CSC336 – Web Technologies

ECMA Scope Soup

'From the can', alternate recipes, and puzzles.

On the menu:

- ECMA Javascript variables and scope
- In the can: Activation objects, scope chains, name resolution, and execution context
- Too much salt (common mistakes)
- Ext Observable binding contexts
- Alternate binding approaches for Ext Components

Global Variables

- Global variables - exist throughout the life of a script. Also considered public, they are:
  - those NOT declared within a function block
  - declared ANYWHERE in your script without the var keyword

```
<script type="text/javascript">
  globalB = 4;
  var globalA = 3,
      say = console.log;
</script>
```

What is the global object(namespace)?

- Browsers, GreaseMonkey : window
- HTML5 Workers, Node.Js : global

Global Variables

```
<script type="text/javascript">
  globalB = 4;
  var globalA = 3,
      say = console.log;
</script>
```

or, referenced through the global object as:

```
window.globalA
window.say
```

Local Variables

- Local variables - survive only as long as the Function block they are declared in has an execution context.

```
<script type="text/javascript">
  var globalA,
      say = console.log;

  function doIt ( ) {
      var localA = 5;  //local scope only
  }
</script>
```

Something wrong here?

```
<script type="text/javascript">
    var globalA,
          say = console.log,
          a = 4;
    doIt();
    function doIt ( ) {
         say ( a );
         var a = 5;
         say( ++a );
     }
</script>
```

Something wrong here?

```
<script type="text/javascript">
    var globalA,
          say = console.log,
          a = 4;
    doIt();
    function doIt ( ) {
        say ( a );
        var a = 5;
        say( ++a );
     }
</script>
```

Something wrong here?

```
<script type="text/javascript">
    var globalA,
        say = console.log,
         a = 4;
    doIt();   <- this can't work!
    function doIt ( ) {
        say ( a ); <- this is 4, right?
        var a = 5;
        say( ++a );
     }
</script>
```

```
<script type="text/javascript">
    var globalA,
           say = console.log,
           a = 4;
    doIt();    <- sure it does, why?
    function doIt ( ) {
         say ( a );  <- undefined! why?
         var a = 5;
         say( ++a );
     }
</script>
```

introducing...

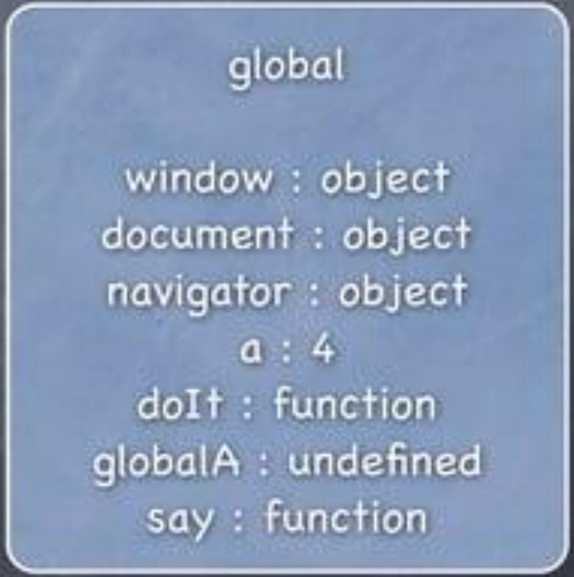
activation objects,
scope chain,
identifier resolution

- Execution context initialized containing a ‘root’ or global object.

- Execution context initialized containing a 'root' or global object.

```
<script type="text/javascript">
  var globalA,
      say = console.log,
      a = 4;
  doIt();
  function doIt ( ) {
       say ( a );
     var a = 5;
     say( ++a );
  }
</script>
```

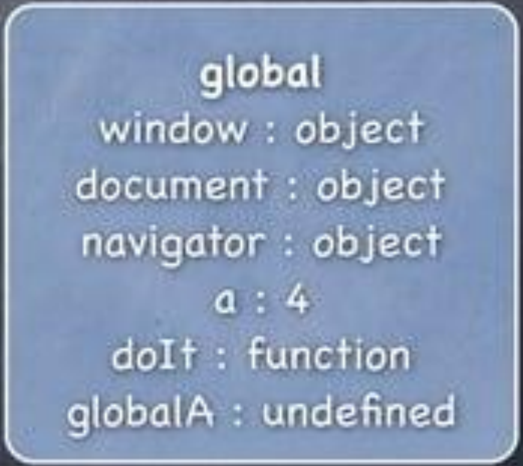
Scope chain



global

window : object
document : object
navigator : object
a : 4
doIt : function
globalA : undefined
say : function

- next, an activation object is prepended to the scope-chain by <u>first scanning the function body</u> for local var's:

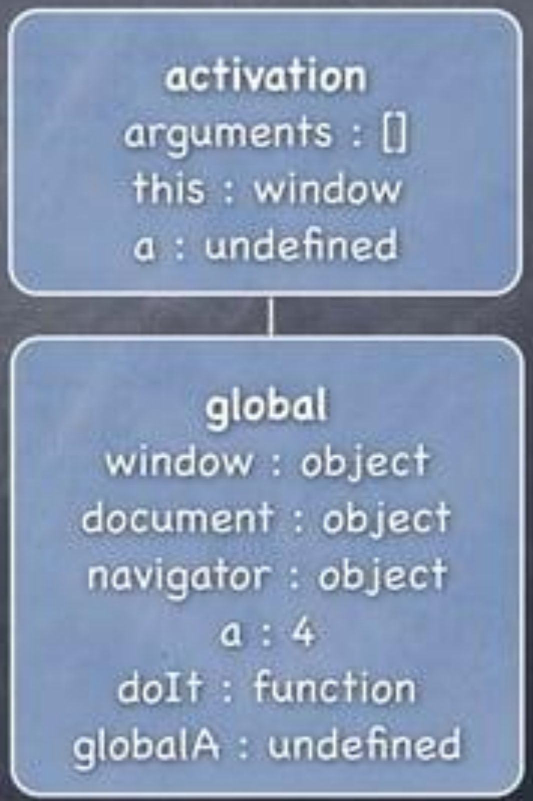


global
window : object
document : object
navigator : object
a : 4
doIt : function
globalA : undefined

- next, an activation object is prepended to the scope-chain by first scanning the function body for local var's:

```
doIt();
function doIt ( ) {
      say ( a );
    var a = 5;
    say( ++a );
  }
```

