Writing Extensions For Zimbra Administration Console 5.0

Index

1 Extensions for Zimbra Administration Console

1.1 What is an extension for Zimbra administration console

Thanks to the flexibility of JavaScript, it is possible to modify nearly every aspect of the administration console through the mechanism of extensions. The framework allows developers to add new views to the administration console, manage new data objects in the administration console, extend existing objects with new properties, and customize existing views. Here are some examples of what can be done through this mechanism:

- Integrate Zimbra with Samba and manage user and machine accounts through Zimbra administration console
- Manage SSL certificates though Zimbra administration console
- Expose Webmin, Plesk, cPanel and WHM Control Panels functionality through Zimbra administration console

In this chapter I will provide examples of extensions to the administration console.

1.2 Installing extensions

Lets assume that you want to install zimbra_samba extension. This extension is included with ZCS 5.0 distribution package. You can install the extension using the administration console or using a zmzimletctl command.

1.2.1 Installing extensions using the administration console

Log in to the administration console, navigate to Admin Extensions and click “Deploy new” button in the tool bar. Click the “Browse” button in the wizard and find the extension ZIP file (in this case zimbra_samba.zip). Click “Deploy” button in the wizard.

1.2.2 Installing extensions from command line

Upload the extension ZIP file to your Zimbra server (zimbra_samba.zip is already available at /opt/zimbra/zimlets-admin-extra/zimrbra_samba.zip). Log in to the zimbra server box, change user to zimbra, and run zmzimletctl install command.

```
$> su zimbra
$> zmzimletctl install /opt/zimbra/zimlets-admin-extra/zimrbra_samba.zip
```

1.3 Understanding the components of an extension

An extension for Zimbra administration console consists of a UI component and an optional server component.

The UI component consists of the following files:

- Extension manifest

An XML file that describes the extension to the framework. The file should has the same name as the extension. I.e. for zimbra_samba extension, the XML file is zimbra_samba.xml.

- Configuration template

An XML file that describes configuration properties and default values.

- *.properties file with localized string constants. This file has the same nae as the extension. I.e. for com_zimbra_backuprestore extension's strings file is com_zimbra_backuprestore.properties.
- JavaScript files

The Server component of the extension may follow the server extension framework, but this is not a requirement. The server component can be a server-side script, or a servlet hosted on the same or on a different server.

1.4 Loading extensions

Extensions are loaded by the administration console UI after user authentication. If you want to know exactly how the process works, take a look at `ZaSettings.init` method.

First, the framework looks for the resource file that contains localized string constants. The resource file is generated at runtime at /zimbraAdmin/res/*name of the extension*.js. I.e. if the extension is zimbra_samba, then the resource file will be /zimbraAdmin/res/zimbra_samba.js. We will discuss resource files later in this chapter.

Next, the framework loads all the JavaScript files that are included in the extension. JavaScript files are loaded by dynamically creating <script> tags to the document. An important consequence of this method of loading extensions is that after each javaSript file is loaded, all statements in the file are executed by the browser.

1.5 Creating content of an extension

In this guide, we will focus on the UI component and the most generic and common uses of the admin extension framework:

- adding new views
- modifying existing views
- adding data objects
- extending existing data objects

1.5.1 Adding a list view

As you can see from the description of the framework, in order to add a list view to the Zimbra administration console, you need to create a list view class, and a list view controller class. Once the classes are implemented, you need a way to tell the administration console when to show the new list view. The easiest way to do this is to add a node to the navigation tree (aka Overview Panel) and show the new list view when the node is selected.

1.5.1.1 Creating a list view class

Lets consider the following task for this example. We want to retrieve a list of sambaDomain objects from an LDAP server and show the list in the administration console. List views are described previously in details in the section List Views. Here, I will give an example of a real list view class which is used in zimbra_samba extension.

Code example

```
function ZaSambaDomainListView(parent)  {
  var className = null;
  var posStyle = DwtControl.ABSOLUTE_STYLE;
  var headerList = this._getHeaderList();
  ZaListView.call(this, parent, className, posStyle, headerList);
  this._appCtxt = this.shell.getData(ZaAppCtxt.LABEL);
  this.setScrollStyle(DwtControl.SCROLL);
}

ZaSambaDomainListView.prototype = new ZaListView;
ZaSambaDomainListView.prototype.constructor = ZaSambaDomainListView;

ZaSambaDomainListView.prototype.toString = function() {
  return "ZaSambaDomainListView";
}

ZaSambaDomainListView.prototype.getTitle = function () {
  return "Manage Samba Domains";
}
/**
* Renders a single item as a DIV element.
*/
ZaSambaDomainListView.prototype._createItemHtml =
function(object, now, isDndIcon) {
  var html = new Array(50);
  var div = document.createElement("div");
  div[DwtListView._STYLE_CLASS] = "Row";
  div[DwtListView._SELECTED_STYLE_CLASS] =
    div[DwtListView._STYLE_CLASS] + "-" + DwtCssStyle.SELECTED;

  div.className = div[DwtListView._STYLE_CLASS];
  this.associateItemWithElement(object, div, DwtListView.TYPE_LIST_ITEM);

  var idx = 0;
  html[idx++] = "<table width='100%' cellspacing='2' cellpadding='0'>";
  html[idx++] = "<tr>";
  var cnt = this._headerList.length;
  for (var i = 0; i < cnt; i++) {
    var id = this._headerList[i]._id;
    if(id.indexOf(ZaSambaDomain.A_sambaSID)==0) {
      // name
      html[idx++] =
        "<td align='left' width=" + this._headerList[i]._width + "><nobr>";

      html[idx++] =
        AjxStringUtil.htmlEncode(object.attrs[ZaSambaDomain.A_sambaSID]);
      html[idx++] = "</nobr></td>";
    } else if (id.indexOf(ZaSambaDomain.A_sambaDomainName)==0) {
```

```
        // description
        html[idx++] =
          "<td align='left' width=" + this._headerList[i]._width + "><nobr>";
        html[idx++] =
           AjxStringUtil.htmlEncode(
            object.attrs[ZaSambaDomain.A_sambaDomainName]);

        html[idx++] = "</nobr></td>";
      }
      html[idx++] = "</tr></table>";
      div.innerHTML = html.join("");
      return div;
    }
}

ZaSambaDomainListView.prototype._getHeaderList = function() {
  var headerList = new Array();
  var sortable=1;
  headerList[0] =
    new ZaListHeaderItem(
      ZaSambaDomain.A_sambaDomainName,
      "Domain Name",
      null,
      200,
      null,
      ZaSambaDomain.A_sambaDomainName,
      true,
      true);

  headerList[1] =
    new ZaListHeaderItem(
      ZaSambaDomain.A_sambaSID,
      "sambaSID",
      null,
      null,
      null,
      ZaSambaDomain.A_sambaSID,
      true,
      true);

  return headerList;
}
```

1.5.1.2Constructing the view object

In most cases, the constructor of any view class will look like the constructor in this example. All the constructor does is initializing default values and calling the parent constructor. When creating a list view class, you can always use the same code for the constructor.

