheet" type="text/css"
      href="extjs/resources/css/ext-all.css">
  <script type="text/javascript" src="extjs/ext-debug.js"></script>
  <script type="text/javascript" src="extjs/locale/ext-lang-pl.js"></script>
  <script type="text/javascript" src="app.js">
  </script>
</head>
```

app.js:

```
Ext.application({
  name: 'MyApp',
  launch: function() {
    Ext.create( 'Ext.container.Viewport', {
      layout: 'fit',
      ...
    });
  }
});
```

Deployment

```
sencha generate app MyApp /path/to/MyApp
cd /path/to/MyApp
sencha app build
sencha web start # http://localhost:1841/
sencha web stop
...
sencha generate model User id:int,name,email
sencha generate controller Central
sencha generate view SomeView
```

Class System

```
Ext.define(className, members, onClassCreatedFun);
```

Simple class:

```
Ext.define('My.sample.Person', {
  extends: 'My.sample.Object',
  statics: {
    instanceCount: 0,
    factory: function(name) {
        // 'this' in static methods refer to the class itself
        return new this({name: name});
    }
  },
  config: {
    name: 'Unknown',
    pesel: '776987337823' // generate get/set/apply
  },
  constructor: function(name, config) {
    this.initConfig( config );
    this.self.instanceCount ++ ;
    this.callParent(arguments);
  },
  applyPesel: function(pesel) {
    if (!Ext.isString(pesel) || pesel.length === 0) {
        alert('Error: PESEL must be a valid non-empty string');
    }
    else {
        return pesel;
    }
  }
});
var john = Ext.create(
  'My.sample.Person', { name: 'John', pesel: '86567892332' }
);
```

Simple component:

```
Ext.define('AM.view.user.Edit', {
  extend: 'Ext.window.Window',
  alias: 'widget.useredit',
  title: 'Edit User',
  layout: 'fit',
  autoShow: true,
  initComponent: function() {
    Ext.apply( this, {
      items: [ ... ],
      buttons: [ ... ]
      ...
    });
    this.callParent(arguments);
  }
});
```

Mixins

```
Ext.define('HasCamera', {
  takePhoto: function(){
    alert('Say Cheese! .... Click!');
  }
});

Ext.define( 'Cookbook.Smartphone', {
  mixins: {
    camera: 'HasCamera'
  },
  useCamera: function() {
    this.takePhoto();
  },
  takePhoto: function() {
    this.focus();
    this.mixins.camera.takePhoto.call(this);
  },
  focus: function(){
    alert('Focusing Subject...');
  }
});
```

Singleton

```
Ext.define( 'myclass', {
  singleton: true,
  myFun: function() { console.info( 'x' ); }
}
myclass.myFun();
```

Overriding

```
Ext.define( 'Cookbook.overrides.TextField', {
  override: 'Ext.form.field.Text',
  onRender: function() {
    this.callParent( arguments );
    Ext.core.DomHelper.append( this.el, '<div>' + this.infoText + '</div>' );
  }
});
```

Dynamic class loading

```
Ext.Loader.setConfig({
  enabled: true,
  paths: {
    'Cookbook': 'src/Cookbook'
  }
});
...
Ext.require('Cookbook.Vehicle', function() {
  var van = Ext.create('Cookbook.Vehicle', 'Ford', 'Transit', 60);
  van.travel(200);
});
```

It is recursive and won't stop until all the files needed by the original required classes are loaded.

Naming conventions

One class per file, e.g. `MyCompany/form/action/AutoLoad.js`

Class	`MyCompany.form.action.AutoLoad`
Method & variable & class property	`encodeUsingMd5()`
Static class property	`Ext.MessageBox.YES = "Yes"`

Error handling

```
throw new Error('['+ Ext.getDisplayName(arguments.callee) +'] Some message here');
```

MVC Architecture

Model is a collection of fields and their data. Models know how to persist themselves through the `data` package, and can be linked to other models through associations. Models are normally used with Stores to present data into grids and other components.

View is any type of component - grids, trees and panels are all views.

Controllers are special places to put all of the code that makes your app work - whether that's rendering views, instantiating Models, handling events, or any other app logic. Controllers are the glue that binds an application together. All they really do is listen for events (usually from views) and take some actions.