Scope chain



activation
arguments : []
this : window
a : undefined

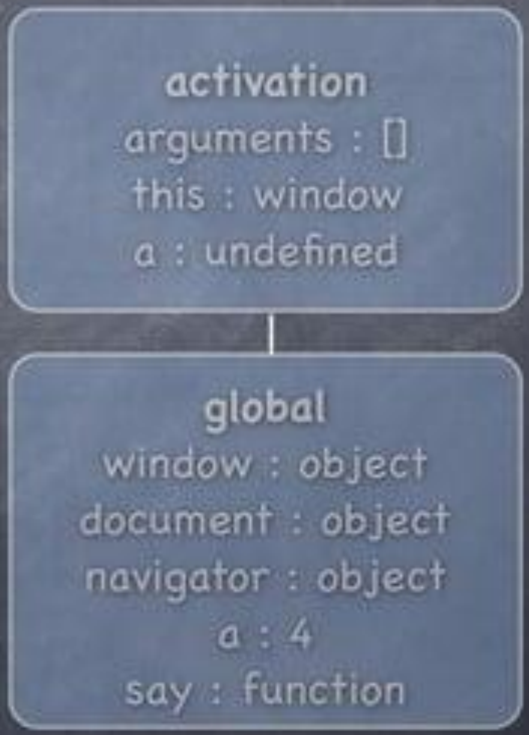
global
window : object
document : object
navigator : object
a : 4
doIt : function
globalA : undefined

thus, a new context is created for doIt, containing a local 'a'

- Identifier resolution: get me 'say' ! (the hunt begins)

```
function doIt ( ) {
    say ( a );
  var a = 5;
  say( ++a );
}
```

Scope chain

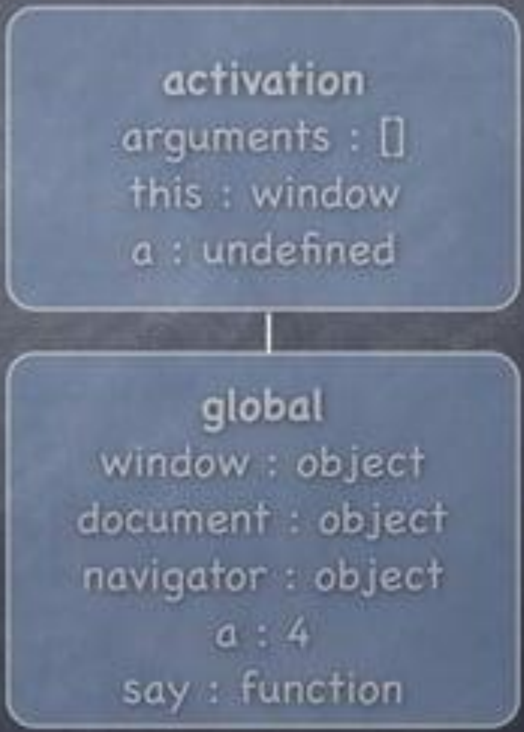


- Identifier resolution: get me 'say' ! (the hunt begins)

```
function doIt ( ) {
     say ( a );
   var a = 5;
   say( ++a );
 }
```

local?, nope!

Scope chain



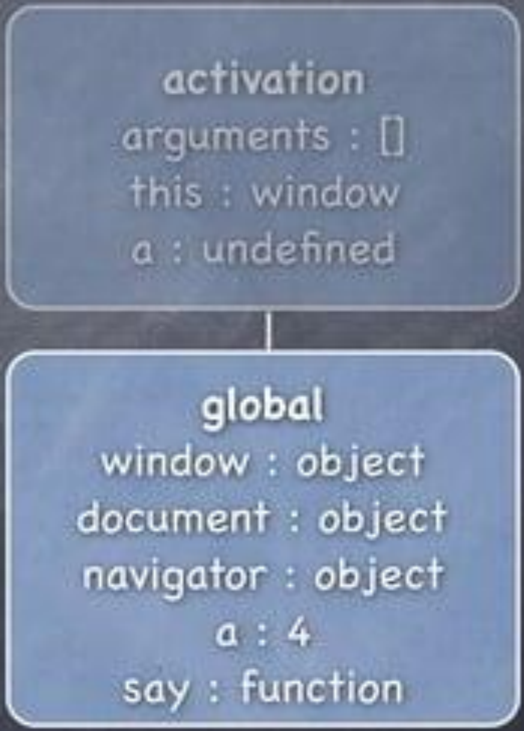
- Identifier resolution: get me 'say' ! (the hunt begins)

```
function doIt ( ) {
      say ( a );
   var a = 5;
   say( ++a );
 }
```

local?, nope!

global has it!

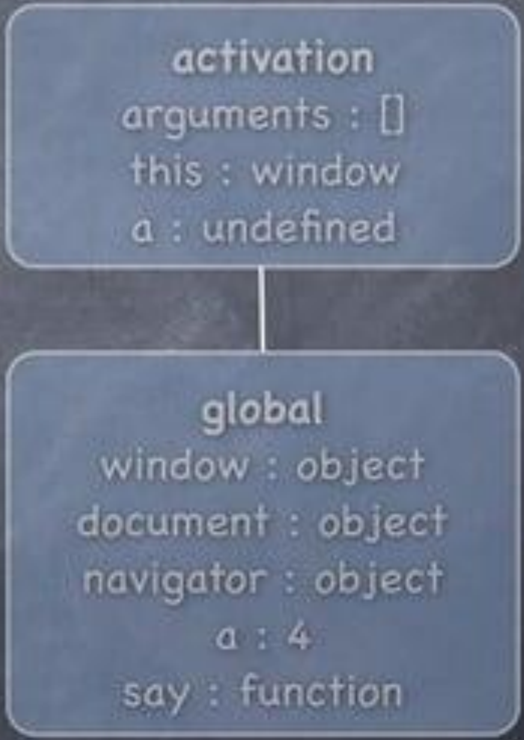
Scope chain



- Now, on to the function argument: ‘a’

```
function doIt ( ) {
    say ( a );
  var a = 5;
  say( ++a );
}
```