Next two lines after the constructor defines the inheritance of the class

```
ZaSambaDomainListView.prototype = new ZaListView;
ZaSambaDomainListView.prototype.constructor = ZaSambaDomainListView;
```

All list view classes must have ZaListView as their prototype. However, javascript does not have a built-in mechanism to restrict inheritance; and therefore, nothing prevents you from creating a list view class that does not have ZaListView class as a prototype. As long as the list view class correctly implements all the methods of ZaListView and DwtListView classes, it will work.

1.5.1.3Setting the title of the list view

The next method `ZaSambaDomainListView.prototype.getTitle` is called by the view manager when the user switches to this list view. String returned by this method will be displayed in the address area. In the screen shot below, getTitle method returned “Manage Accounts” string when I clicked on Accounts in the navigation panel.

The string that is returned by the getTitle method can be localized by using *.properties files.

1.5.1.4Rendering list items

The next method `ZaSambaDomainListView.prototype._createItemHtml` is called by the parent class DwtListView when it renders the list. This method has to return a DIV object, which has to be created by calling `document.createElement(“div”)`. `DwtListView` calls `_createItemHtml` method for each item that has to be displayed in the list, and places DIVs returned by `_createItemHtml` on the list view. In order to your list look similar to other lists in the administration console, we should use the same CSS classes that are used in this example:

```
div[DwtListView._STYLE_CLASS] = “Row”;
div[DwtListView._SELECTED_STYLE_CLASS] = div[DwtListView._STYLE_CLASS] +
“-” + DwtCssStyle.SELECTED;
div.className = div[DwtListView._STYLE_CLASS];
```

That will ensure that the list looks like other lists and that selected list elements are properly highlighted.

The next statement is important if you plan to implement actions for list elements, such as opening an object in a form view to show more details or edit, deleting an object, etc.

```
this.associateItemWithElement(object, div, DwtListView.TYPE_LIST_ITEM);
```

Next, in this example we iterate through `this._headerList` to output each cell of the column in the row. `this._headerList` is defined in the next method

`ZaSambaDomainListView.prototype._getHeaderList.`

This method returns an array of `ZaListHeaderItem` objects which are used to create the list header and define columns in the list view. If your list view has only one column, you do not have to implement this method.

1.5.1.5Creating a list view controller class

I will continue with the same example, showing a list of sambaDomain objects. If you look at the actaual ZaSambaDomainListController class, it has more functionality then what I will describe here. For simplicity, lets narrow the scope and assume that the tool bar for this list view has only one button: “Edit”, which will open the Xform view with details of the selected sambaDomain.

Code example

```
function ZaSambaDomainListController(appCtxt, container, app) {
  ZaListViewController.call(this,
  appCtxt,
  container,
  app,
  "ZaSambaDomainListController");
}

ZaSambaDomainListController.prototype = new ZaListViewController();
ZaSambaDomainListController.prototype.constructor =
ZaSambaDomainListController;
ZaController.initToolbarMethods["ZaSambaDomainListController"] = new
Array();

ZaSambaDomainListController.prototype.show = function(list) {
  if (!this._UICreated) {
    this._createUI();
  }

  if (list != null)
    this._contentView.set(list.getVector());

  this._app.pushView(this.getContentViewId());

  if (list != null)
    this._list = list;
}

ZaSambaDomainListController.initToolbarMethod = function () {
  this._toolbarOperations.push(
    new ZaOperation(
      ZaOperation.EDIT,
      ZaMsg.TBB_Edit,
      ZaMsg.SERTBB_Edit_tt,
```

```
      "Properties",
      "PropertiesDis",
      new AjxListener(this,
        ZaSambaDomainListController.prototype._editButtonListener)
      )
    );
}

ZaController.initToolbarMethods["ZaSambaDomainListController"].push(
ZaSambaDomainListController.initToolbarMethod);

ZaSambaDomainListController.prototype._createUI = function () {
  try {
    var elements = new Object();
    this._contentView = new ZaSambaDomainListView(this._container);
    this._initToolbar();
    if(this._toolbarOperations && this._toolbarOperations.length) {
      this._toolbar = new ZaToolBar(
                        this._container,
                        this._toolbarOperations);
      elements[ZaAppViewMgr.C_TOOLBAR_TOP] = this._toolbar;
    }

    elements[ZaAppViewMgr.C_APP_CONTENT] = this._contentView;
    var tabParams = {
      openInNewTab: false,
      tabId: this.getContentViewId(),
      tab: this.getMainTab()
     }
    this._app.createView(this.getContentViewId(), elements,tabParams);
    this._contentView.addSelectionListener(
      new AjxListener(this,this._listSelectionListener)
    );
    this._contentView.addActionListener(
      new AjxListener(this, this._listActionListener)
    );
    this._UICreated = true;
  } catch (ex) {
    this._handleException(ex,
      "ZaSambaDomainListController.prototype._createUI",
      null,
      false
    );
    return;
  }
}
```

1.5.1.6Constructing a list view controller object

The constructor is very simple, it has only one statement, which is to tall the parent constructor. Following the constructor are two statements that implement inheritance in JavaScript. The following statement is necessary only if you want your view to have a tool bar.

