# KNOWLEDGE TRANSFER

Greg Solovyev, Staff Engineer, Zimbra Admin Console

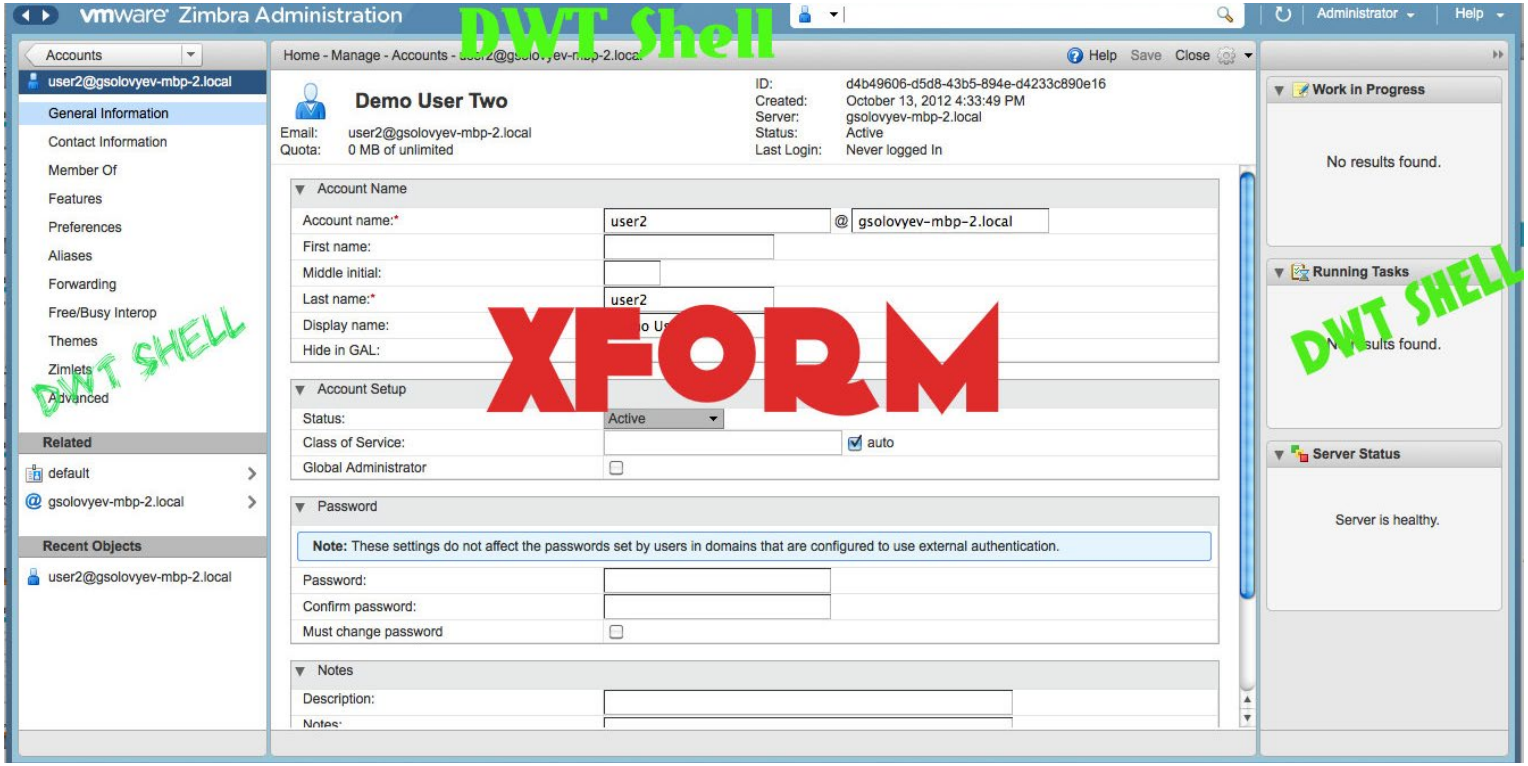
# Table of Contents


1. A bit of background
2. UI layout
3. XForms
    1. Metadata
    2. Code structure
    3. Data abstraction
    4. Handling events
4. Localization
5. Browser support
6. Themes
7. Admin Extensions Framework
    1. Overview
    2. UI components
    3. Server components
    4. Where are they hiding?
    5. FOSS vs. Network extensions
8. Dev environment
    1. Building
    2. Debugging
9. Areas to focus on next