```
appname/
  app/
    controller/
    model/
    store/
    view/
  data/
  extjs/
  resources/
    css/
```

```
      images/
      ...
    app.js
    index.html
```

app.js:

```
Ext.application({
  requires: ['Ext.container.Viewport'],
  name: 'AM', // namespace
  appFolder: 'app',
  controllers: ['Users'],
  launch: function() {
    Ext.create('Ext.container.Viewport', {
      layout: 'fit',
      items: { xtype: 'userlist' }
    });
  }
});
```

app/controller/Users.js:

```
Ext.define('AM.controller.Users', {
  extend: 'Ext.app.Controller',
  views: ['user.List'],
  stores: ['Users'],
  models: ['User'],
  init: function() {
    this.control({
      'userlist': { itemdblclick: this.editUser }
    });
  },
  editUser: function(grid, record) {
    console.log('Double clicked on ' + record.get('name'));
  }
});
```

app/view/user/List.js:

```
Ext.define('AM.view.user.List' ,{
  extend: 'Ext.grid.Panel',
  alias: 'widget.userlist',
  title: 'All Users',
  store: 'Users',
  initComponent: function() {
    this.columns = [ ... ];
    this.callParent(arguments);
  }
});
```

app/model/User.js

```
Ext.define('AM.model.User', {
  extend: 'Ext.data.Model',
  fields: ['name', 'email']
});
```

app/store/Users.js:

```
Ext.define('AM.store.Users', {
  extend: 'Ext.data.Store',
  model: 'AM.model.User',
  data: [
    {name: 'Ed',    email: 'ed@sencha.com'},
    {name: 'Tommy', email: 'tommy@sencha.com'}
  ]
});
```

```
Ext.widget('useredit'); === Ext.create('widget.useredit')
```

The `data` Package

`Model`: represents some type of data in an application

```
Ext.define('User', {
  extend: 'Ext.data.Model',
  idProperty: 'id', // default is 'id'
  fields: [
    'login', // { name: 'login', type: 'auto' }
    { name: 'postcode', type: 'string' },
    { name: 'id', type: 'int' },
    { name: 'lastLogin', type: 'date', dateFormat: 'd-m-Y' },
    { name: 'email', type: 'string' }
  ],
  validations: [
    { type: 'length', field: 'login', min: 1 },
    { type: 'presence', field: 'id' },
    { type: 'inclusion', field: 'login', list: ['Admin', 'Operator'] },
    { type: 'format', field: 'postcode', matcher: /\d{2}-\d{3}/ },
    { type: 'exclusion', field: 'postcode', list: ['00-000'] },
    { type: 'email', field: 'email' }
  ],
  proxy: {
    type: 'rest',
    url : 'data/users',
    reader: {
      type: 'json',
      root: 'users'
    }
  }
});
var user = Ext.create( 'User', { id: 1, login: 'Yogi' } );
user.set( 'login', user.get('login') + 'Bear');
var errors = newUser.validate();
console.log('Is User valid?', errors.isValid());
console.log('All Errors:', errors.items); //array of all errors found
console.log('Age Errors:', errors.getByField('age')); //errors for the field
```

Optional properties:

- **defaultValue**
- **dateFormat** may be also: 'timestamp' (UNIX) or 'time' (milliseconds)
- **useNull** (for int, float, bool, string): If the value cannot be parsed, null will be used instead of defaultValue

Type	Default value
auto	''
string	''
date	null
int/integer	0
float/number	0
boolean/bool	false

`Store`: collection of `Model` instances

Subclasses: `ArrayStore`, `JsonStore`, `JsonPStore`, `XmlStore`, `TreeStore`, `DirectStore`

```
Ext.create('Ext.data.Store', {
  model: 'User', //alternatively, fields: [...]
  sorters: ['name', 'id'],
  filters: {
      property: 'name',
      value   : 'Ed'
  },
  groupField: 'age',
  groupDir: 'DESC',
  data: [ // alternatively: proxy { ... }
    { firstName: 'Ed',    lastName: 'Spencer' },
    { firstName: 'Tommy', lastName: 'Maintz' }
  ],
  autoLoad: false // true for proxy
});
```

Important methods:

```
load(), sync(),
add(),
each(),
sort( 'columnName', 'ASC' ),
filter(), filterBy(), clearFilter(),
findRecord(), queryBy()
```

Important events: `load`, `beforeload`, `update`, `datachanged`, `write`

Proxy

Used by `Store` to handle the loading and saving of `Model` data.

Subclasses: `Memory`, `LocalStorage`, `SessionStorage`, `Ajax`, `Rest`, `JsonP`, `Direct`

Associations

