Table of Contents

Programmers Reference, Client Side Plugins

Writing client-side plugins

In ***Aware IM*** you can not only add plugins for the server (such as custom processes, channels or functions), but you can also add plugins that execute on the client within a web browser. Most of the time you would write these plugins in order to add your custom user interface functionality, or modify the default ***Aware IM*** user interface behaviour.

All client-side plugins must be written in Javascript and in most cases you need the knowledge of the Kendo UI Javascript library from Telerik and a popular open source Javascript library called jQuery . The description that follows assumes that the reader is familiar with Javascript, Kendo UI library and jQuery.

There are several types of the client-side plugins you can add in ***Aware IM***:

1. Modify the default behaviour and presentation of forms
2. Modify the default behaviour and presentation of form sections in forms
3. Modify the default presentation of individual fields within forms
4. Modify the default presentation and behaviour of queries
5. Modify the default presentation and behaviour of content panels inside visual perspectives

We will look at each of these client-side plugins separately

Modifying default behavior of menu in visual perspectives

The idea here is very similar. You have two scripts available – initialization and render scripts. The initialization script has a chance to modify the configuration of the menu widgets (almost all menu types except Plain List are implemented by their own Kendo UI widget (see the table below). The render script can call the methods of the widget once it has been drawn.

The following objects are exposed to the initialization script:

1. “config” – this object represents Kendo UI configuration of the menu widget
2. “parser” – the controller object (AwareApp_VPParser) – see the code in the file

AwareIM/Tomcat/webapps/AwareIM/aware_kendo/parsers/vpParser.js

For example to add some custom menu item to a toolbar menu you could write the following script:

config.items.push ({

type: “button”,

spriteCssClass: “fa fa-edit”,

text: “My Menu Item”,

click: function () {

alert (“this is my menu item”);

}

});

Menu type	Kendo UI widget	Kendo UI reference
Toolbar	ToolBar	http://docs.telerik.com/kendo-ui/api/javascript/ui/toolbar
Standard Menu	Menu	http://docs.telerik.com/kendo-ui/api/javascript/ui/menu
Panel Bar	PanelBar	http://docs.telerik.com/kendo-ui/api/javascript/ui/panelbar
Tree	TreeView	http://docs.telerik.com/kendo-ui/api/javascript/ui/treeview

Modifying default behavior and presentation of content panels in visual perspectives

To modify the default behavior and presentation of content panels in visual perspectives you need to go to a particular visual perspective that you want to modify, select the content panel and then click on the “Scripts” property in the list of properties of the content panel.

The idea here is the same – you have two scripts as before. However, there are no Kendo UI widgets to modify here – you can only modify the markup of the content panel (either during initialization or after it has been drawn).

Two objects are exposed for the initialization script:

1. “config” – this is the configuration of the Aware IM “panel” object, the code of which is in AwareIM/Tomcat/webapps/AwareIM/aware_kendo/panel.js

The markup of the panel is stored in the “bodyContent” property of the object config.bodyContent. This is the markup you are most likely to modify here

“parser” object – this is the controller (AwareApp_Dashboard in AwareIM/Tomcat/webapps/AwareIM/aware_kendo/parsers/dashboard.js

AwareApp object

When writing advanced scripts as described above you can use the AwareApp Javascript object that contains some useful static methods. This is an example of calling one of these methods:

var panelId = AwareApp.getPanelId (“main”, “Accounts”, “My Accounts”);

The code of the object is located here: AwareIM/Tomcat/webapps/AwareIM/aware_ext/awareApp.js

The following methods can be used:

1. getPanelId (frameName, tabName, contentPanelName)

This method returns the unique id of the content panel in a visual perspective.

var id = AwareApp.getPanelId (‘main’, ‘Tab1’, ‘Content Panel1’);

Parameters:

frameName – name of the frame in the visual perspective that contains the panel

tabName – name of the tab inside the frame that contains the panel

contentPanelName – name of the content panel

1. getFramePanelId (frameName)

This method returns the unique id of the frame in a visual perspective.

var id = AwareApp.getFramePanelId (‘main’);

Parameters:

frameName – name of the frame in the visual perspective

1. getTabPanelId (frameName, tabName)

This method returns the unique id of the tab in a visual perspective.

var id = AwareApp.getTabPanelId (‘main’, ‘Tab1’);

Parameters:

frameName – name of the frame in the visual perspective that contains the panel

tabName – name of the tab inside the frame that contains the panel

1. isRTL ()

Return true if the current user uses right-to-left layout

1. getMainTabPanel ()

If a visual perspective has tabs return the tab panel holding the tabs.

1. getProcessId ()

Return the id of the currently running process or -1 if there are no processes currently running.

1. isTestingMode ()

Return true if the current user is running in the testing mode

1. startProcess, runQuery and other methods mentioned in the User Guide

Please refer to the “Links to Aware IM operations” section in the User Guide, that explains how to set up links to perform operations. All functions mentioned there can be used from your Javascript.

Using Javascript to integrate custom Cordova plugins for native mobile applications

Cordova plugins are components that provide access to some built-in features of mobile phones, (such as camera or contacts), for which there is no Javascript access. When components are integrated into the system these features become available through some special Javascript functions that the plugin makes available to the developer. Cordova plugins can only be used in native mobile applications.