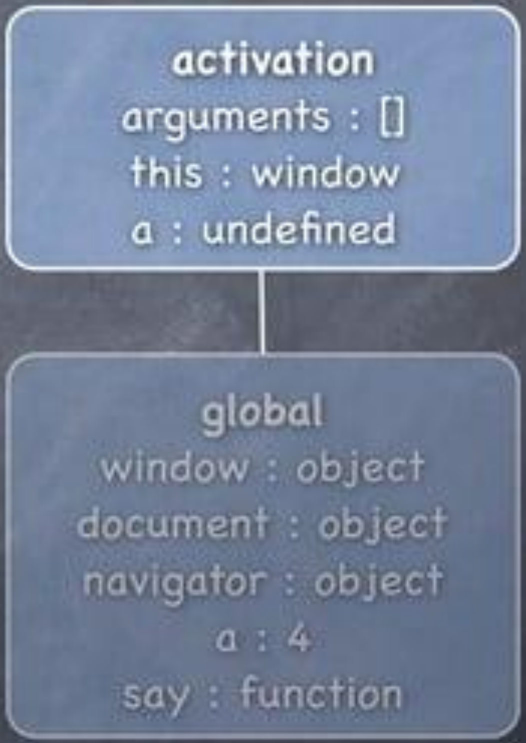
Scope chain



- Now, on to the function argument: ‘a’

```
function doIt ( ) {
    say ( a );   <- prints: undefined
  var a = 5;     <- NOT 5!
  say( ++a );
}
```

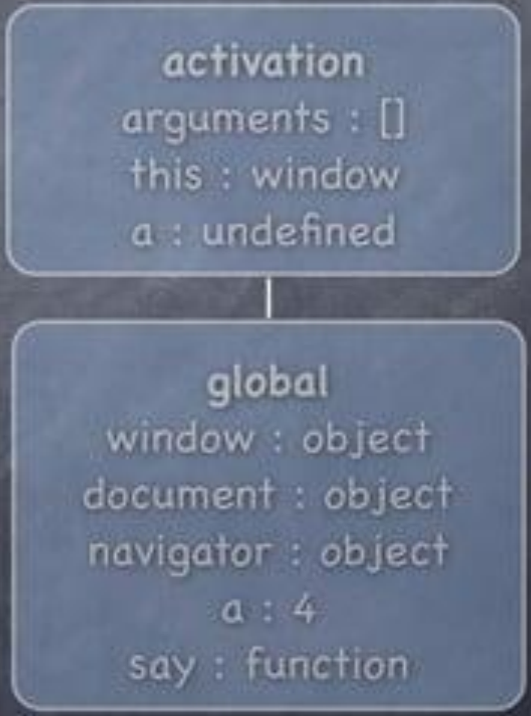
Scope chain



- Now, on to assignment...

```
function doIt ( ) {
     say ( a );
   var a = 5;
   say( ++a );
 }
```

Scope chain

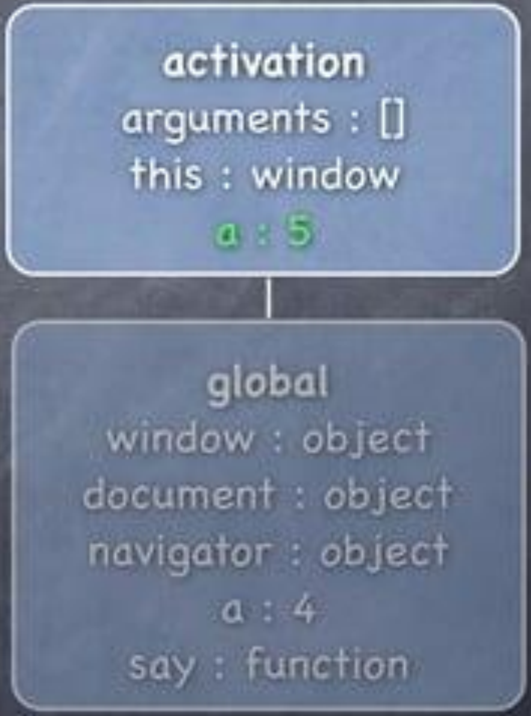


- Now, on to assignment...

```
function doIt ( ) {
     say ( a );
   var a = 5;
   say( ++a );
 }
```

local? yes, set it!

Scope chain

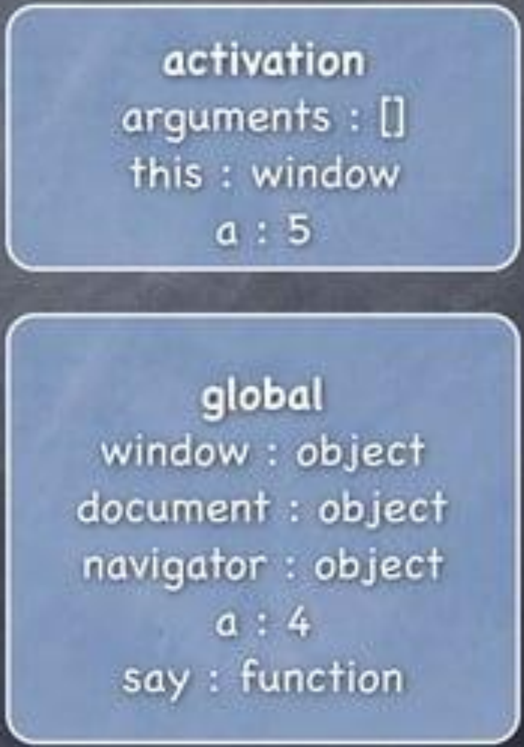


and so on...

- When function doIt is completed, it's execution context (scope chain) is destroyed:

```
doIt();
function doIt () {
     say ( a );
   var a = 5;
   say( ++a );
 }
```

Scope chain



activation
arguments : []
this : window
a : 5

global
window : object
document : object
navigator : object
a : 4
say : function

- When function doIt is completed, it's execution context (scope chain) is destroyed:

```
doIt();
function doIt ( ) {
      say ( a );
    var a = 5;
    say( ++a );
  }
```

and life continues, until the next function block...

Extra fat ?

(Scope Chain Augmentation)

(Scope Chain Augmentation)

Scope Chain Augmentation

The big offenders:

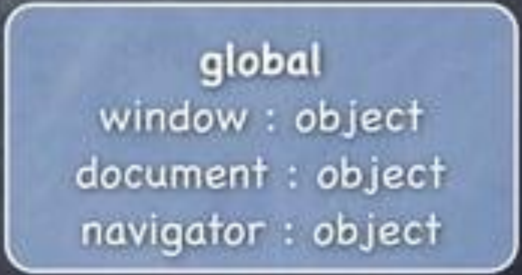
Scope Chain Augmentation

The big offenders:

- Closures
- with Clause
- catch clause of try/catch

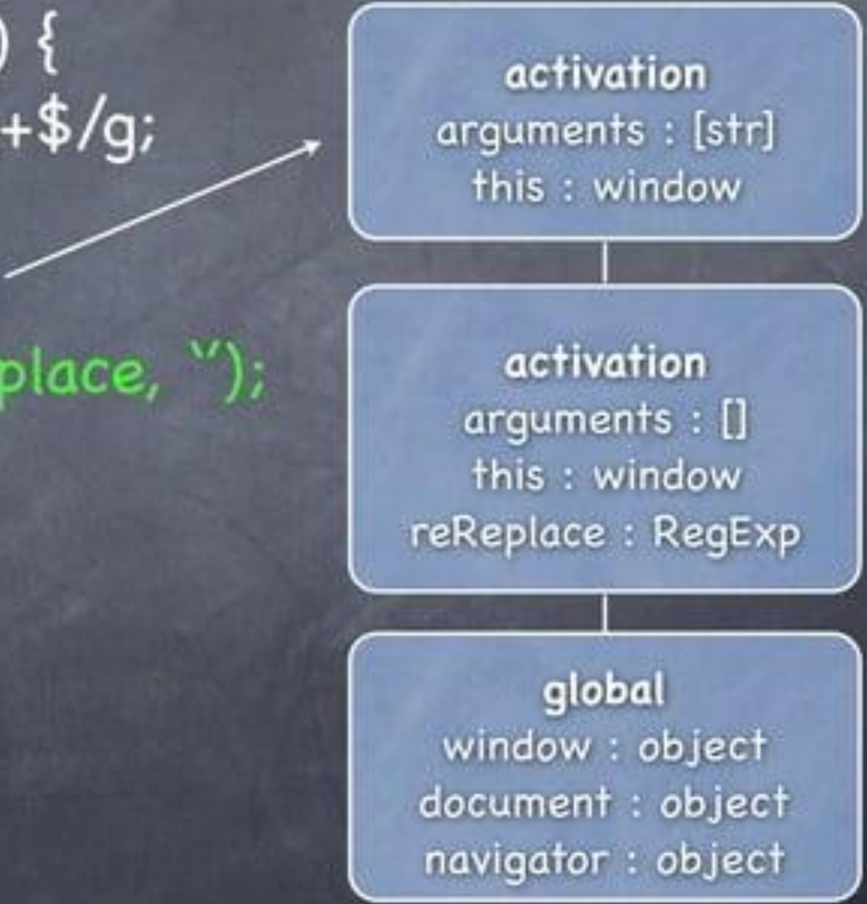
Scope Chain Augmentation
Function Closures

```
var trimString =  function () {
    var reReplace = /^\s+|\s+$/g;

    return ( function (str){
      return str.replace(reReplace, ‘’);
    } );
}( ); <- create the closure
```



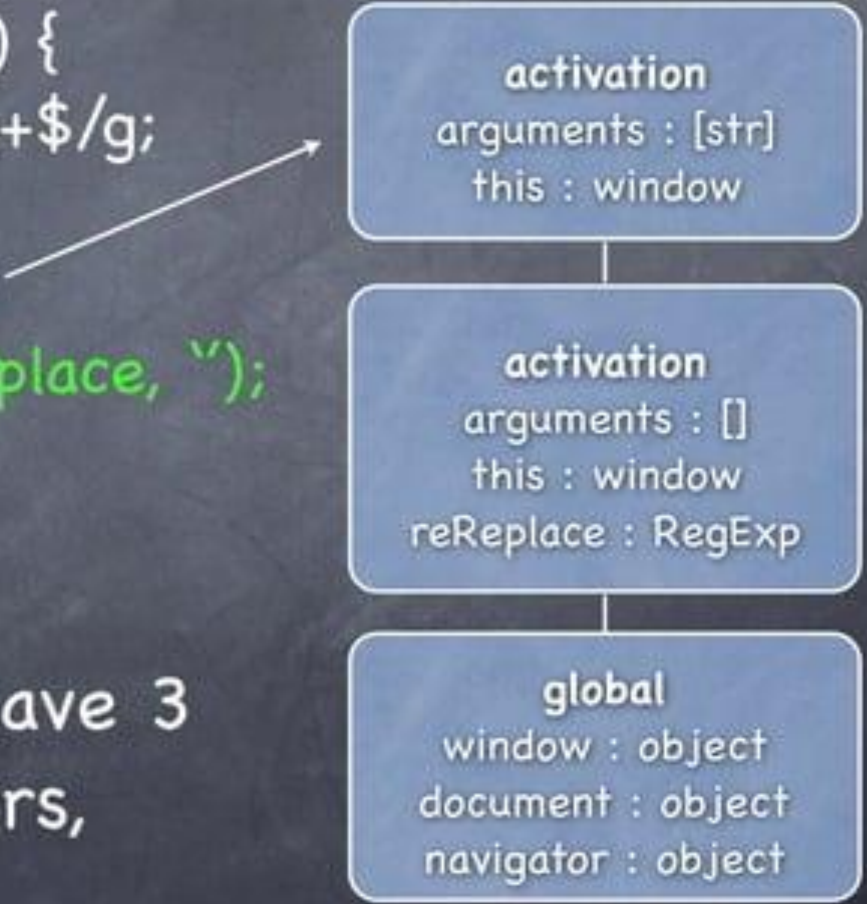
Scope Chain Augmentation
Function Closures

```
var trimString =  function () {
    var reReplace = /^\s+|\s+$/g;

    return ( function (str){
      return str.replace(reReplace, '');
    } );
}( );  <- create the closure
```