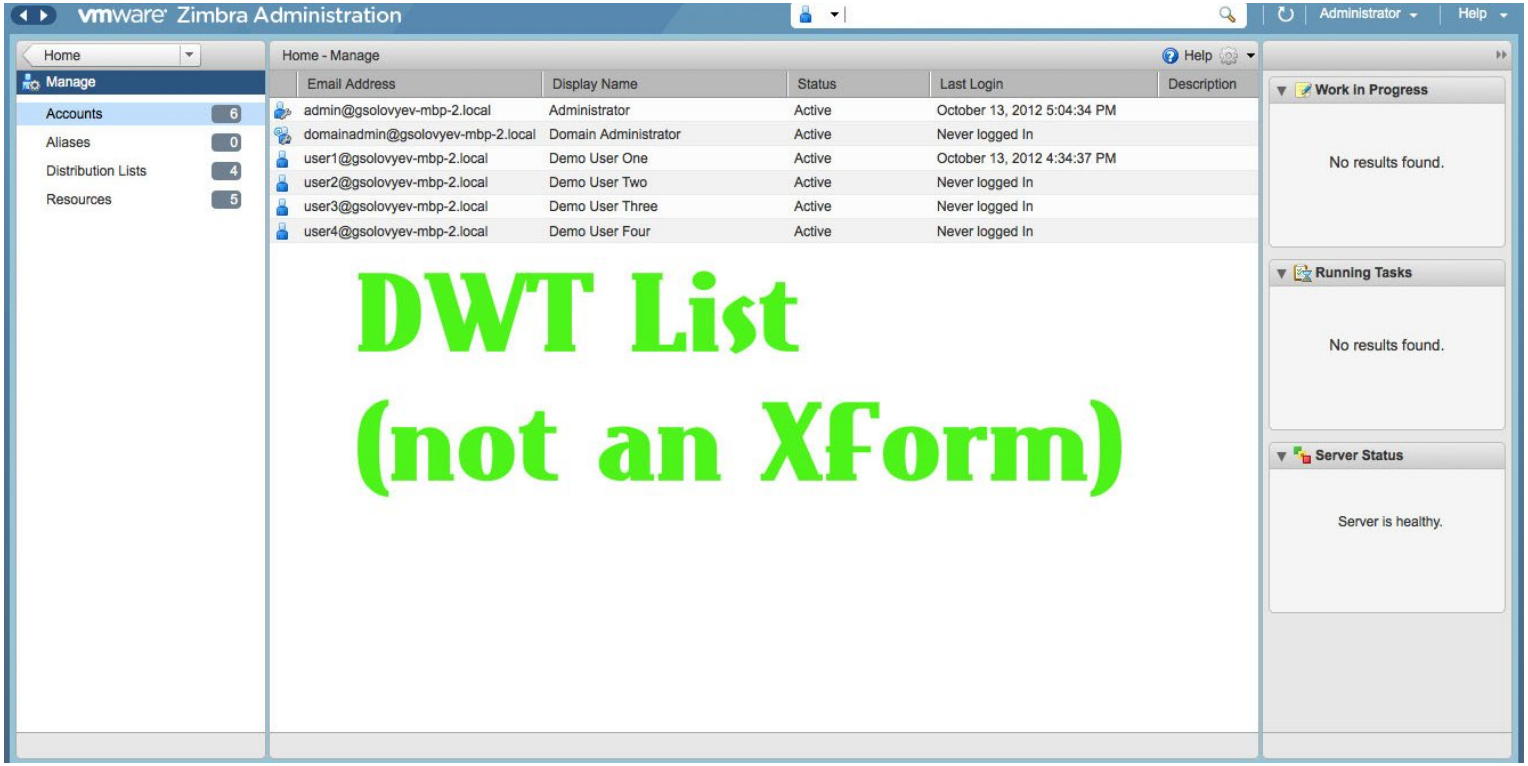
# Background

1. Why this section?
    1. To help you find bugs
2. 8 years of code
    1. before jQuery, SproutCore, Dojo, Cappuccino and most other contemporary AJAX frameworks
3. 8 different engineers
    1. no sinlge coding convention
4. Started as a side project of UI team
    1. Initial framework not optimized for the use case
5. Framework overhaul in GNR (Greg and Charles)
    1. XForms v2 (separate from mail UI)
6. UI overhaul in Iron Maden (Charles' team in China)

# Where are XForms?



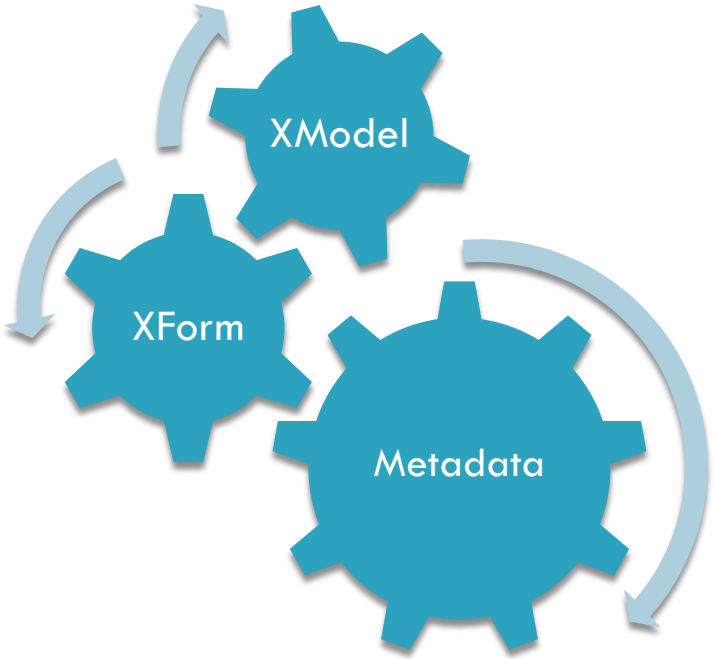
# Where are XForms?



# XForms

What's an XForm?

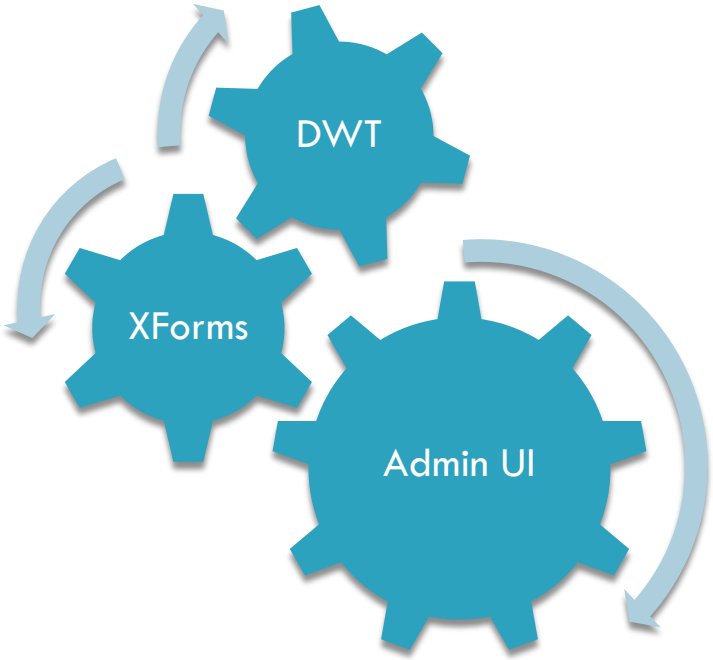
- DWT control that can render a bunch of complicated data forms using HTML elemtns and other DWT Controls (via DWT-XForm item adapters)
- Layout and data binding defined by metadata
- Data behavior rules defined by meta data (XModel)