```
Ext.define('User', {
  extend: 'Ext.data.Model',
  fields: ['id', 'name'],
  proxy: {
    type: 'rest',
    url : 'data/users',
    reader: {
      type: 'json',
      root: 'users'
    }
  },
  hasMany: 'Post' // shorthand for { model: 'Post', name: 'posts' }
});

Ext.define('Post', {
  extend: 'Ext.data.Model',
  fields: ['id', 'user_id', 'title', 'body'],
  proxy: {
    type: 'rest',
    url : 'data/posts',
    reader: {
      type: 'json',
      root: 'posts'
    }
  },
  belongsTo: 'User',
  hasMany: { model: 'Comment', name: 'comments' }
});

Ext.define('Comment', {
  extend: 'Ext.data.Model',
  fields: ['id', 'post_id', 'name', 'message'],
  belongsTo: 'Post'
});
```

Keyboard navigation

```
Ext.FocusManager.enable(true);
var map = Ext.create('Ext.util.KeyMap', Ext.getBody(), [
  {
    key: Ext.EventObject.E, // E for east
    shift: true,
    ctrl: false, // explicitly set as false to avoid collisions
    fn: function() {
      var parentPanel = eastPanel;
      expand(parentPanel);
    }
```

```
  },
  {
    key: Ext.EventObject.W, // W for west
    shift: true,
    ctrl: false,
    fn: function() {
      var parentPanel = westPanel;
      expand(parentPanel);
    }
  },
  {
    key: Ext.EventObject.S, // S for south
    shift: true,
    ctrl: false,
    fn: function() {
      var parentPanel = southPanel;
      expand(parentPanel);
    }
  }
]);
```

Drag & Drop

```
Ext.create("Ext.form.FieldSet", {
  renderTo : Ext.getBody(),
  width : 100,margin:15,
  title : "Drag",
  listeners : {
    render : function(src){
      Ext.create("Ext.dd.DragSource",src.getEl(),{
        ddGroup : "A"
      });
    }
  }
});

Ext.create("Ext.panel.Panel", {
  title: "DropTarget panel",
  height: 200, width: 350, padding: 20,
  renderTo : Ext.getBody(),
  listeners: {
    render: function (src) {
      Ext.create("Ext.dd.DropTarget",src.getEl(),{
        ddGroup : "A",
        notifyDrop : function(dragSource,e,data){
          var elem = Ext.getCmp(dragSource.id);
          src.add(elem);
        }
      });
    }
  },
});
```

Tree:

```
Ext.create('Ext.tree.Panel', {
  title: 'Conference', height: 400,
  store: conferenceStore,
  viewConfig: {
    plugins: {
      ptype: 'treeviewdragdrop',
      dragText:"Drag and drop"
    },
    listeners: {
      beforedrop: function (node, data,overModel) {
        var session = data.records[0];
        if (!session.get("leaf") || overModel.isRoot())
          return false;
        }
      }
    },
    rootVisible: true
});
```

Forms

A `Ext.form.Panel` is nothing more than a basic `Panel` with form handling abilities added. In addition, Form Panels can use any Container Layout, providing a convenient and flexible way to handle the positioning of their fields (default is `anchor`). Form Panels can also be bound to a `Model`, making it easy to load data from and submit data back to the server. Under the hood a Form Panel wraps a `BasicForm` (`formPanel.getForm()`) which handles all of its input field management, validation, submission, and form loading services.

ExtJS4 does not have a form layout. It utilizes the `Ext.form.Labelable` mixin, which allows form fields to be decorated with labels and error messages without requiring a specific layout to be applied to container. `Ext.form.FieldContainer` allows to combine multiple fields into a single container, which also implements the `Ext.form.Labelable` mixin.