Aware IM integrates some Cordova plugins out-of- the-box and provides rule actions that activate them (for example, MOBILE PUSH or MOBILE CAMERA SNAP INTO. However, there are many plugins around and it is impossible to integrate all of them into Aware IM.

Still there is a way to do this by adding some custom Javascript to your application. The following section explains how to do it.

This is the high level overview of what needs to be done:

1. Study the documentation of the Cordova plugin to fully understand Javascript methods that it exposes
2. Write the Javascript that calls the appropriate Javascript function that the plugin provides
    1. Give this function the data obtained from Aware IM if necessary. For example, read the data from the database and provide this data to the function of the plugin. The useful Aware IM function that can be used here is AwareApp.getObjectData()
    2. Handle the return of this function if necessary – for example write the data returned by the function to the database. Useful Aware IM functions for this are AwareApp.createOrUpdateObject() and AwareApp.startProcessWithInit()
3. Define panel operations or menu items in the mobile part of your business space version (using the Configuration Tool) that would run this Javascript. You should select an operation or menu item of the “Execute Javascript” type for this.
4. Build a native mobile application for your business space version using the “Build Native Mobile Application” command in the Configuration Tool. This will create a zip file.
5. Uzip this zip file somewhere. Find the config.xml file in the root of the unzipped application and open it for editing.
6. Find the section in this file that lists the plugins used by the application, for example:

<plugin name=“cordova-plugin-camera” spec=“2.0.0” />

Add the definition of the Cordova plugin you need to integrate – look up the documentation of the plugin for details of the plugin name and version number

1. Zip up the application again and use the PhoneGap build to create application files in the native format of the mobile phone

Let’s look at an example. We will be integrating a Cordova plugin for Contacts into the CRM mobile sample application.

The documentation of the plugin can be found here:
https://github.com/apache/cordova-plugin-contacts

As we can see the plugin provides the navigator.contacts object that can be used to create contacts,

find existing contact or pick a particular one. Let’s add the following functionality to the CRM application:

1. From the form of a customer or from a customer list create a phone contact populated with the information from the customer record in the application
2. Pick a contact from the list of phone contacts and send this contact an email that includes some information stored in the application

Creating a contact on the phone

We need to use the “create” method of the navigator.contacts object and provide contact data available in the customer record that we are parked on. Retrieving the data can be done using the AwareApp.getObjectData function. It has the following signature:

getObjectData: function (objectName, objectId, callBackFunction)

ObjectName and objectId identify the record to retrieve and callBackFunction specifies a function that will be called when the data has been retrieved. The function will be called with the object storing the retrieved values.

How do we get object name and id? When we define an Aware IM operation of the “Execute Script” type Aware IM automatically defines the following objects that we can use in our Javascript:

1. parser
2. context

The parser object should be already familiar and the context object stores an array of objects with objectName and objectId attributes. The record we are parked on is the first and only one in this array. So to get objectName we use the following context[0].objectName; and to get object id we use context[0].objectId

So the Javasrcipt we need to write to create a contact looks like this:

```
AwareApp.getObjectData (

context[0].objectName,

context[0].objectId,

function (objectData)

{

navigator.contacts.create ({

“displayName”: objectData[“FirstName”] + “ “ + objectData[“LastName”],

“birthday”: kendo.parseDate (objectData[“DateOfBirth”], “dd/MM/yyyy”, “en-US”)

});

}
```

);

Note that here “displayName” and “birthday” are the names of the attribute of the Contact object on the phone exposed by the plugin, whereas “FirstName”, “LastName” and “DateOfBirth” are the names of the attributes of the Customer object in the CRM application. Note also that all Aware IM attribute values are strings and if the plugin requires some other type (for example, date), then the strings need to be converted to the appropriate type.

The next step is to create operations of the “Execute Script” type to the form and customer list. We can add a panel operation to the “Editing Mobile” form of the Customer object and an operation with record to the “Customer – all mobile” query. We then specify the above script as a parameter of the operation.

Send email to the selected contact

We need to use the pickContact method of the navigator.contacts object to display a list of contacts, let the user pick one and then we need to start a process in the application to send an email to the email address of the contact picked by the user.

The email address returned by the plugin needs to be saved in some temporary object and then this object can be used in the process. So we will create a temporary business object (persistence type – memory) called ContactParam with the single EmailAddress attribute. We will then create a process called SendEmailToContact with the ContactParam object as its input. The process will then use the SEND action to send any email to this email address (the email can use tag expressions to retrieve some information from the system – for example, from SystemSettings or from the logged in user).

To start a process we will use the AwareApp.startProcessWithInit function. It has the following signature:

startProcessWithInit: function (procName, renderOption, objName, initValues, context)

Here procName is the name of the process to start, renderOption is where to display the results of the process (we can use null), objName is the name of the parameter object, initValues is the object storing values of the parameter object and context contains additional parameter objects (null in our case)

So our Javascript can look like this:

navigator.contacts.pickContact (function (contact)

{

var email = contact.emals[0].value;

AwareApp.startProcessWithInit (

“SendEmailToContact”,

null,

“ContactParam”,

{ “EmailAddress” : email }

},

function (error { console.log (error); }

);

Then we just need to add a command of the “Execute Script” type to the mobile menu of the application.