# XForms

XForms are for rendering forms:

- Rendering (`XForm`, `XFormItem`)
  - Layout
  - UI abstraction
  - `DWT` widgets
  - HTML form elements
- Data binding (`XModel`, `XModelItem`)
  - Apply data to UI elements
  - Data updates
  - Data changes
  - Dependency rules
- Event handling



# XForms Metadata

XForms rendering:

Classes that inherit from `XFormItem` implement the rules for rendering metadata.

Examples of metadata for defining a simple form element:

1) Textfield

```
{ref:ZaAccount.A_firstName, type:_TEXTFIELD_, label:ZaMsg.NAD_FirstName}
```

2) Button with a custom label, a custom icon, an activation handler and a rule that that enables this button when an item is selected in another form element:

```
{type:_DWT_BUTTON_, label:ZaMsg.Previous, width:75, icon:"LeftArrow", disIcon:"LeftArrowDis”,
enableDisableChangeEventSources:[ZaAccount.A2_nonMemberList + "_offset"],
enableDisableChecks[[ZaAccountMemberOfListView.shouldEnableBackButton,ZaAccount.A2_nonMemberList]],
onActivate:"ZaAccountMemberOfListView.backButtonHndlr.call(this,event, ZaAccount.A2_nonMemberList)”
}
```

# XForms Code Structure

Where are XForms?

ZimbraWebClient/WebRoot/js/ajax/dwt/xforms/

- XForm.js
- XFormItem.js
- XModel.js
- XModelItem.js
- XFormGlobal.js

# XForms Code Structure

## XForm.js (XForms "class")

- Draws the form container `div` and `table` that contains all the elements
- pushes data (`instance`) down to form elements (`XFormItems`)

```
XForm.prototype = new DwtComposite;
XForm.prototype.constructor = XForm;
```

Key methods:

```
XForm.prototype.draw
XForm.prototype.outputForm
XForm.prototype.outputItemList
XForm.prototype.setInstance
```

# XForms Code Structure

## XFormGlobal.js

A bunch of utility methods.

Key methods that manage global object references:

```
XFG.assignUniqueId
XFG.getUniqueId
XFG.cacheGet
```

# XForms Code Structure

## XFormItem.js (XFormItem “classes”)

- Each form element is defined by an `XFormItem` class
- Most `XFormItem` classes are defined in XFormItem.js

What does an `XFormItem` class do?

- draws a form element (`<input>` or `<div>`)
    - or places a `DWT` control into the form
- declares metadata handles
- fires UI events
- handles UI events
- handles data events
- fires data evnets
- renders the data by picking it from an data `instance` object

Types of XFormItem

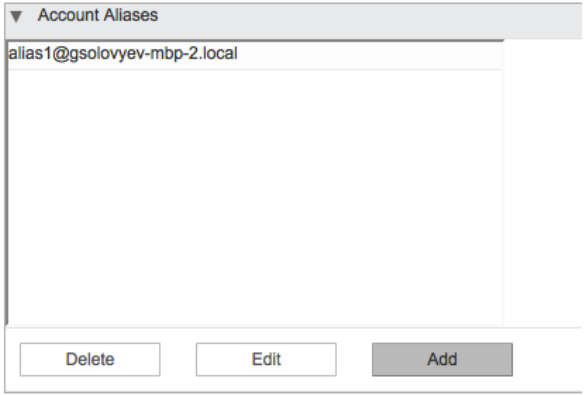
- HTML element
- DWT widget
- Group (several XFormItem elements related by layout)
- Composite (several XFormItem elements rendering single data element)

# XForms Code Structure

## XFormItem.js (XFormItem "classes")

- Base `XFormItem` class
  - defines a basic HTML element
- `Textfield_XFormItem`
  - defines a form element rendered as `<input>` tag

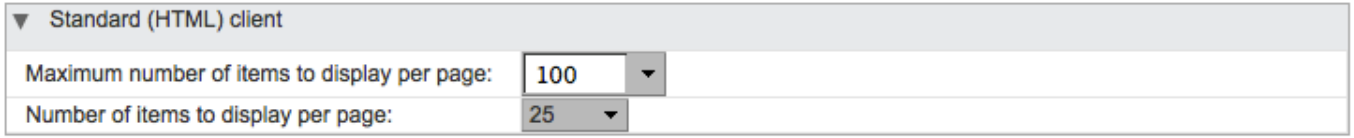

- `Dwt_Adaptor_XFormItem`
  - adaptor for placing a DWT widget on XForm-based form



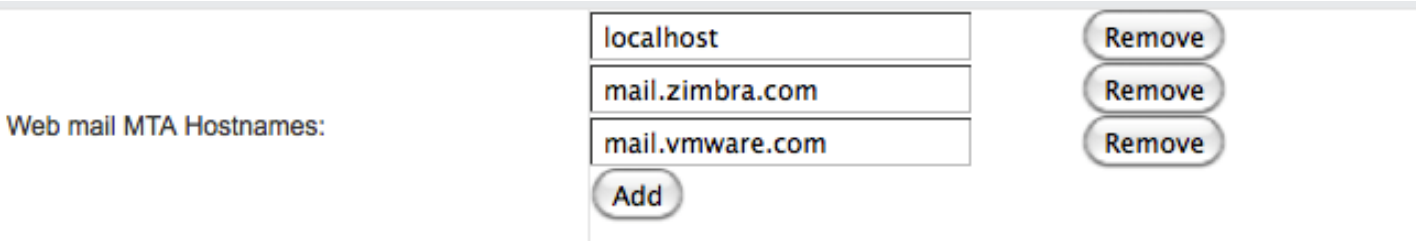
# XForms Code Structure

## XFormItem.js (XFormItem "classes")

- `Group_XFormItem`
  - combine several form items for layout purpose: show/hide together, arrange in a specific way, tabs