```
Ext.create(
  'Ext.form.Panel',
  {
    title: 'User Form',
    bodyPadding: 10,
    url: 'add_user',
    defaults: {
      xtype: 'textfield',
    },
    fieldDefaults: {
      labelAlign: 'top',
      msgTarget: 'side', // qtip, under, title, side, none, component ID
      labelWidth: 75
    },
    ...
    items: [
      {
        xtype: 'displayfield',
        fieldLabel: 'First Name',
        name: 'firstName'
      },
      {
        xtype: 'textfield',
        fieldLabel: 'Login',
        validator: function( val ) {
          if( val.length <= 4 ) return "Login too short";
          return true;
        }
      },
      {
        fieldLabel: 'Email',
        name: 'email',
        vtype: 'email' // 'alpha', 'alphanum', 'url',
        minLength: 4,
        maxLength: 100
      },
      { xtype: 'timefield', name: 'time' },
      { xtype: 'multislider', values: [25, 50, 75], minValue: 0,
        maxValue: 100 },
      {
        fieldLabel: 'Last Login Time',
        name: 'loginTime',
        regex: /^([1-9]|1[0-9]):([0-5][0-9])(\s[a|p]m)$/i,
        maskRe: /[\d\s:amp]/i,
        invalidText: 'Not a valid time.  Must be in the format "12:34 PM".'
      }
    ],
    buttons: [
      {
        text: 'Submit',
        formBind: true, // enabled if form is valid
        disabled: true, // enabled if form is valid
        handler: function() {
          var form = this.up('form').getForm(),
          record = form.getRecord();
          if (form.isValid()) {
            form.updateRecord(record);
            record.save({
              success: function(user) {
                Ext.Msg.alert('Success', 'User saved successfully.');
              },
              failure: function(user) {
                Ext.Msg.alert('Failure', 'Failed to save user.');
              }
            });
          } else {
            Ext.Msg.alert('Invalid Data', 'Please correct form errors.');
          }
        }
      }
    ]
  });

Ext.ModelMgr.getModel('User').load(
  1, { // load user with ID of "1"
    success: function(user) {
      userForm.loadRecord(user);
    }
  });
```

Populating form

```
formPanel.getForm().setValues( dataObj );
formPanel.getForm().loadRecord( model );
formPanel.getForm().load( {
  url: '/myurl',
  waitMsg: 'Processing form...',
  // action is successfull and 'success' property in result is true
  success: function( form, action ) {
    ...
  },
  failure: function( form, action ) {
    if(action.failureType === Ext.form.action.Action.CLIENT_INVALID){
      // form.isValid() returned false
    } else if(action.failureType === Ext.form.action.Action.CONNECT_FAILURE){
      // i.e. invalid URL
      console.info( 'status: ' + action.response.status + ': ' +
        action.response.statusText );
    } else if(action.failureType === Ext.form.action.Action.SERVER_INVALID){
      // i.e. 'succes' property is false
      console.info( action.result.message );
    }
  }
} )
```

Submitting form

```
formPanel.getForm().submit( { url: '...', success: ..., failure: ... } )
```

```
formPanel.getForm().getValues()
formPanel.getForm().updateRecord( formPanel.getForm().getRecord() )
```

Validating form

```
formPanel.getForm().isValid()
```

VType: validation function + error message + (optional) input regexp mask

```
{ xtype: 'textfield', vtype: 'email/alpha/alphnum/url' }
```

```
var telNumberVType = {
  telNumber: function( val, field ) {
    var re = /^\d{4}-\d{3}-\d{4}$/;
    return re.test( val );
  },
  telNumberText: 'Your phone number must only include numbers and hyphens.',
  telNumberMask: /[\d\-]/
};

Ext.apply( Ext.form.field.VTypes, telNumberVType );

...
{ xtype: 'textfield', vtype: 'telNumber' }
...
```

Combobox

```
xtype: 'combobox',
displayField: 'fullName',
valueField: 'userID',
store: store,
queryMode: 'remote', // default
forceSelection: true

// autocompletion
typeAhead: true,
typeAheadDelay: 100,
minChars: 3,
hideTrigger: true
```

DOM

Select

```
el = Ext.get( id/DOM/Ext.Element ) // -> Ext.Element
el.dom
el.select( css-selector )
el.query( css-selector ) -> array of DOM nodes
```

Manipulating

```
el.setStyle( 'font-size', '1.5em' );
el.setStyle( { backgroundColor: 'red' } );
el.addClass( 'book-title' );
el.update( HTML );
```

Traverse

```
el.prev()/next() first()/last() parent()/child('li') up()/down('li')
```

Creating

```
Ext.core.DomHelper.append( el, {
  cls: 'someClass',
  tag: 'li',
  html: 'some HTML here',
  children: [ {...} ]
} );
helper.insertBefore( el, {...});
helper.insertAfter( el, {...});
```

Templates allow us to create HTML strings that contain data placeholders.