Scope Chain Augmentation
Function Closures

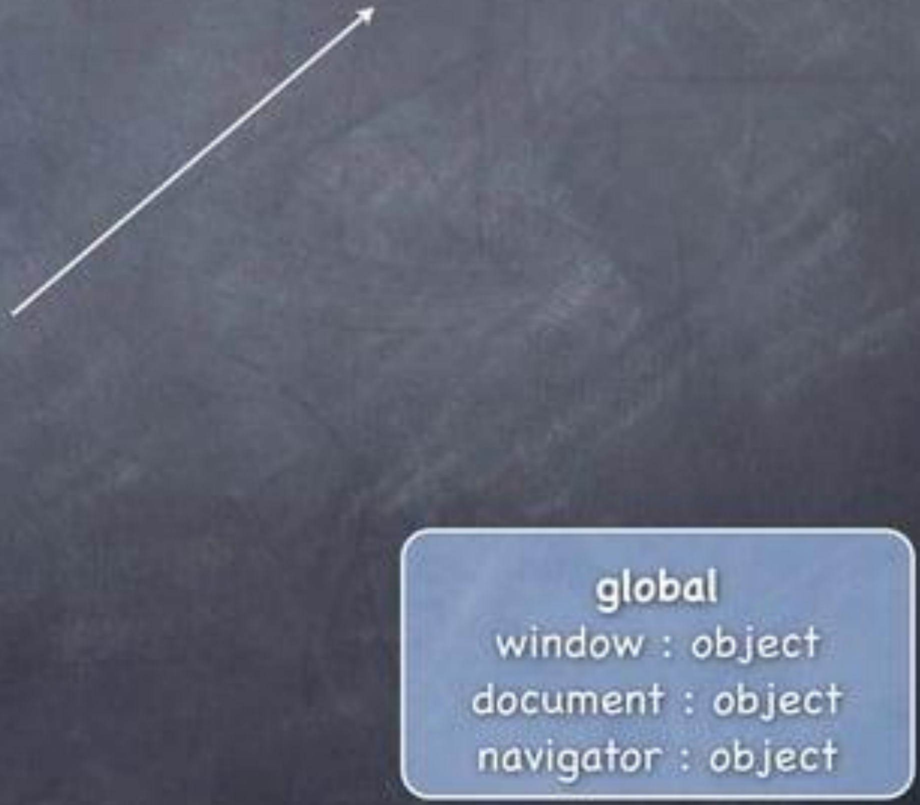
```
var trimString =  function () {
    var reReplace = /^\s+|\s+$/g;

    return ( function (str){
      return str.replace(reReplace, '');
    } );
}( );  <- create the closure
```



Closures will always have 3 scope chain members, minimum!

Scope Chain Augmentation
Function Closures

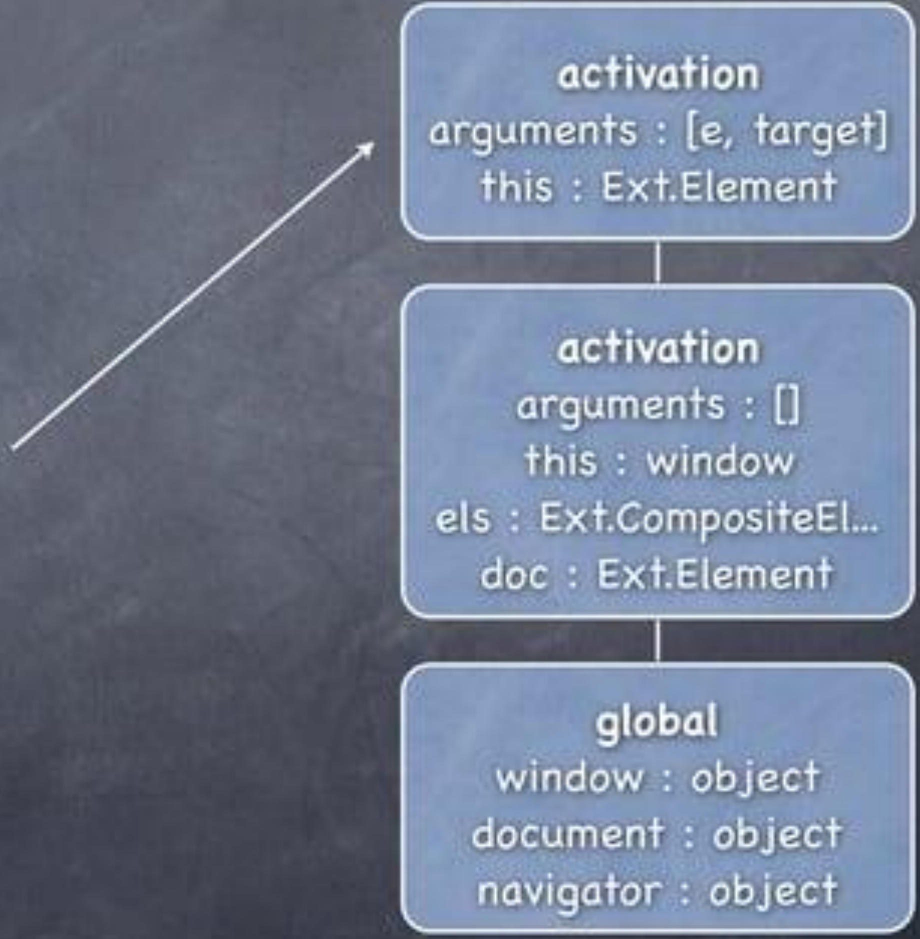
```
function puffEmUp () {
    var els = Ext.select('div'),
        doc = Ext.getDoc();

    els.addClass('puff');

    doc.on({
        'click' : function(e, target){
            els.removeClass('puff');
            els.highlight();
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```



Scope Chain Augmentation

Function Closures

```
function puffEmUp () {
    var els = Ext.select('div'),
        doc = Ext.getDoc();

    els.addClass('puff');

    doc.on({
        'click' : function(e, target){
            els.removeClass('puff');
            els.highlight();
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```



Even more fat !?

(with Clause)