- `Repeat_XFormItem`
  - show a form item (or several form items) multiple times. Often used for multi-value attributes such as hostnames assigned to a server, list of installed services, etc



# XForms Code Structure

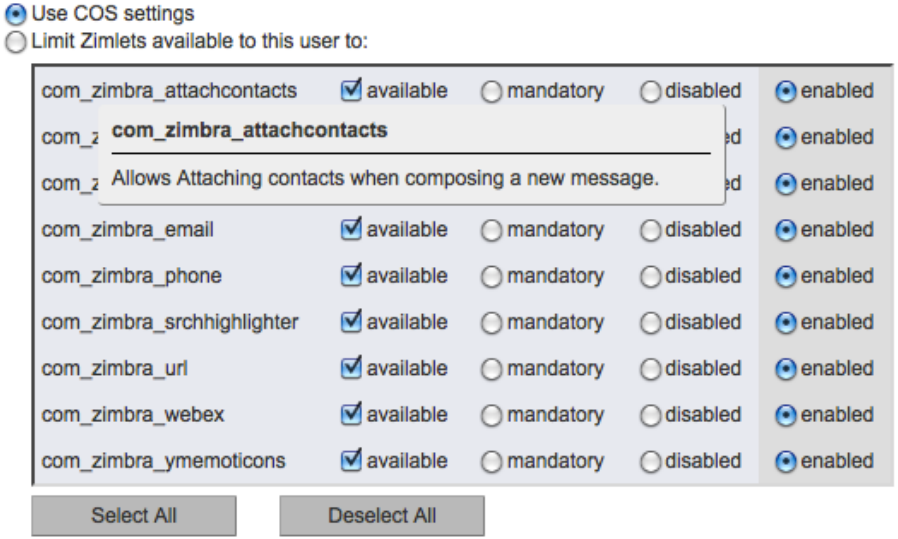
XFormItem.js (XFormItem “classes”)

- `Switch_XFormItem (_SWITCH_) and Case_XFormItem (_CASE_)`
- `_SWITCH_` is a special kind of `_GROUP_` where only one item in the group can be visible at a time
- `_CASE_` is an element inside `_SWITCH_`
- used for all tabbed forms
    - each tab is a `_CASE_`
    - tabbed view is a `_SWITCH_`

# XForms Code Structure

## XFormItem.js (XFormItem "classes")

- `Composite_XFormItem`
  - Similar to Group, but the form items are rendering a single data property: host/port combination (two text fields), a data property that has a default (fall back) value



# XForms Code Structure

Example of declaring an XFormItem class:

```
Textfield_XFormItem = function() {}
XFormItemFactory.createItemType("_TEXTFIELD_", "textfield", Textfield_XFormItem, XFormItem);
Textfield_XFormItem.prototype.containerCssClass = "xform_field_container";
Textfield_XFormItem.prototype.visibilityChecks = [XFormItem.prototype.hasReadPermission];
Textfield_XFormItem.prototype.enableDisableChecks = [XFormItem.prototype.hasWritePermission];
[...]
//appends this form item's HTML code to "html"
Textfield_XFormItem.prototype.outputHTML = function (html,  currentCol) {
[...]
    html.append("<input autocomplete='off' id=\"", this.getId(),
    "\" type=\"", this._inputType, "\"",
    this.getCssString(), this.getChangeHandlerHTML(), this.getFocusHandlerHTML(),
    this.getClickHandlerHTML(), this.getMouseoutHandlerHTML(),
    this.getValue() != null ? " value=\"" + this.getValue() + "\"" : ""),
    ">");
}
```

# XForms Code Structure

This line of code:

```
XFormItemFactory.createItemType("_TEXTFIELD_", "textfield", Textfield_XFormItem, XFormItem);
```

turns "_TEXTFIELD_" into a constant that can be referenced in meta data to place a textfield form item on a form:

```
{ref:ZaAccount.A_firstName, type:_TEXTFIELD_, label:ZaMsg.NAD_FirstName}
```

# XForms Code Structure

Other XFormItem classes in ZimbraWebClient/WebRoot/js/ajax/dwt/xforms folder:

- OSelect_XFormItem.js
- DynSelect_XFormItem.js

More XFormItem classes are defined in many other files in ZimbraWebClient/WebRoot/js/zimbraAdmin and throughout admin extensions:

ZimbraWebClient/WebRoot/js/zimbraAdmin/common/

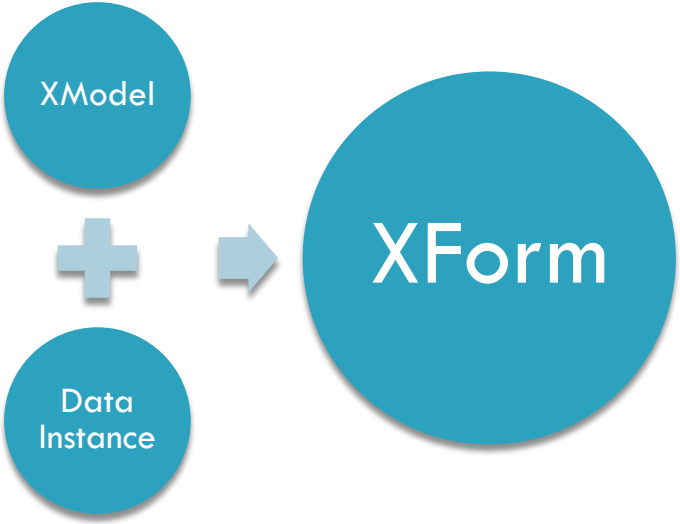
- ACLXFormItem.js
- AutoComplete_XFormItem.js
- EmailAddr_FormItem.js
- HostPort_XFormItem.js
- LDAPURL_XFormItem.js

etc...