1.5.1.7Creating contents of a list view

Method `ZaSambaDomainListController.prototype._createUI` instantiates elements of the UI: tool bar and list view. After being instantiated, these elements are assigned to the main application controller which takes care of the layout

```
this._app.createView(this .getContentViewId(), elements,tabParams);
```

Object elements is a map where keys correspond to a predefined position of the element on the screen: `ZaAppViewMgr.C_APP_CONTENT` for the main content area and `ZaAppViewMgr.C_TOOLBAR_TOP` for the tool bar area. When we call **this**.`_app.createView` method, layout manager arranges the elements on the screen according to the application skin.

1.5.1.8Showing the list view

Next method is ZaSambaDomainListController.**prototype**.show. As the name suggests, this method is responsible for showing the view. The only argument that this method takes is the data object. There are no restrictions to what the data object is. In this case, it is an instance of ZaList. This statement **this**._contentView.set(list.getVector()); passes the data to the instance of the list view class. The following statement **this**._app.pushView(**this**.getContentViewId()); calls the main application controller's method “pushView” to push the DIV that contains this view to the top of the stack of divs that contain views, which makes the view visible.

1.5.1.9Defining a tool bar for a list view

```
ZaController.initToolbarMethods[”ZaSambaDomainListController”] = new
Array();
```

This statement initializes the array of method references in ZaController.initToolbarMethods map with the key that corresponds to this controller class. The key should be the same value that we passed to the parent constructor call in the constructor.

A tool bar is created by the `ZaController.prototype._initToolbar` method based on the **this**.`_toolbarOperations` array. The array consists of `ZaOperation` objects. Each `ZaOperation` object describes one button in a tool bar.

Code example

```
this._toolbarOperations.push(
    new ZaOperation(
    ZaOperation.EDIT,
    ZaMsg.TBB_Edit,
    ZaMsg.SERTBB_Edit_tt,
    “Properties”,
    “PropertiesDis”,
```

```
    new AjxListener(this,
      ZaSambaDomainListController.prototype._editButtonListener)
    )
);
```

`ZaController.prototype._initToolbar` calls methods referenced in `ZaController.initToolbarMethods["ZaSambaDomainListController"]` array. In this example we add a reference to `ZaSambaDomainListController.initToolbarMethod` method to this array.

1.5.2 Registering the list view controller

Before we add a handle to the new view to the navigation panel, we need to register the new controller with the main application controller (ZaApp). This can be done by adding a getter method to ZaApp class.

Code example

```
ZaZimbraAdmin._SAMBA_DOMAIN_LIST = ZaZimbraAdmin.VIEW_INDEX++;

ZaApp.prototype.getSambaDomainListController = function() {
  if (this._controllers[ZaZimbraAdmin._SAMBA_DOMAIN_LIST] == null) {
    this._controllers[ZaZimbraAdmin._SAMBA_DOMAIN_LIST] =
      new ZaSambaDomainListController(
        this._appCtxt,
        this._container,
        this
    );
  }
  return this._controllers[ZaZimbraAdmin._SAMBA_DOMAIN_LIST];
}
```

A global constant `ZaZimbraAdmin.VIEW_INDEX` is a counter that keeps track of how many view controller classes exist in the application.

In ZCS 5.0 all list view controllers are singletons, although this design pattern is not enforced by the framework and will be hard (if at all possible) to enforce in JavaScript. Therefore, instead of usual implementation of singleton pattern, where we would have a private constructor and a getInstance method, we let ZaApp.**prototype**.getSambaDomainListController method take care of maintaining only one instance of the controller class. All other objects should use this method to access the list view controller.

1.5.3 Adding A Node to The Navigation Panel

ZaOverviewPanelController class has API hooks similar to ZaModel, ZaController, and ZaTabView classes. Adding a node to the navigation panel is done in two steps. First, implement a method which will takes the navigation tree object as an argument and adds the new node to the tree. Second, add a reference to this method to ZaOverviewPanelController.treeModifiers array. In this example, we want to add a single node which will trigger the new list view described earlier.

Code example

```
Zambra.ovTreeModifier = function (tree) {
```

```
  if(ZaSettings.SYSTEM_CONFIG_ENABLED) {
    this._sambaTi = new DwtTreeItem(this._configTi);
    this._sambaTi.setText("Samba Domains");
    this._sambaTi.setImage("ZimbraIcon");
    this._sambaTi.setData(
      ZaOverviewPanelController._TID,
      ZaZimbraAdmin._SAMBA_DOMAIN_LIST
    );

    if(ZaOverviewPanelController.overviewTreeListeners) {

ZaOverviewPanelController.overviewTreeListeners[ZaZimbraAdmin._SAMBA_DOMAIN
_LIST] = Zambra.sambaDomainListTreeListener;
    }
  }
}

if(ZaOverviewPanelController.treeModifiers)
  ZaOverviewPanelController.treeModifiers.push(Zambra.ovTreeModifier);

Zambra.sambaDomainListTreeListener = function (ev) {
  if(this._app.getCurrentController()) {
    this._app.getCurrentController().switchToNextView(
      this._app.getSambaDomainListController(),
      ZaSambaDomainListController.prototype.show,
      ZaSambaDomain.getAll(this ._app)
    );
  } else {
    this._app.getSambaDomainListController().show(
      ZaSambaDomain.getAll(this ._app)
    );
  }
}
```

1.5.4 Adding a new tab to an existing form view

Adding a tab to an existing form view is the easiest task in creating an extension. All you needto do is to define the form elements for the new tab and add the tab to the tab bar. Following is a code snippet from zimbra_posixaccount extension that adds a tab to the Account view.