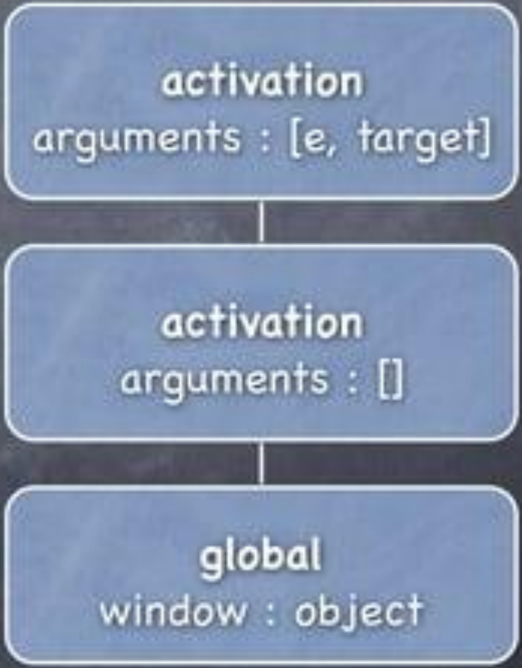
Scope Chain Augmentation

'with' Clause

```
function puffEmUp () {
    var els = Ext.select('div'),
        doc = Ext.getDoc();

    els.addClass('puff');


     doc.on({
        'click' : function(e, target){
            with (els) {
                removeClass('puff');
                highlight();
            }
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

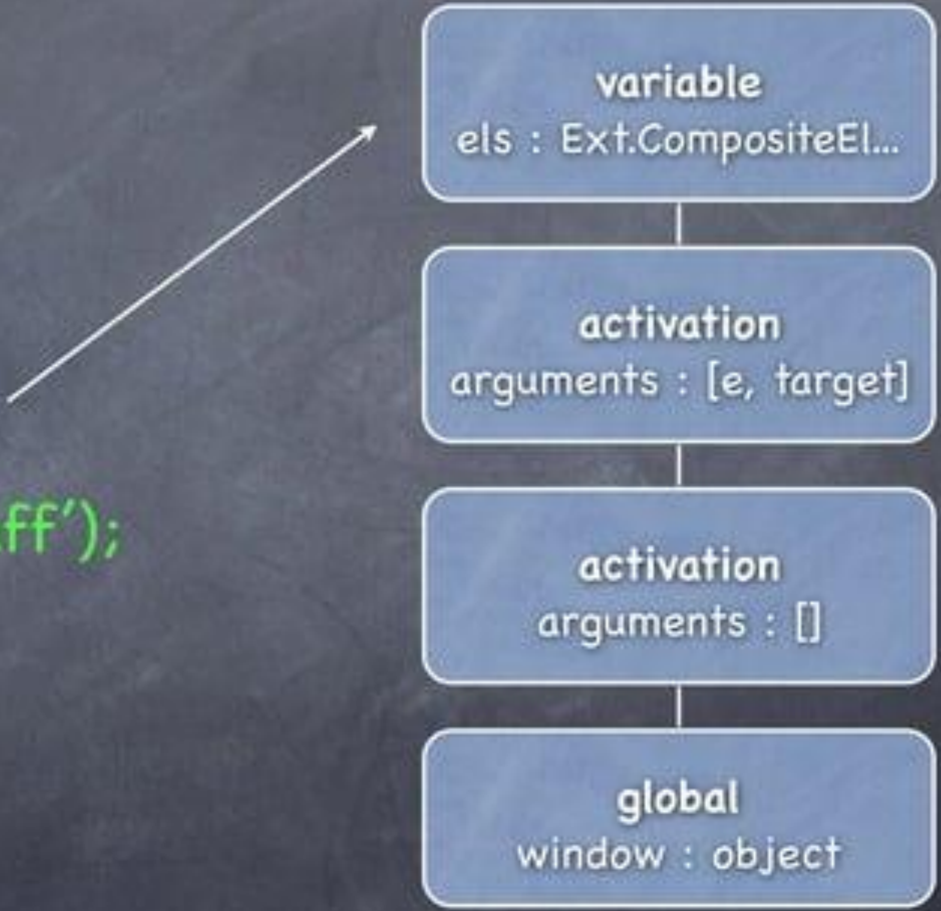
Scope Chain Augmentation
'with' Clause

```
function puffEmUp () {
    var els = Ext.select('div'),
        doc = Ext.getDoc();

    els.addClass('puff');


     doc.on({
        'click' :  function(e, target){
          with (els) {
            removeClass('puff');
            highlight();
          }
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```



Scope Chain Augmentation
'with' Clause

```
function puffEmUp () {
    var els = Ext.select('div'),
        doc = Ext.getDoc();

    els.addClass('puff');


     doc.on({
        'click' : function(e, target){
          with (els) {
            removeClass(`puff');
            highlight();
          }
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```



Let's just eat Lard !

(catch in try/catch)

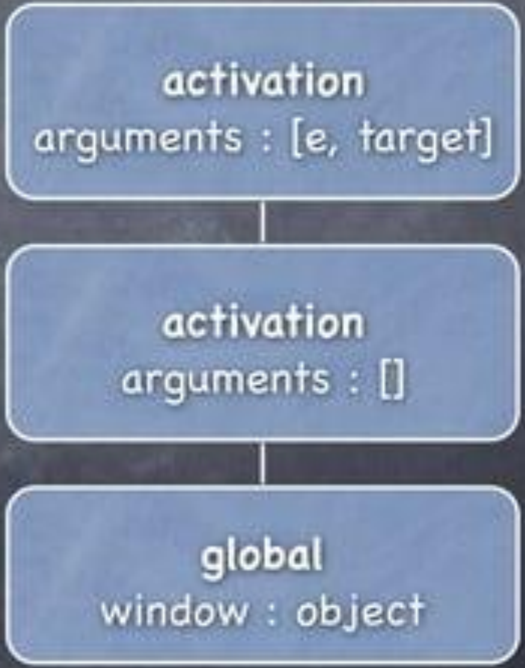
Scope Chain Augmentation
catch block

```
doc.on({
    'click' : function(e, target){
      try{
         with (els) {
           removeClass('puff');
           highlight();
         }
      } catch( err ) {
        Ext.MessageBox.alert('Ooops');
      }
  },
```

Scope Chain Augmentation

catch block

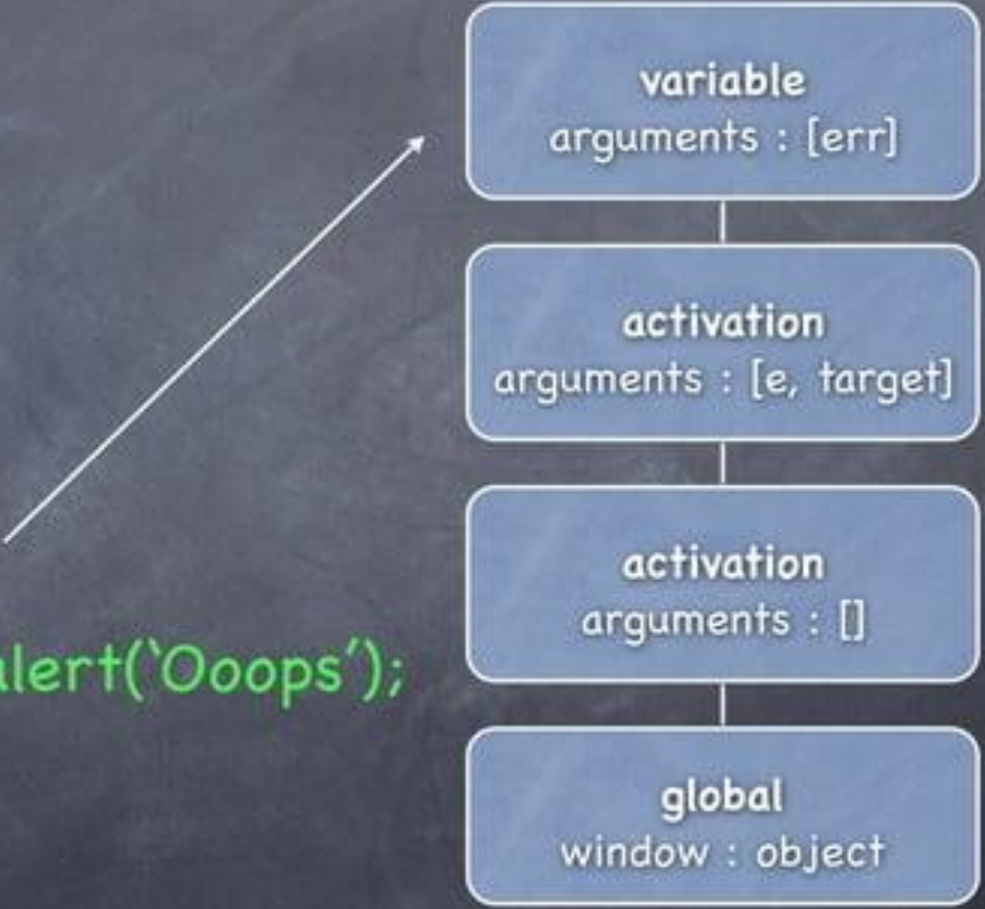
```
doc.on({
    'click' : function(e, target){
      try{
        with (els) {
          removeClass('puff');
          highlight();
        }
      } catch( err ) {
        Ext.MessageBox.alert('Ooops');
      }
  },
```



Scope Chain Augmentation

catch block