# XForms Data Abstraction

XModel (XModel.js) and XModelItem (XModelItem.js) are data abstraction components of XForms

- `XModel` is a "data type"
- `XModel` defines how an `XForm` interacts with a data instance
- `XModel` consists of a collection of `XModelItem` definitions
- `XModelItem` describes a section of data that applies to one `XFormItem`
- Each object type has it's own `XModel`



# XForms Data Abstraction

`XModel` defines:

- data manipulation rules
  - If propery P1 has value V1, then property P2 has value V2
- data retrieval rules
  - If property P1 is null, then it's *effective* value is the value of property V2
- data events
  - each item in the model can fire an event to its subscribers when it is being changed
  - Example 1: when any property of an account data object is changed, an event is fired that tells the `XForm` to enable "Save" button
  - Example 2: when zimbraCOSId property of an account data object is changed, all properties that are inherited from COS change their displayed values

# XForms Data Abstraction

Examples of XModels:

- `ZaAccount.myXModel` (describes an Account object)
- `ZaCos.myXModel` (describes a COS object)
- `ZaServer.myXModel` (describes a Server object)
- `ZaReindexMailbox.myXModel` (describes an object that is used by Reindex Mailbox dialog

etc...

Usually defined in files with corresponding names:

- ZimbraWebClient/WebRoot/js/zimbraAdmin/accounts/model/ZaAccount.js
- ZimbraWebClient/WebRoot/js/zimbraAdmin/cos/model/ZaCos.js
- ZimbraWebClient/WebRoot/js/zimbraAdmin/server/model/ZaServer.js

# XForms Data Abstraction

Similarly to how `XFormItem` describes a type of UI element, `XModelItem` describes a type of data: String, Integer, Date, List, Enumeration, etc.. each data type has a corresponding class that inherits from `XModelItem` class.

Example of defining an `XModelItem` for String data type:

```
String_XModelItem = function(){}
XModelItemFactory.createItemType("_STRING_", "string", String_XModelItem)  ;
String_XModelItem.prototype.validateType =
XModelItem.prototype.validateString;
String_XModelItem.prototype.getDefaultValue = function () { return ""; };
```

# XForms Data Abstraction

More examples of classes that inherit from `XModelItem` and define various data types:

```
Datetime_XModelItem = function(){}
XModelItemFactory.createItemType("_DATETIME_", "datetime", Datetime_XModelItem);
Datetime_XModelItem.prototype.validateType = XModelItem.prototype.validateDateTime;
Datetime_XModelItem.prototype.getDefaultValue = function () {    return new Date(); };

List_XModelItem = function(){}
XModelItemFactory.createItemType("_LIST_", "list", List_XModelItem);
List_XModelItem.prototype.getDefaultValue = function () {return new Array(); };
List_XModelItem.prototype.outputType = _LIST_;
List_XModelItem.prototype.itemDelimiter = ",";
List_XModelItem.prototype.getListItem = function ()   {return this.listItem;}
List_XModelItem.prototype.getterScope = _MODELITEM_;
List_XModelItem.prototype.setterScope = _MODELITEM_;
List_XModelItem.prototype.getter = "getValue";
List_XModelItem.prototype.setter = "setValue";
List_XModelItem.prototype.getValue =  function(ins, current, ref) { ... }
List_XModelItem.prototype.setValue = function(val, ins, current, ref) { ... }
List_XModelItem.prototype.initializeItems = function () { ... }
```

# XForms Data Abstraction

Example of XModel:

```
ZaAccount.myXModel = {
    items: [
           {id:ZaItem.A_zimbraId, type:_STRING_, ref:"attrs/" + ZaItem.A_zimbraId},
           {id:ZaItem.A_zimbraACE, ref:"attrs/" + ZaItem.A_zimbraACE, type:_LIST_},
           {id:ZaAccount.A2_errorMessage, ref:ZaAccount.A2_errorMessage, type:_STRING_},
           {id:ZaAccount.A2_warningMessage, ref:ZaAccount.A2_warningMessage, type:_STRING_},
           {id:ZaAccount.A_name, type:_STRING_, ref:"name", required:true,
               constraints: {type:"method", value:
                   function (value, form, formItem, instance) {
                      if (value){
                          if(ZaAccount.isValidName(value)) {
                                 return value;
                          } else {
                                 throw ZaMsg.ErrorInvalidEmailAddress;
                          }
                      }
                   }
               }
            },
...   ]
}
```