Code example

```
if(ZaTabView.XformModifiers["ZaAccountXFormView"]) {
  zimbra_posixaccount.AccountXFormModifier = function (xFormObject) {
    /* find the SWITCH element which is the parent element for all tabs */
    var cnt = xFormObject.items.length;
    var i = 0;
    for (i = 0; i <cnt; i++) {
      if(xFormObject.items[i].type=="switch")
```

```
  break; //index i now points to the SWITCH element
}
//find the index of the next tab
var posixTabIx = ++this.TAB_INDEX;
//tab bar is the element with index 1
var tabBar = xFormObject.items[1];
//add the new tab button to the tab bar
tabBar.choices.push({value:posixTabIx, label:"Posix Account"});
//define meta data for new form elements
var posixAccountTab = {
  type:_ZATABCASE_, numCols:1,
  relevant:("instance[ZaModel.currentTab] == " + posixTabIx),
  items: [
    {type:_ZAGROUP_,
      items:[
       {ref:ZaPosixAccount.A_gidNumber,
         type:_OSELECT1_, editable:false,
         choices:this._app.getPosixGroupIdListChoices(true),
         msgName:"Posix group",
         label:"Posix group",
         labelLocation:_LEFT_,
         onChange:ZaTabView.onFormFieldChanged},

       {ref:ZaPosixAccount.A_gidNumber,
         type:_TEXTFIELD_,
         msgName:ZaPosixAccount.A_gidNumber,
         label:ZaPosixAccount.A_gidNumber,
         labelLocation:_LEFT_,
         onChange:ZaTabView.onFormFieldChanged,
         cssClass:"admin_xform_number_input"},

       {ref:ZaPosixAccount.A_uidNumber,
         type:_TEXTFIELD_,
         msgName:ZaPosixAccount.A_uidNumber,
         label:ZaPosixAccount.A_uidNumber,
         labelLocation:_LEFT_,
         onChange:ZaTabView.onFormFieldChanged,
         cssClass:"admin_xform_number_input"},

       {ref:ZaPosixAccount.A_homeDirectory,
         type:_TEXTFIELD_,
         msgName:ZaPosixAccount.A_homeDirectory,
         label:ZaPosixAccount.A_homeDirectory,
         labelLocation:_LEFT_,
         onChange:ZaTabView.onFormFieldChanged,
         width:250},

      {ref:ZaPosixAccount.A_loginShell,
```

```
                type:_TEXTFIELD_, msgName:ZaPosixAccount.A_loginShell,
                label:ZaPosixAccount.A_loginShell,
                labelLocation:_LEFT_,
                onChange:ZaTabView.onFormFieldChanged,
                width:250}
          ]
        }
      ]
    };

     //add the new tab to the list of tabs
    xFormObject.items[i].items.push(posixAccountTab);
  }

  ZaTabView.XformModifiers["ZaAccountXFormView"].push(
    zimbra_posixaccount.AccountXFormModifier
  );
}
```

First, we define a modifier function

```
zimbra_posixaccount.AccountXFormModifier = function (xFormObject)
```

which will add new form elements to the existing form. Then we add the reference to the new modifier function to the array of modifier references for the Account view

```
ZaTabView.XformModifiers["ZaAccountXFormView"].push(zimbra_posixaccount.Acc
ountXFormModifier);
```

When the Account view is instantiated, all functions referenced in this array will be called.

The argument of the modifier function is an object that contains meta data for the XForm. The object is being passed to each modifier method.

1.5.5 Adding a Form View

Adding a form view is similar to adding a list view. You need to create a form view controller class, and a form view class, and then connect the form view controller either to a navigation tree or to a list view controller.

1.5.5.1Creating a form view class

Following the same example, we will create a form view class for displaying and editing a sambaDomain object.

```
function ZaSambaDomainXFormView (parent, app) {
  ZaTabView.call(this, parent, app,"ZaSambaDomainXFormView");
  this.initForm(ZaSambaDomain.myXModel,this.getMyXForm());
  this._localXForm.setController(this._app);
}

ZaSambaDomainXFormView.prototype = new ZaTabView();
```

```
ZaSambaDomainXFormView.prototype.constructor = ZaSambaDomainXFormView;
ZaTabView.XFormModifiers["ZaSambaDomainXFormView"] = new Array();


ZaSambaDomainXFormView.prototype.setObject =
function (entry) {
  this._containedObject = new ZaSambaDomain(this._app);
  this._containedObject.attrs = new Object();
  if(entry.id)
    this._containedObject.id = entry.id;
  if(entry.name)
    this._containedObject.name = entry.name;

  for (var a in entry.attrs) {
     if(entry.attrs[a] instanceof Array) {
       this._containedObject.attrs[a] = new Array();
       var cnt = entry.attrs[a].length;
       for (var ix = 0; ix < cnt; ix++) {
         this._containedObject.attrs[a][ix]=entry.attrs[a][ix];
       }
     } else {
       this._containedObject.attrs[a] = entry.attrs[a];
     }
   }

   if(!entry[ZaModel.currentTab])
     this._containedObject[ZaModel.currentTab] = "1";
    else
     this._containedObject[ZaModel.currentTab] = entry[ZaModel.currentTab];

   this._localXForm.setInstance(this._containedObject);
   this.updateTab();
}

ZaSambaDomainXFormView.prototype.getTitle =
function () {
  return "Samba Domains";
}

ZaSambaDomainXFormView.myXFormModifier = function(xFormObject) {
  xFormObject.tableCssStyle="width:100%;overflow:auto;";

  xFormObject.items = [
    {type:_GROUP_,
      cssClass:"ZmSelectedHeaderBg",
      colSpan: "*",
      id:"xform_header",
      items: [
```

```
        {type:_GROUP_,
         numCols:4,
         colSizes:["32px","350px","100px","250px"],
         items: [
          {type:_AJX_IMAGE_,
           src:"Domain_32",
           label:null},
          {type:_OUTPUT_,
           ref:ZaSambaDomain.A_sambaDomainName,
           label:null,
           cssClass:"AdminTitle",
           rowSpan:2},
          {type:_OUTPUT_,
           ref:ZaSambaDomain.A_sambaSID,
           label:"sambaSID"}
         ]
        }
  ],
   cssStyle:"padding-top:5px; padding-left:2px; padding-bottom:5px"},
{type:_TAB_BAR_,
 ref:ZaModel.currentTab,
 choices:[{value:1, label:ZaMsg.Domain_Tab_General}],
 cssClass:"ZaTabBar",
 id:"xform_tabbar"
},
{type:_SWITCH_,
 items:[
 {type:_ZATABCASE_,
  relevant:"instance[ZaModel.currentTab] == 1",
  colSizes:["250px","*"],
  items:[
    {ref: ZaSambaDomain.A_sambaDomainName,
     type:_TEXTFIELD_,
     label:ZaMsg.Domain_DomainName,
     onChange:ZaTabView.onFormFieldChanged
    },
    {ref: ZaSambaDomain.A_sambaSID,
     type:_TEXTFIELD_,
     label:"sambaSID",
     width:300,
     onChange:ZaTabView.onFormFieldChanged
    },
    {ref: ZaSambaDomain.A_sambaAlgorithmicRidBase,
     type:_TEXTFIELD_,
     label:"sambaAlgorithmicRidBase",
     cssClass:"admin_xform_number_input",
     onChange:ZaTabView.onFormFieldChanged
    }
```

```
        ]
      }
    ]
   }
  ];
}

ZaTabView.XFormModifiers["ZaSambaDomainXFormView"].push(ZaSambaDomainXFormV
iew.myXFormModifier);
```

1.5.5.2Constructing a form view object

The contructor in the previous example has three statements. A call to the parent contructor

```
ZaTabView.call(this, parent, app,"ZaSambaDomainXFormView");
```
in which we pass the name of the new class as to the fourth argument. A call to _initForm method.

```
this.initForm(ZaSambaDomain.myXModel,this.getMyXForm());
```
First argument in this statement is meta data that describes a ZaSambaDomain object, which will be displayed in this view; second argument is metadata that describes the elements of the form.