```
doc.on({
    'click' : function(e, target){
      try{
        with (els) {
          removeClass('puff');
          highlight();
        }
      } catch( err ) {
        Ext.MessageBox.alert('Ooops');
      }
    },
```



Optimizations

```
function puffEmUp () {
    var els = Ext.select('div'),
        doc = Ext.getDoc();

    els.addClass('puff');

    doc.on({
        'click' : function(e, target){
            els.removeClass('puff');
            els.highlight();
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

Optimizations

Expensive

```
function puffEmUp () {
    var els = Ext.select('div'),
        doc = Ext.getDoc();

    els.addClass('puff');

    doc.on({
        'click' : function(e, target){
            els.removeClass('puff');
            els.highlight();
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

Optimizations

Expensive

```
function puffEmUp () {
    var els = Ext.select('div'),
        doc = Ext.getDoc();

    els.addClass('puff');

    doc.on({
        'click' : function(e, target){
            els.removeClass('puff');
            els.highlight();
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

Better

```
function puffEmUp () {
    var E = Ext,
        els = E.select('div');

    els.addClass('puff');

    E.getDoc().on({
        'click' : function(e, target){
            var collect = els;
            collect.removeClass('puff');
            collect.highlight();
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

Optimize Further

```
function puffEmUp () {
    var E = Ext,
      els = E.select('div');

    els.addClass('puff');

    E.getDoc().on({
      'click' : function(e, target){
          var collect = els;
          collect.removeClass('puff');
          collect.highlight();
      },
      'delegate' : 'div.puff',
      'single' : true
    });
}
```

```
function puffEmUp () {
    var E = Ext,
      els = E.select('div');

    els.addClass('puff');
    E.getDoc().on({
      'click' : function(e, target){
        try{
          this.removeClass('puff');
          this.highlight();
        } catch (err) {
          // a compromise
          E.MessageBox.alert('Oops');
        }
      },
      'scope' : els,
      'delegate' : 'div.puff',
      'single' : true
    });
}
```

Optimize Further

Better

```
function puffEmUp () {
    var E = Ext,
        els = E.select('div');

    els.addClass('puff');

    E.getDoc().on({
        'click' : function(e, target){
            var collect = els;
            collect.removeClass('puff');
            collect.highlight();
        },
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

```
function puffEmUp () {
    var E = Ext,
        els = E.select('div');

    els.addClass('puff');
    E.getDoc().on({
        'click' : function(e, target){
            try{
                this.removeClass('puff');
                this.highlight();
            } catch (err) {
                // a compromise
                E.MessageBox.alert('Oops');
            }
        },
        'scope' : els,
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

Leverage Execution Context

```
function puffEmUp () {
    var E = Ext,
        els = E.select('div');

    els.addClass('puff');
    E.getDoc().on({
        'click' : function(e, target){
            try{
              this.removeClass('puff');
              this.highlight();
            } catch (err) {
              // a compromise
              E.MessageBox.alert('Oops');
            }
        },
        'scope' : els,
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

Leverage Execution Context

```
function puffEmUp () {
    var E = Ext,
        els = E.select('div');

    els.addClass('puff');
    E.getDoc().on({
        'click' : function(e, target){
            try{
              this.removeClass('puff');
              this.highlight();
            } catch (err) {
              // a compromise
              E.MessageBox.alert('Oops');
            }
        },
        'scope' : els,
        'delegate' : 'div.puff',
        'single' : true
    });
}
```

Replace scope-chain traversal with a single context (this) prototype search.

Recommendations

- Declare frequently used function variables as locals.
- Promote frequently used globals UP the scope chain (creating local references to them as necessary)
- Use closures and try/catch handlers sparingly.
- Forget about the 'with' clause! (deprecated in ECMA Javascript 5)

Why is this important at all?

trivia time!

trivia time!

Can you guess how many function definitions there are in the Ext 3.3 framework?

Can you guess how many function definitions there are in the Ext 3.3 framework?

Can you guess how many function definitions there are in the Ext 3.3 framework?

Is it:

Can you guess how many function definitions there are in the Ext 3.3 framework?

Is it:

a: at least 2900

Can you guess how many function definitions there are in the Ext 3.3 framework?

Is it:

a: at least 2900

b: at least 5300

Can you guess how many function definitions there are in the Ext 3.3 framework?

Is it:

a: at least 2900
b: at least 5300
c: at least 8800

Can you guess how many function definitions there are in the Ext 3.3 framework?

Is it:

a: at least 2900

b: at least 5300

c: at least 8800

d: omg! I can't count that high !

If you guessed:

b: at least 5300...

If you guessed:

b: at least 5300...

If you guessed:

b: at least 5300...

You're Correct!

but, you can't leave early!

Common Mistakes and Bottlenecks

Where did it go?

```
<script type="text/javascript">

    function doIt ( ) {
       var a = 5;
       say( ++a );
    }
    setTimeout( `doIt();` , 1000);

</script>
```

Where did it go?

```
<script type="text/javascript">
    function doIt ( ) {
        var a = 5;
        say( ++a );
    }
    setTimeout( `doIt();` , 1000);
</script>
```

```
<script type="text/javascript">
    var doIt = function ( ) {
        var a = 5;
        say( ++a );
    }
    setTimeout( `doIt();` , 1000);
</script>
```

Where did it go?

```
<script type="text/javascript">
    function doIt ( ) {
        var a = 5;
        say( ++a );
    }
    setTimeout( `doIt();` , 1000);
</script>
```

```
<script type="text/javascript">
    var doIt = function ( ) {
        var a = 5;
        say( ++a );
    }
    setTimeout( `doIt();` , 1000);
</script>
```

create a global reference!