```
var template = Ext.core.DomHelper.createTemplate(
  { tag: 'li', html: '{newFeature}' });
template.append( el, { newFeature: 'Row Editor' } );
```

Animations

```
el.puff({
  easing: 'easeOut',
  duration: 1000,
  useDisplay: false
});
component.getEl().frame( 'red', 3, {
  duration: 500
});
```

Events: `beforeanimate`, `afteranimate`, `keyframe`.

Events

```
el.on( 'click', function( e, target, options ) { ... }, self );
el.on({
  click: function(e, target, options) { ... },
  contextmenu: function(e, target, options) { ... },
  self: this
});
```

Dates

```
Ext.Date.format( date, format );
Ext.Date.parse( '2010-05-17', 'Y-m-d' );
Ext.Date.between( date, startDate, endDate ); // -> boolean
Ext.Date.add( date, Ext.Date.YEAR, -1 );
Ext.Date.isValid( 2011, 29, 2 );
Ext.Date.getDayOfYear( date );
Ext.Date.getDayInMonth( date );
Ext.Date.getElapsed( dateA, [dateB] );
Ext.Date.getTimezone( date );
Ext.Date.defaultFormat // used in Ext.util.Format.date( date )

Ext.JSON.encodeDate( date );
```

AJAX, JSON

```
Ext.Ajax.request({
  url: '...',
  params: {
    par1: 10,
    par2: 'value'
  },
  disableCaching: true,
  success: function( response, options ) {
    console.info( response.responseText );
  },
  failure: function( response, options ) { ... },
  callback: function( options, success, response ) { ... },
  timeout: 60000 // default: 30000 msec
});

Ext.encode( json ); // -> String
Ext.decode( str ); // -> JSON
```

Ext.ComponentQuery

Selectors

`panel`
: `Ext.panel` instances

`panel button`
: `Ext.button` that are descendants of `Ext.Panel`

`button, textfield`
: buttons *or* textfields

`component[action]`
: components based on attribute presence

`button[action="saveUser"]`
: components based on attribute values

`textfield:not`
: pseudo-selector `not`

`#usersPanel`
: component based on `id` or `itemId`

`form > textfield{isValid()}`
: all text fields who are direct children of a form and whose `isValid` method evaluates to `true`

Methods

```
Ext.ComponentQuery.query( selector ) // -> Ext.Component[]
Ext.ComponentQuery.is( panel, 'panel' ) // -> boolean
Ext.container.AbstractContainer.
  .query( selector )
  .up( selector )
  .down( selector )
  .child( selector )
```

Aliasing

```
Ext.define('Customer.support.SupportMessage', {
  extend: 'Ext.panel.Panel',
  alias: 'widget.supportMessage',
  ...
});
...
Ext.create('Ext.container.Viewport',layout: 'fit',
  items: [{
    xtype: 'supportMessage'
  }]
});
```

Scoping

```
Ext.bind( function, self ) // -> wrapped function
...
listeners: {
  click: {
    fn: function() { ... },
    scope: self
  },
  dblclick: function() { ... },
  scope: self2
}
```

XTemplate

```
var bugData = [ {
  id: 1, title: 'Bug 1',
  description: 'Bug 1 Description', status: 'In progress', severity: 1
}, ... ];
var tpl = new Ext.XTemplate( '<table>',
     '<tpl for=".">',
       '<tr style="background-color:
```

```
            <tpl if="this.isHighPriority(values.severity)">pink
            <tplelse>yellow</tpl>;">',
          '<td style="background-color: {[values.status === "Complete"
             ? "green" : "transparent" ]};">{title}</td>',
          '<td>{description}</td>',
        '</tr>',
      '</tpl>',
    '</table>',
    {
      isHighPriority: function( severity ) {
        return severity > 3;
      }
    }
  );
tpl.overwrite( Ext.getBody(), bugData );
```

Switch:

```
'<tplswitch="owner">',
    '<tplcase="this.currentUser">',
      --
    '<tpldefault>',
      --
```

Date:

```
  new Ext.XTemplate( '{date:date("Y-m-d")}' )
```

Special variables: `values`, `parent`, `xindex`, `xcount`

Tooltips