Third statement assigns a reference to the main application controller to the form instance.

```
this._localXForm.setController(this._app);
```
The purpose of this call is to let form elements access to utility methods of ZaApp instance.

1.5.5.3Passing data to the form

setObject method is responsible for passing data to the form. This method is called by the controller. Following statement passes the data to the Xfrorm

```
this._localXForm.setInstance(this._containedObject);
```
In this example, we create a local copy of the data (this._containedObject) in the view instance before passing it on to the form in order to be able to compare the data to the original state after it is modified by the user. Then we pass the local copy of the data to the XForm.

1.5.5.4Defining form fields

When we create a new form view, we define form fields similar to how we define them when modifying an existing form view. In order to define form fields, we need to create a modifier function

```
ZaSambaDomainXFormView.myXFormModifier = function(xFormObject)
```
and add a reference to this function to the appropriate array of function references.

```
ZaTabView.XFormModifiers["ZaSambaDomainXFormView"].push(ZaSambaDomainXFormV
iew.myXFormModifier);
```
Because this is a new view, the array of references is not initialized yet, so in order to avoid a null-reference error, we need to create the array before we push any references in it. A good practice is to define the array right after the constructor definitions at the top of the file

```
ZaTabView.XFormModifiers["ZaSambaDomainXFormView"] = new Array();
```

1.5.5.5Creating a form view controller class

It is recommended that a form view controllers class extends ZaXFormViewController class. This way, the new class gets access to all default implementations of the common methods of a form view controller.

```
function ZaSambaDomainController(appCtxt, container, app) {
  ZaXFormViewController.call(
    this, appCtxt, container,app,
    "ZaSambaDomainController");
  this._UICreated = false;
  this._toolbarOperations = new Array();
}

ZaSambaDomainController.prototype = new ZaXFormViewController();
ZaSambaDomainController.prototype.constructor = ZaSambaDomainController;

ZaController.initToolbarMethods["ZaSambaDomainController"] = new Array();
ZaController.setViewMethods["ZaSambaDomainController"] = new Array();

/**
* @method initToolbarMethod
* This method creates ZaOperation objects
* All the ZaOperation objects are added to this._toolbarOperations array
which is then used to
* create the toolbar for this view.
* Each ZaOperation object defines one toolbar button.
* Help button is always the last button in the toolbar
**/
ZaSambaDomainController.initToolbarMethod =
function () {
  this._toolbarOperations.push (
    new ZaOperation(
      ZaOperation.SAVE,
      ZaMsg.TBB_Save,
      ZaMsg.SERTBB_Save_tt,
      "Save",
      "SaveDis",
      new AjxListener(this, this.saveButtonListener)
    )
  );

  this._toolbarOperations.push(
    new ZaOperation(
      ZaOperation.CLOSE,
      ZaMsg.TBB_Close,
      ZaMsg.SERTBB_Close_tt,
      "Close",
     "CloseDis",
     new AjxListener(this, this.closeButtonListener)
```

```
    )
 );

}
ZaController.initToolbarMethods["ZaSambaDomainController"].push(
ZaSambaDomainController.initToolbarMethod);

/**
*@method setViewMethod
*@param entry - isntance of ZaSambaDomain class
*/
ZaSambaDomainController.setViewMethod =
function(entry) {
  if(entry.name && entry.id)
    entry.load("name", entry.name);

  if(!this ._UICreated)
    this._createUI();


  this._app.pushView(this.getContentViewId());
  this._contentView.setDirty(false);
/* setObject is delayed to be called after pushView in order to avoid
jumping of the view */
  this._contentView.setObject(entry);
  this._currentObject = entry;
}

ZaController.setViewMethods["ZaSambaDomainController"].push(
ZaSambaDomainController.setViewMethod);

/**
* @method _createUI
**/
ZaSambaDomainController.prototype._createUI =
function () {
  this._contentView = this._view =
    new ZaSambaDomainXFormView(this._container, this._app);

  this._initToolbar();

  this._toolbar = new ZaToolBar(
    this._container,
    this._toolbarOperations);

  var elements = new Object();
  elements[ZaAppViewMgr.C_APP_CONTENT] = this._contentView;
  elements[ZaAppViewMgr.C_TOOLBAR_TOP] = this._toolbar;
```

```
  var tabParams = {
     openInNewTab: true,
     tabId: this.getContentViewId()
  };
  this._app.createView(this.getContentViewId(), elements, tabParams);
  this._app._controllers[this.getContentViewId ()] = this;
  this._UICreated = true;
}

ZaSambaDomainController.prototype._saveChanges =
function () {

  var mods = new Object();
  for (var a in obj.attrs) {
    if(a == ZaItem.A_objectClass )
      continue;

    if(this._currentObject.attrs[a] != obj.attrs[a] ) {
          mods[a] = obj.attrs[a];
    }
  }

  try {
    this._currentObject.modify(mods);
    this.fireChangeEvent(this._currentObject);
  } catch (ex) {
    var detailStr = "";
    for (var prop in ex) {
      if(ex[prop] instanceof Function)
          continue;

      detailStr = detailStr + prop + " - " + ex[prop] + "\n";

    }
    this._handleException(ex,
      "ZaSambaDomainController.prototype._saveChanges",
      null,
      false);
    return false;
  }

  this._contentView.setDirty(false);
  return true;
}
```

1.5.5.6Constructing a form view controller object

The constructor is similar to list view controller constructor. Only the first statement in the constructor is required – the call to the parent contructor. The last argument of the parent constructor call is the String representation of the name of this class.

```
ZaXFormViewController.call(
    this, appCtxt, container,app,
    "ZaSambaDomainController");
```

1.5.5.7Creating contents of the form view

Method `ZaSambaDomainController.prototype._createUI` instantiates elements of the UI: tool bar and form view. After being instantiated, these elements are assigned to the main application controller which takes care of the layout

```
this._app.createView(this.getContentViewId(), elements, tabParams);
```

Object elements is a map where keys correspond to a predefined position of the element on the screen: `ZaAppViewMgr.C_APP_CONTENT` for the main content area and `ZaAppViewMgr.C_TOOLBAR_TOP` for the tool bar area. When we call `this._app.createView` method, layout manager arranges the elements on the screen according to the application skin.