Identifier Resolution Mayhem!

```
var getAddress = function(){
 return (
   some.favorite.customer.we.love.name + '\n' +
   some.favorite.customer.we.love.address1 + '\n' +
   some.favorite.customer.we.love.address2 + '\n' +
   some.favorite.customer.we.love.city + '\n' +
   some.favorite.customer.we.love.state + '\n' +
   some.favorite.customer.we.love.zip
  );
};
```

Identifier Resolution Mayhem!

```
var getAddress = function(){
 return (
   some.favorite.customer.we.love.name + '\n' +
   some.favorite.customer.we.love.address1 + '\n' +
   some.favorite.customer.we.love.address2 + '\n' +
   some.favorite.customer.we.love.city + '\n' +
   some.favorite.customer.we.love.state + '\n' +
   some.favorite.customer.we.love.zip
  );
};
```

Don't be a copy/paste victim !

Identifier Resolution Optimized!

```
var getAddress = function () {
    //resolve the global once!
   var cust = some.favorite.customer.we.love;
   return [
     cust.name,
     cust.address1,
     cust.address2,
     cust.city,
     cust.state,
     cust.zip
     ].join('\n');
 };
```

Iteration (with calories)

- Function-based
  - Ext.each (iterable, function)
  - Ext.iterate (iterable, function)
  - $each
  - $jQuery.each(iterable, function)
  - Enumerable.each(iterable, function)
  - Array.forEach(function)

Iteration (with calories)

- Function-based
  - Ext.each (iterable, function)
  - Ext.iterate (iterable, function)
  - $each
  - $jQuery.each(iterable, function)
  - Enumerable.each(iterable, function)
  - Array.forEach(function)

These iterators create additional scope chains.
Reserve for light-duty use only!

Traditional event binding strategies

Classic Quandary

```
{
xtype: 'grid',
store : 'storeId',
buttons : [{
      text : 'Remove Item',
      handler : function(){..} , scope : ???
   },{
      text : 'Close', handler : function(){..}, scope: ???
   }]
}
```

```
{
xtype: 'grid',
store : 'storeId',

initComponent : function(){  //template method
   this.buttons = [{
     text : 'Remove Item',   iconCls : 'remove-icon',
     handler : this.removeItem , scope : this
   },{
     text : 'Close', handler : this.destroy, scope : this
   }];
   this.constructor.prototype.initComponent.call(this);
  },

removeItem : function(button){
    var record = this.getSelectionModel().getSelected();
     .......  //remove the entity...
  }
}
```

Bring your desired scope into context (without sub-classing)

```
{
xtype: 'grid',
store : 'storeId',

initComponent : function(){  //template method
   this.buttons = [{
     text : 'Remove Item',   iconCls : 'remove-icon',
     handler : this.removeItem , scope : this
   },{
     text : 'Close', handler : this.destroy, scope : this
   }];
   this.constructor.prototype.initComponent.call(this);
  },

removeItem : function(button){
    var record = this.getSelectionModel().getSelected();
     .......  //remove the entity...
  }
}
```

Poor-man's Message Bus

- Revised version of Ext.util.Observable class
- Mechanism to loosely-couple behaviors using events (messaging)
- Event binding complexity reduced for most situations.

```
(function(){

      Ext.extend(
          Ext.ux.MsgBus = function(config){
               this.events = {};
               Ext.apply(this,config||{});
               Ext.ux.MsgBus.superclass.constructor.call(this);
            },
            Ext.util.Observable,
            {
               publish : function(topic /* ,variable arguments ,,, */  ){
                    var t = String(topic);
                    this.events[t] || (this.addEvents(t));
                    return this.fireEvent.apply(
                       this,
                       [t].concat(Array.prototype.slice.call(arguments,1))
                    );
               }
            }
       );
       var uxp = Ext.ux.MsgBus.prototype;
       Ext.apply(uxp,{
          subscribe    : uxp.on,       //aliases
          unsubscribe  : uxp.un,
          destroy      : uxp.purgeListeners
       });

})();
```

Follow along at: http://www.sencha.com/forum/showthread.php?42942

Why not as a singleton?

- Poor candidates for Unit testing as ‘state’ is unpredictable over time, cannot be reset.
- Inhibits implementation flexibility.
- The class can be extended/overloaded further to handle custom messaging behaviors.

Basic ux.MsgBus sample

```
Ext.ns('your');
your.bus = new Ext.ux.MsgBus();

your.bus.subscribe('test',
      function(){ console.log(arguments); },
      context,
      {single:true, delay: 500 }
      //standard Observable arguments and options
    );

var wasCancelled = (
   your.bus.publish(
   'test',
   'this is only a test',
   someObj,
   someArray) === false);
```

Multi-Channel

```
Ext.namespace('your');
your.bus = {
   channels: {   // slots, topics (call it what you will)
       chat  : new Ext.ux.MsgBus(),
      feeds  : new Ext.ux.MsgBus(),
      orders : new Ext.ux.MsgBus()
      },
   destroy : function(){
      Ext.iterate(this.channels, Ext.destroy);
   }
};

var channels = your.bus.channels;
your.control = {
    removeItem : function(itemRecord){
      var success;
      //handle order removal centrally here (via data.Store, Ajax, etc)
      success ? channels.orders.publish('itemremoved', itemRecord)
        : channels.orders.publish('itemremovalfailure', itemRecord, response);
    }
};

channels.orders.subscribe({
   'remove' : your.control.removeItem,
   'cancel' : your.control.cancelOrder
});
```

```
{
xtype: ‘grid’,
store : ‘storeId’,
initComponent : function(){  //template method
   this.buttons = [{
     text : ‘Remove Item’,   iconCls : ‘remove-icon’,
     handler : this.removeItem , scope : this
   },{
     text : ‘Close’, handler : this.destroy, scope : this
   }];
   this.constructor.prototype.initComponent.call(this);
  },

removeItem : function(button){
    var record = this.getSelectionModel().getSelected(),
        CB = function( success) {
          success && button.enable(); };

    channels.orders.publish(‘remove’, record, CB );
  }
}
```

Questions ?