```
Ext.tip.QuickTipManager.init();

tooltip: {
  title: "Tooktip Header",
  text: "Tooltip Text"
}
```

Toolbars, menus

```
var panel = new Ext.panel.Panel({
  headerPosition: 'left',
  tools: [
    { type: 'close', handler: function() { ... } }
    // close, minimize, maximize, restore, toggle, gear, prev, next, pin, unpin,
    // right, left, down, up, refresh, plus, minus, search, save, help, print,
    // expand, collapse
  ],
  dockedItems: [{
    xtype: 'toolbar',
    dock: 'top',
    items: [
      { xtype: 'tbtext', text: 'My label' },
      { text: 'My button' }
    ]
  }],
  tbar/bbar/lbar/rbar: ['label', { xtype: 'button', text: 'Click me' } ]
});
```

- `'->'`: `Ext.toolbar.Fill`
- `'-'`: `Ext.toolbar.Separator`
- `' '`: `Ext.toolbar.Spacer`

Layouts

anchor (allows anchoring the container's children according to its dimension; if you resize the outer container, all its children will be automatically resized according to the children's anchor rules)

```
anchor: '80% -20'
```

absolute

```
x: 10,
y: 20,
anchor: '80% -20'
```

hbox

```
flex: 2 | width: 30
layout: {
  type: 'hbox',
  align: 'top' | 'middle' | 'bottom' | 'stretch' | 'stretchmax',
  pack: 'start' | 'center' | 'end'
}
```

vbox

```
flex: 2 | height: 30
layout: {
  type: 'vbox',
  align: 'left' | 'center' | 'right' | 'stretch' | 'stretchmax',
  pack: 'start' | 'center' | 'end'
}
```

accordion

```
layout: {
  type: 'accordion',
  titleCollapse: false,
  animate: true,
  multi: false
}
```

table (HTML table)

```
layout: {
  type: 'table',
  columns: 3,
  tableAttrs: {
    style: {
      width: '100%',
      height: '100%'
    }
  },
  trAttrs: { ... },
  tdAttrs: { ... }
},
items: [
  { rowspan: 3, colspan: 2 }
]
```

column

```
items: [ { columnWidth: 0.3 }, { columnWidth: 0.7 }, { width: 100 } ]
```

card

```
activeItem: 0
```

border

```
items: [
  { region: 'center' },
  { region: 'south', height: 200, split: true, collapsible: true,
    collapsed: true, animCollapse: false, collapseMode: 'mini',
    header: false }
]
```

Grid

```
Ext.create( 'Ext.grid.Panel', {
  store: store,
  selModel: Ext.create( 'Ext.selection.CheckboxModel' ),
  columns: [
    Ext.create('Ext.grid.column.RowNumberer'),
    { xtype: 'templatecolumn', tpl: '{topic} {version}' },
    { xtype: 'booleancolumn', trueText: 'Yes', falseText: 'No' },
    { xtype: 'datecolumn', format: 'Y-m-d H:i:s' },
    { xtype: 'numbercolumn', renderer: Ext.util.Format.usMoney },
    { xtype: 'actioncolumn',
      items: [
        { icon: 'images/edit.png', tooltip: 'Edit',
          handler: function(grid, index, colIndex) { ... } }
      ]
    }
  ]
})
```

Saving component state

```
Ext.state.Manager.setProvider( new Ext.state.LocalStorageProvider() );

var panel = Ext.create( 'Ext.panel.Panel', {
  stateful: true,
  stateId: 'my-panel'
  ...
});
```

To include new properties in state save:

```
constructor: function(config) {
  this.callParent( [config] );
  this.addStateEvents( 'titlechange' );
},
getState: function() {
  var state = this.callParent();
  state.title = this.title;
  return state;
}
```

MessageBox

```
Ext.Msg.alert('Status', 'Changes saved successfully.');

// Prompt for user data and process the result using a callback:
Ext.Msg.prompt('Name', 'Please enter your name:', function(btn, text){
    if (btn == 'ok'){
        // process text value and close...
    }
});
// Show a dialog using config options:
Ext.Msg.show({
     title:'Save Changes?',
     msg: 'You are closing a tab that has unsaved changes. '
       + 'Would you like to save your changes?',
     buttons: Ext.Msg.YESNOCANCEL,
     fn: processResult,
     animateTarget: 'elId',
     icon: Ext.window.MessageBox.QUESTION
});
```