# XForms Data Abstraction

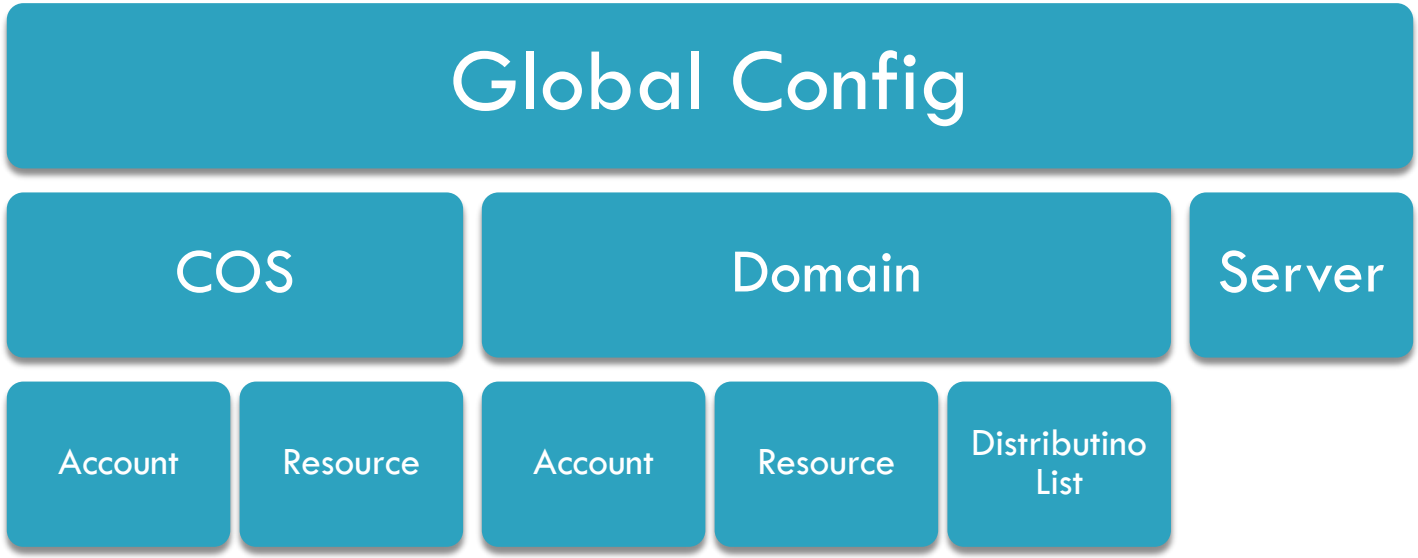
Important properties of an item in an XModel:

- `id` – identificator
- `ref` – path to the value inside the instance obejct. E.g.: “attrs/lastName” means that this value is located in `instance.attrs.lastName`. if “ref” is not defined, the value of “id” is used
- `type` – reference to a data type (a corresponding `XModelItem` class). Similar to `XFormItem` types, `XModelItem` types are referenced by constants: `_STRING_`, `_INTEGER_`, `_EMAIL_ADDRESS_`, `_ENUM_`, etc
- `getter` – a reference to a function that retreives the value of the item. Allows customizing data retrieval rules. E.g.: account properties that fall back to COS values.
- `setter` – a reference to a function that sets the value of the item.
- `getterScope` – tells the model on which object to call the `getter` method (data instance object or model object)
- `setterScope` – same for `setter` method (look at `XModel.prototype._makePathGetter` in XModel.js for more details)

# XForms Data Abstraction

What are these _SUPER_****_ITEM_ and _COS_***_ITEM_ that I see everywhere?

These are special XModelItems and XFormItems that describe behavior of data and UI elements that can inherit values from other data objects:



# XForms Data Abstraction

- Examples of inherited property values:
  - Account preferences (zimbraPref***) fall back to COS
  - Server settings fall back to Global Config
  - Domain settings fall back to Global Config
  - Some Account settings fall back to Domain settings

# XForms Data Abstraction

```
Cos_String_XModelItem = function () {}
XModelItemFactory.createItemType("_COS_STRING_", "cos_string", Cos_String_XModelItem);
Cos_String_XModelItem.prototype.getter = "getValue";
Cos_String_XModelItem.prototype.getterScope = _MODELITEM_;
Cos_String_XModelItem.prototype.setter = "setLocalValue";
Cos_String_XModelItem.prototype.setterScope = _MODELITEM_;
Cos_String_XModelItem.prototype.getValue = function (instance, val, ref) {...}
...
```

**When Account's property is NULL, the effective value is the value of the same property in this account's Class Of Service or Domain**

- Data types (`XModelItem`): `_COS_NUMBER_`, `_COS_EUM_`, `_COS_LIST_`, etc
- Form element types (`XFormItem`): `_SUPER_CHECKBOX_`, `_SUPER_TEXTFIELD_`, `_SUPER_HOSTPORT_`, `_SUPER_SELECT1_...`

# XForms: handling UI events

Events can be fired by UI objects (XForm and XFormItem) and by data model.

Events fired by UI objects:

- onChange – something was typed into a text box
- onClick
- onFocus
- keyUp
- keyDown
- onActivate (_BUTTON_ and _DWT_BUTTON_)
- onSelection (_DWT_LIST_)
- any DWT event that you want to expose via metadata

# XForms: handling data events

How do we:

- enable/disable “calendar” preferences when “Calendar” feature checkbox is checked/unchecked?
- update all COS-based properties of an Account when admin changes this account’s COS?
- hide a UI element if the admin does not have a “read” permission for the LDAP attribute(s) represented by the element in the context of the LDAP object currently loaded into the form?
- disable a UI element if the admin does not have a “write” permission for the LDAP attribute(s) represented by the element in the context of the LDAP object currently loaded into the form?
- unhide/enable UI elements when a different object is loaded into the form without redrawing the whole form?

# XForms: handling data events

A UI element can change dynamically in one of the three ways:

- show/hide
- enable/disable
- value change

Hence, each `XFormItem` has three "changeEventSources" (see XFormItem.js)

- `visibilityChangeEventSources`
- `enableDisableChangeEventSources`
- `valueChangeEventSources`

Each of these sources is an array of references to objects in a data model. E.g., the following line tells a group of UI elements to reevaluate it's enabled/disabled state each time `zimbraFeatureEnabled` or `zimbraCosId` attributes of an account are changed:

```
enableDisableChangeEventSources:[ZaAccount.A_zimbraFeatureMailEnabled, ZaAccount.A_COSId]
```

# XForms: handling data events

How does a UI element evaluate whether it should be hidden or visible, enabled or disabled?