1.5.5.8Registering a form view controller

Just like we had to register the list view controller, we also need to register the form view controller. However, in ZCS 5.0 form view controllers are not singletons. An instance of a controller is created for each open form. When the form is closed, the isntance is destroyed.

In order to register the form view controller we need to add another getter method to ZaApp class.

Code example

```
ZaApp.prototype.getSambaDomainController = function() {
  var c = new ZaSambaDomainController(
    this._appCtxt,
    this._container,
    this
  );

  var ctrl = this.getSambaDomainListController();
  c.addChangeListener(new AjxListener(ctrl, ctrl.handleChange));
  c.addCreationListener(new AjxListener(ctrl, ctrl.handleCreation));
  c.addRemovalListener(new AjxListener(ctrl, ctrl.handleRemoval));
  return c;
}
```

1.5.5.9Showing the form view

In order to show the form view, you need to call `show` method of the corresponding form view controller. In this example, we do not overwrite this method, because the default

implementation is sufficient. Default implementation of `show` method runs through all functions referenced in ZaController.setViewMethods["ZaSambaDomainController"] array. Therefore, we define `ZaSambaDomainController.setViewMethod` function and add it to the array.

1.5.5.10 Defining a tool bar for a form view

```
ZaController.initToolbarMethods["ZaSambaDomainController"] = new Array();
```

This statement initializes the array of method references in ZaController.initToolbarMethods map with the key that corresponds to this controller class. The key should be the same value that we passed to the parent constructor call in the constructor.

A tool bar is created by the `ZaController.prototype._initToolbar` method based on `this._toolbarOperations` array. First, `ZaController.prototype._initToolbar` calls functions referenced in ZaController.initToolbarMethods["ZaSambaDomainController"] array; next, it creates tool bar buttons based on the `ZaOperation` objects found in `this._toolbarOperations` array. Each `ZaOperation` object describes one button in a tool bar. In this example, the tool bar will have two buttons: Save and Close.

Code example

```
this._toolbarOperations.push (
  new ZaOperation(
    ZaOperation.SAVE,
    ZaMsg.TBB_Save,
    ZaMsg.SERTBB_Save_tt,
    "Save",
    "SaveDis",
    new AjxListener(this, this.saveButtonListener)
  )
);

this._toolbarOperations.push(
  new ZaOperation(
    ZaOperation.CLOSE,
    ZaMsg.TBB_Close,
    ZaMsg.SERTBB_Close_tt,
    "Close",
   "CloseDis",
   new AjxListener(this, this.closeButtonListener)
  )
);
```

1.5.6 Creating a new Model class

A model class performs several tasks. First, it is a convenient way of encapsulating read and write operations for retreiving data from the server and sending data to the server via HTTP using XML or JSON message format. Second, it contains meta data for the form (XModel). However, you do not have to create a Model class when creating an extension. All

communication with server can be handled by Controller classes. Xmodel can be just a JavaScript object which you can define anywhere in your extension.