- each UI element can have an array of “check”
- each “check” is a function reference that returns true or false
- when a re-evaluation event is triggered (e.g.: in the previous example value of zimbraCosId attribute is changed) the UI element evaluates all “checks” and if any of the “checks” returns false, the result is false:

```
enableDisableChecks:[[XForm.checkInstanceValue,ZaAccount.A_zimbraFeatureMailEnabled,"TRUE"]]

visibilityChecks:[ZaAccountXFormView.isSendingFromAnyAddressDisAllowed,[ZaItem.hasReadPermission
,ZaAccount.A_zimbraAllowFromAddress]]
```

# XForms: handling data events

Some most frequently used “checks”:

- `XForm.checkInstanceValue`
- `XForm.checkInstanceValueNot`
- `XForm.checkInstanceValueEmty`
- `XForm.checkInstanceValueNotEmty`
- `ZaItem.hasReadPermission`
- `ZaItem.hasWritePermission`

```
XForm.checkInstanceValue = function(refPath,val) {
    return (this.getInstanceValue(refPath) == val);
}
XForm.checkInstanceValueNot = function(refPath,val) {
    return (this.getInstanceValue(refPath) != val);
}
XForm.checkInstanceValueEmty = function(refPath) {
return
AjxUtil.isEmpty(this.getInstanceValue(refPath));
}
```

by default most form elements have `ZaItem.hasReadPermission` as a visibility check and `ZaItem.hasWritePermission` as an enable/disable check.

See ZimbraWebClient/WebRoot/js/zimbraAdmin/common/ZaItem.js

# Localization

- Same framework as mail UI
- ZaMsg.properties (ZaMsg_**.properties) – strings used in the UI: labels, messages, notices, etc
- ZabMsg.properties – “branded” strings
- AjxMsg.properties – strings used by DWT controls
- *.properties files for each admin extension:
  - com_zimbra_cert_manager.properties

# Localization

Downloads

Below are links to Zimbra utilities. For instructions about how to use these utilities, go to Help

**Downloads for Administrators**

General ZCS Migration Wizard (32bits) General ZCS Migration Wizard (64bits)

This Windows application performs a server to server migration of mail, calendar, and contacts from Microsoft Exchange, PST file or IBM Domino to ZCS.

ZCS Migration Wizard Exchange

This Windows application performs a server to server migration of mail, calendar, and contacts from Microsoft Exchange to ZCS.

ZCS Migration Wizard Domino

This Windows application performs a server to server migration of mail, calendar

**Downloads for Em**

PST Import Wizard

This Windows application imports Outlook PST file to a ZCS server for a s

Work in Progress

No results found.

Running Tasks

No results found.

Server Status

Server is healthy.

These links are being generated by the installer and appended to the ZaMsg.properties

I cannot seem to find the constants for these links in any of the *.properties files

# Browser support

- Same as mail UI
- Firefox (works best)
- Chrome (works)
- Safari (mostly works as long as it works in Chrome)
- IE (a pain)
- No mobile version

# Themes

- Almost same as mail UI
  - ZimbraWebClient/WebRoot/admin_skins
  - defines layout outside of XForms
  - can be overwritten in web.xml (zimbraDefaultAdminSkin)
  - can be overwritten with ?skin={theme name} query parameter
  - name of the currently loaded skin is saved in ZA_SKIN cookie
  - consists of skin.html, skin.js, images, logos, manifest file, properties file, CSS file
  - Current skin created by Charles' team in China

# Admin Extensions

- Intent:
  - separate network code from FOSS code
  - allow extending the Admin Console in any possible way (add/remove controls to/from any form, tie into external data sources, add entire new sections)
- Kind of like a virus
  - no security
  - partially implemented, unenforced integration hooks

# Admin Extensions UI

- Integration Hook Example (ZaTabView.js):

(function that returns metadata for rendering a form)

```
ZaTabView.prototype.getMyXForm = function (entry) {
    var xFormObject = new Object();
    //Instrumentation code start
    if(ZaTabView.XFormModifiers[this._iKeyName]
        var methods = ZaTabView.XFormModifiers[this._iKeyName];
        var cnt = methods.length;
        for(var i = 0; i < cnt; i++) {
            if(typeof(methods[i]) == "function") {
                methods[i].call(this,xFormObject,entry,
            }
        }
    }
    //Instrumentation code end
    return xFormObject;
}
```

Empty metadata object

Form name

Iterate through registered "modified" functions

Return metadata object

# Admin Extensions UI

- Integration Hook Example (ZaItem.js):

(function that submits a modified data object such as Account, Server, Domain, etc to the server)

```
ZaItem.prototype.modify = function (mods, tmpObj) {
    //Instrumentation code start
    if(ZaItem.modifyMethods[this._iKeyName]) {
        var methods = ZaItem.modifyMethods[this._iKeyName];
        var cnt = methods.length;
        for(var i = 0; i < cnt; i++) {
            if(typeof(methods[i]) == "function") {
                methods[i].call(this, mods, tmpObj);
            }
        }
    }
    //Instrumentation code end
}
```

# Admin Extensions UI

- Integration Example (ZaServer.js):

(registering a funciton that submits Server object modifications to the server)

```
ZaServer.modifyMethod = function (tmpObj) {
    ...
    //actually send JSON request to the server
    var params = new Object();
    params.soapDoc = soapDoc;
    var reqMgrParams = {
        controller : ZaApp.getInstance().getCurrentController(),
        busyMsg : ZaMsg.BUSY_MODIFY_SERVER
    }
    var resp = ZaRequestMgr.invoke(params, reqMgrParams).Body.ModifyServerResponse;
    this.initFromJS(resp.server[0]);
}
ZaItem.modifyMethods["ZaServer"].push(ZaServer.modifyMethod);
```

# Admin Extensions UI

- Loading
  - via AjxInclude
  - single gzippped file in production mode (/service/zimlet/res/Zimlets-nodev_all.js.zgz)
  - one by one in dev mode
    - GetAdminExtensionZimletsRequest returns the list of zimlet objects
    - ZaSettings.init loads each JS and CSS file
    - current admin user has to have getZimlet permission to every admin extension that needs to be loaded in dev mode
  - errors in admin extensions can make Admin Console fail to load
  - usually follow the same MVC design pattern as the rest of Admin Console code
  - use hooks to add UI elements and extend data models

# Admin Extensions UI

Example:

- com_zimbra_backuprestore extension adds 4 new fields to Server object data model (com_zimbra_backuprestore.js)

```
if(ZaServer) {
    ZaServer.A_zimbraBackupMode = "zimbraBackupMode";
    ZaServer.A_zimbraBackupAutoGroupedThrottled = "zimbraBackupAutoGroupedThrottled";
    ZaServer.A_zimbraBackupAutoGroupedNumGroups = "zimbraBackupAutoGroupedNumGroups";
    ZaServer.A_zimbraBackupMinFreeSpace = "zimbraBackupMinFreeSpace";
    if(ZaServer.myXModel) {
        ZaServer.myXModel.items.push({id:ZaServer.A_zimbraBackupMode, ref:"attrs/" +
             ZaServer.A_zimbraBackupMode, type:_COS_STRING_});
        ZaServer.myXModel.items.push({id:ZaServer.A_zimbraBackupAutoGroupedNumGroups, ref:"attrs/" +
             ZaServer.A_zimbraBackupAutoGroupedNumGroups, type:_COS_NUMBER_});
        ZaServer.myXModel.items.push({id:ZaServer.A_zimbraBackupAutoGroupedThrottled, ref:"attrs/" +
             ZaServer.A_zimbraBackupAutoGroupedThrottled, type:_COS_ENUM_, choices:ZaModel.BOOLEAN_CHOICES});
        ZaServer.myXModel.items.push({id:ZaServer.A_zimbraBackupMinFreeSpace,ref:"attrs/” +
             ZaServer.A_zimbraBackupMinFreeSpace, type:_COS_STRING_});
    }
}
```

# Admin Extensions UI

Example:

- com_zimbra_backuprestore extension adds a tab with a bunch of fields to Global Config form

```
ZaHotBackup.GlobalConfigXFormModifier = function (xFormObject, entry) {
    var tabBar = xFormObject.items[1] ;
    ZaHotBackup.GlobalConfigTabIndex = ++this.TAB_INDEX;
    tabBar.choices.push({value:ZaHotBackup.GlobalConfigTabIndex , label:com_zimbra_backuprestore.BNR_Tab_BNR});

    var backupTabView = { ... }

    for (var i=0;i<xFormObject.items.length;i++) {
        if(xFormObject.items[i].type=="switch") {
            xFormObject.items[i].items.push(backupTabView);
            break;
        }
    }
};

if(ZaTabView.XFormModifiers["GlobalConfigXFormView"]) {
    ZaTabView.XFormModifiers["GlobalConfigXFormView"].push(ZaHotBackup.GlobalConfigXFormModifier);
};
```

# Admin Extensions Examples

Where they are hiding:

- com_zimbra_backuprestore: ZimbraBackup/src/zimlet/
- com_zimbra_cert_manager: ZaAdminExt/CertificateMgr/js/
- com_zimbra_delegatedadmin: ZimbraNetwork/ZimbraAdminExt/DelegatedAdmin/js/
- com_zimbra_viewmail: ZimbraNetwork/ZimbraAdminExt/com_zimbra_viewmail/js/
- com_zimbra_cluster: ZimbraCluster/src/zimlet/
- com_zimbra_hsm: ZimbraHSM/src/zimlet/
- com_zimbra_bulkprovision (aka we-based migration wizard): ZaAdminExt/BulkProvision/js/
- com_zimbra_adminversioncheck: ZimbraAdminVersionCheck/src/zimlet/
- com_zimbra_xmbxsearch (aka cross-mailbox search): ZimbraXMbxSearch/src/admin_extension/com_zimbra_xmbxsearch/
- com_zimbra_ucconfig (aka Voice/Chat service): ZimbraNetwork/ZimbraAdminExt/com_zimbra_ucconfig/js/

# Admin Extensions :: Server Components

- Most admin extensions require server extensions that implement additional SOAP request handlers:
  - VersionCheckRequest
  - InstallCertRequest
  - GetCertRequest
  - GenCSRRequest
  - BulkImportAccountsRequest
  - BulkIMAPDataImportRequest
  - GetXMbxSearchRequest
  - GetXMbxSearchesListRequest
- These are packaged in jar files and dropped into /opt/zimbra/lib/ext
- jetty picks them up, finds DocumentService implementation and calls registerHandlers method
- When server returns *UNKNOWN_DOCUMENT* that usually means that the server extension failed to load properly and SOAP handler was not registered
- Ask Server team for more info on how to implement server extensions

# Admin Extensions :: FOSS vs Network

- Nothing prevents a Network extension from being loaded in a FOSS
- Network Server extensions may not work in FOSS
- Network extensions that will work in FOSS just fine:
  - deletaged admin
  - view mail
- Network extensions that will load, but will not be fully functional in FOSS
  - cross mailbox Search
  - license
  - backup
  - HSM
  - convertd
  - cluster

# Dev environment

- To deply a dev build:
  $ant admin-deploy
- To deploy a production war file:
  $ant prod admin-deploy
- If you built web client, you have to run “clean” target before building admin war, otherwise the templates get messed up
- To make a production build load separate JS files instead of gzipped bundles add ?dev=1
- ?dev=1 also opens a popup window with debug trace, but it is not very useful – FireBug is far more useful
- default admin login/password is admin/test123
- do not use zimbra/test123 to log in to admin UI

# Areas to focus on next

- Client side data caching
  - expecially broken in Iron Maden
  - no good UI support for concurrent modification
- Managing asynchronous requests
  - still a few synchronous requests for no good reason
- Admin extensions framework
  - poorly written extensions break the UI
  - PS often stumbles trying to write extensinos because there is no good documentation
  - customers shoot themselves in the foot by installing 3rd party extensions
- Popup wizards need to go
- Talk to the users
- Add metrics/analytics

# Widescreen Test Pattern (16:9)