Following is a `ZimbraSambaDomain` class from `zimbra_samba` extension. This code is the contents of ZaSambaDomain.js file.

```
if(ZaItem) {
  ZaItem.SAMBA_DOMAIN = "sambaDomain";
}

function ZaSambaDomain(app) {
  if (arguments.length == 0)
    return;

ZaItem.call(this, app,"ZaSambaDomain");
  this.type = ZaItem.SAMBA_DOMAIN;
  this.attrs = [];
  this._init(app);
}


ZaSambaDomain.prototype = new ZaItem;
ZaSambaDomain.prototype.constructor = ZaSambaDomain;

ZaSambaDomain.A_sambaSID = "sambaSID";
ZaSambaDomain.A_sambaDomainName = "sambaDomainName";
ZaSambaDomain.A_sambaAlgorithmicRidBase = "sambaAlgorithmicRidBase";

ZaItem.loadMethods["ZaSambaDomain"] = new Array();
ZaItem.initMethods["ZaSambaDomain"] = new Array();
ZaItem.modifyMethods["ZaSambaDomain"] = new Array();
ZaItem.createMethods["ZaSambaDomain"] = new Array()

ZaSambaDomain.loadMethod = function(by, val) {
  if(!val)
    return;

  var soapDoc = AjxSoapDoc.create(
    "GetLDAPEntriesRequest",
    "urn:zimbraAdmin",
     null
  );
  soapDoc.set("ldapSearchBase", Zambra.ldapSearchBase);
  soapDoc.set(
    "query",
    "(&(objectClass=sambaDomain)(sambaDomainName="+val+"))"
  );
  var getSambaDomainsCommand = new ZmCsfeCommand();
  var params = new Object();
  params.soapDoc = soapDoc;
```

```
      var resp =
getSambaDomainsCommand.invoke(params).Body.GetLDAPEntriesResponse.LDAPEntry[0];
      this.initFromJS(resp);
    }

    if(ZaItem.loadMethods["ZaSambaDomain"]) {
      ZaItem.loadMethods["ZaSambaDomain"].push(ZaSambaDomain.loadMethod);
    }

    ZaSambaDomain.prototype.initFromJS = function(sambaDomain) {
      ZaItem.prototype.initFromJS.call(this, sambaDomain);
      if(this .attrs && this.attrs[ZaSambaDomain.A_sambaSID])
        this.id = this.attrs[ZaSambaDomain.A_sambaSID];

      if(this .attrs && this.attrs[ZaSambaDomain.A_sambaDomainName])
        this.name = this.attrs[ZaSambaDomain.A_sambaDomainName];

    }

    ZaSambaDomain.prototype.remove = function(callback) {
      var soapDoc = AjxSoapDoc.create(
        "DeleteLDAPEntryRequest",
        "urn:zimbraAdmin",
         null
      );

      var dn = [[ZaSambaDomain.A_sambaDomainName,
        "=",
        this.attrs[ZaSambaDomain.A_sambaDomainName]
      ].join("")];

      if(zimbra_posixaccount.ldapSuffix)
        dn.push(Zambra.ldapSuffix);

      soapDoc.set("dn", dn.join(","));

      this.deleteCommand = new ZmCsfeCommand();
      var params = new Object();
      params.soapDoc = soapDoc;
      if(callback) {
        params.asyncMode = true;
        params.callback = callback;
      }
      this.deleteCommand.invoke(params);
    }

    ZaSambaDomain.getAll = function(app) {
      var soapDoc = AjxSoapDoc.create(
```

```
    "GetLDAPEntriesRequest",
    "urn:zimbraAdmin",
     null
  );
  soapDoc.set("ldapSearchBase", Zambra.ldapSuffix);
  soapDoc.set("query", "objectClass=sambaDomain");
  var getSambaDomainsCommand = new ZmCsfeCommand();
  var params = new Object();
  params.soapDoc = soapDoc;
  var resp =
    getSambaDomainsCommand.invoke(params).Body.GetLDAPEntriesResponse;
  var list = new ZaItemList(ZaSambaDomain, app);
  list.loadFromJS(resp);
  return list;
}

ZaSambaDomain.myXModel = {
  items: [
    {id:ZaSambaDomain.A_sambaSID,
      type:_STRING_,
      ref:"attrs/" + ZaSambaDomain.A_sambaSID
    },
    {id:ZaSambaDomain.A_sambaDomainName,
      type:_STRING_,
      ref:"attrs/" + ZaSambaDomain.A_sambaDomainName
    },
    {id:ZaSambaDomain.A_sambaAlgorithmicRidBase,
      type:_NUMBER_,
      ref:"attrs/" + ZaSambaDomain.A_sambaAlgorithmicRidBase
    }
  ]
};

ZaSambaDomain.createMethod = function(tmpObj, group, app) {
  //test
  var soapDoc = AjxSoapDoc.create(
    "CreateLDAPEntryRequest",
    "urn:zimbraAdmin",
     null
  );

  var sambaDomainName = tmpObj.attrs[ZaSambaDomain.A_sambaDomainName];

  var dn = [[ZaSambaDomain.A_sambaDomainName,
    "=",
    tmpObj.attrs[ZaSambaDomain.A_sambaDomainName]
  ].join("")];
```

```
  if(Zambra.ldapSuffix)
    dn.push(Zambra.ldapSuffix);

  soapDoc.set("dn", dn.join(","));
  tmpObj.attrs["objectClass"] = "sambaDomain";
  for (var aname in tmpObj.attrs) {
    if(tmpObj.attrs[aname] instanceof Array) {
      var cnt = tmpObj.attrs[aname].length;
      if(cnt) {
        for(var ix=0; ix <cnt; ix++) {
          if(typeof(tmpObj.attrs[aname][ix])=="object") {
            var attr =
              soapDoc.set("a", tmpObj.attrs[aname][ix].toString());
              attr.setAttribute("n", aname);
          } else {
            var attr = soapDoc.set("a", tmpObj.attrs[aname][ix]);
            attr.setAttribute("n", aname);
          }
        }
      }
    } else {
      if(tmpObj.attrs[aname] != null) {
        if(typeof(tmpObj.attrs[aname]) == "object") {
          var attr =
            soapDoc.set("a", tmpObj.attrs[aname].toString());

          attr.setAttribute("n", aname);
        } else {
          var attr = soapDoc.set("a", tmpObj.attrs[aname]);
          attr.setAttribute("n", aname);
        }
      }
    }
  }

  var testCommand = new ZmCsfeCommand();
  var params = new Object();

  params.soapDoc = soapDoc;
  resp = testCommand.invoke(params).Body.CreateLDAPEntryResponse;
  group.initFromJS(resp.LDAPEntry[0]);
}

if(ZaItem.createMethods["ZaSambaDomain"]) {
  ZaItem.createMethods["ZaSambaDomain"].push(ZaSambaDomain.createMethod);
}
```

```
/**
* @method modify
* Updates ZaSambaDomain attributes (SOAP)
* @param mods set of modified attributes and their new values
*/
ZaSambaDomain.modifyMethod =
function(mods) {
  var sambaDomainName = this.attrs[ZaSambaDomain.A_sambaDomainName];
  if(mods[ZaSambaDomain.A_sambaDomainName]) {
    sambaDomainName = mods[ZaSambaDomain.A_sambaDomainName];
    var soapDoc = AjxSoapDoc.create("RenameLDAPEntryRequest",
      "urn:zimbraAdmin",
       null
    );
    var dn = [[ZaSambaDomain.A_sambaDomainName,
      "=",
      this.attrs[ZaSambaDomain.A_sambaDomainName]
    ].join("")];

    var new_dn = [[ZaSambaDomain.A_sambaDomainName,
      "=",
      mods[ZaSambaDomain.A_sambaDomainName]
    ].join("")];

    if(Zambra.ldapSuffix) {
      dn.push(Zambra.ldapSuffix);
      new_dn.push(Zambra.ldapSuffix);
    }

    soapDoc.set("dn", dn.join(","));
    soapDoc.set("new_dn", new_dn.join(","));
    var renameLDAPEntryCommand = new ZmCsfeCommand();
    var params = new Object();
    params.soapDoc = soapDoc;
    resp =
      renameLDAPEntryCommand.invoke(params).Body.RenameLDAPEntryResponse;

    this.initFromJS(resp.LDAPEntry[0]);
    this._toolTip = null ;
  }

  var needToModify = false;
  for(var a in mods) {
    if(a == ZaSambaDomain.A_sambaDomainName) {
      continue;
    } else {
      needToModify = true;
      this._toolTip = null ;
```

```
    break;
  }
}

if(!needToModify)
  return;

//update the object
var soapDoc = AjxSoapDoc.create(
  "ModifyLDAPEntryRequest",
  "urn:zimbraAdmin",
   null
);
var dn = [[ZaSambaDomain.A_sambaDomainName,
  "=",
  this.attrs[ZaSambaDomain.A_sambaDomainName]
].join("")];

if(Zambra.ldapSuffix)
  dn.push(Zambra.ldapSuffix)

soapDoc.set("dn", dn.join(","));

for (var aname in mods) {
  //multy value attribute
  if(mods[aname] instanceof Array) {
    var cnt = mods[aname].length;
    if(cnt) {
      for(var ix=0; ix <cnt; ix++) {
        var attr = null;
        if(mods[aname][ix] instanceof String)
          var attr = soapDoc.set("a", mods[aname][ix].toString());
        else if(mods[aname][ix] instanceof Object)
          var attr = soapDoc.set("a", mods[aname][ix].toString());
        else if(mods[aname][ix])
          var attr = soapDoc.set("a", mods[aname][ix]);

        if(attr)
          attr.setAttribute("n", aname);
      }
    } else {
      var attr = soapDoc.set("a", "");
      attr.setAttribute("n", aname);
    }
  } else {
    var attr = soapDoc.set("a", mods[aname]);
    attr.setAttribute("n", aname);
  }
```

```
  }

  var modifyLDAPEntryCommand = new ZmCsfeCommand();
  var params = new Object();
  params.soapDoc = soapDoc;
  resp =
    modifyLDAPEntryCommand.invoke(params).Body.ModifyLDAPEntryResponse;

  this.initFromJS(resp.LDAPEntry[0]);
    return;
}

ZaItem.modifyMethods["ZaSambaDomain"].push(ZaSambaDomain.modifyMethod);


ZaApp.prototype.getSambaDomainSIDListChoices = function(refresh) {
  if (refresh || this._sambaDomainList == null) {
    this._sambaDomainList = ZaSambaDomain.getAll(this);
  }
  if(refresh || this._sambaDomainSIDChoices == null) {
    var arr = this._sambaDomainList.getArray();
    var sambaDomainArr = [];
    for (var i = 0 ; i < arr.length; ++i) {
      var obj = new Object();
      obj.name = arr[i].name;
      obj.id = arr[i].id;
      sambaDomainArr.push(obj);
    }
    if(this._sambaDomainSIDChoices == null) {
      this._sambaDomainSIDChoices = new XformChoices(
        sambaDomainArr,
        XFormChoices.OBJECT_LIST,
        "id",
        "name"
      );
    } else {
      this._sambaDomainSIDChoices.setChoices(sambaDomainArr);
      this._sambaDomainSIDChoices.dirtyChoices();
    }
  }
  return this._sambaDomainSIDChoices;
}


ZaApp.prototype.getSambaDomainNameListChoices =  function(refresh) {
  if (refresh || this._sambaDomainList == null) {
    this._sambaDomainList = ZaSambaDomain.getAll(this);
  }
```

```
  if(refresh || this._sambaDomainNameChoices == null) {
    var arr = this._sambaDomainList.getArray();
    var sambaDomainArr = [];
    for (var i = 0 ; i < arr.length; ++i) {
      var obj = new Object();
      obj.name = arr[i].name;
      obj.id = arr[i].id;
      sambaDomainArr.push(obj);
    }
    if(this._sambaDomainNameChoices == null) {
      this._sambaDomainNameChoices = new XformChoices(
        sambaDomainArr,
        XFormChoices.OBJECT_LIST,
        "name",
        "name"
      );
    } else {
      this._sambaDomainNameChoices.setChoices(sambaDomainArr);
      this._sambaDomainNameChoices.dirtyChoices();
    }
  }
  return this._sambaDomainNameChoices;
}
```

1.5.7 Adding properties to an existing model class

Lets assume that you extended your LDAP schema and added two new attributes to zimbraDomain object (zimbraDomainPrefMailSignatureEnabled and zimbraDomainPrefMailSignature), and you wrote some server code that uses these new attributes. It would also be nice if you could set/change values for these new attributes through the Zimbra Administration Console. The following example exaplains how to add this functionality.

First, lets declare constants for the new attributes.

```
// LDAP Attributes definition
ZaDomain.A_zimbraDomainPrefMailSignatureEnabled =
"zimbraDomainPrefMailSignatureEnabled";
ZaDomain.A_zimbraDomainPrefMailSignature = "zimbraDomainPrefMailSignature";
```

Next, we need to add the attributes to XModel for ZaDomain object, in order to be able to manage the values in a Domain XForm view.

```
// Add my new LDAP attrributes to ZaDomain.myXModel
if(ZaDomain.myXModel && ZaDomain.myXModel.items) {
    ZaDomain.myXModel.items.push(
          {id:ZaDomain.A_zimbraDomainPrefMailSignatureEnabled,
           type:_ENUM_,
           choices:ZaModel.BOOLEAN_CHOICES,
          ref:"attrs/"+ZaDomain.A_zimbraDomainPrefMailSignatureEnabled
```

```
            }
    );
    ZaDomain.myXModel.items.push(
        {id:ZaDomain.A_zimbraDomainPrefMailSignature,
         type:_STRING_,
         ref:"attrs/"+ZaDomain.A_zimbraDomainPrefMailSignature
        }
    );
}
```

Now, all that is left to do is to add the form elements to the Domain XForm. In section Adding a new tab to an exising form view.

1.5.8 Understanding resource files

Resource file of an extension is a text file with the name “name of the extension.properties”. I.e. for zimbra_samba extension, the resource file is zimbra_samba.properties. The purpose of the resource file is to allow developers to localize extensions. A resource file consists of name value pairs, where a name is a name of a constant, and value is plain text or a pattern. As mentioned earlier, a resource file is translated into JavaScript file by Zimbra server when a browser requests the resource. For example, if your resource file is zimbra_samba.properties, and it contains the following name-value pair

```
Domain_name_label = Samba domain name
```

The generated JavaScript file will be zimbra_samba.js and will contain the following code:

```
function zimbra_samba () {}
zimbra_samba. Domain_name_label = “Samba domain name”;
```

Thus, if you add resource files to your extension, instead of using strings for displayed text, you can use constants. In order to avoid null-reference errors, resource file is always loaded before all other JavaScript files of an extension.