HP Anywhere

Windows

Software Version: 10.11

Events Sample App Cookbook

Document Release Date: January 2014

Software Release Date: December 2013

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

Chapter 1

'Events' Sample App Cookbook Overview

This cookbook show you how to create a simple business app to manage events in your calendar and enable users to collaborate on a specific event. The recipes show you how to get started, as well as introducing advanced platform capabilities, including:

- Connecting to the backend system
- Defining user and administrator settings
- Collaboration
- Localization
- Toolbar customization
- Application state
- Recommended apps and participants

Intended Audience

This document is intended for developers that use the HP Anywhere API to build business applications that run within the HP Anywhere framework, and leverage HP Anywhere capabilities. This cookbook describes the development of a specific business application in detail.

It includes steps such as:

- The basic requirements needed to start app development
- How to read data from the backend system
- Guides for implementation
- Specific HP Anywhere capabilities: participants, activities, recommended apps

After reading this guide, you should be able to implement a business application using the HP Anywhere API.

Prerequisites

Before using this cookbook, it is recommended that you familiarize yourself with:

- HP Anywhere User Guide and related movies
- HP Anywhere IDE guides (for Eclipse or IntelliJ IDEA), and related movies

Chapter 2

Events App Functional Description

The basic cookbook application is **Events**, which is a simple business application to manage events in your calendar.

The Events app includes the following functionality:

- **View Events list**, which is a simple list display for each event and contains the event name, event date, importance, and owner name. Events are sorted by date or name, according to user setting. From this list you can add a new event by tapping on the **Add** icon.

  In the **More**  menu, when you view the Event list, two new customized options are added: **About** and **Do something**.

- **When a user adds a new post** while in the View Events list page, the app page is added to the activity's context.
- **When a user adds a new post** while in the View/Edit Event page, the event is added to the activity's context.
- **Create New Event**, which enables users to create an event that includes the following details:
  - **Date**. Default is the current date
  - **Name**. String, no default
  - **Importance**. High or regular.
  - **Owner (read-only)**. Automatically set according to user.
- **Delete Event**, which enables users to delete an event.
- **View/Edit Event**, which enables users to view or edit an event. Available from the landing page.

  In this view, the **Delete** button is added to the toolbar.
- **All Apps**, shows the **View Events list** and the **New Event** pages.
- **Recommended Apps**, shows the **View Event** page when a context object exists.
- **Recommended Participants**, shows a user with an email address, **support@example.com**.
- **Automatic Creation of Activity,** when a new event is created (by the user), uses the Handle

Event API to automatically create an activity for the user, with the event as context object, and a message “A new event was created : <event name>”.

a. The notification channel is “Front Page”.

b. This functionality is controlled by the admin setting **Create activity on new event (yes/no)**.

When a user clicks the activity, the View Event entry point opens.

Chapter 3

How to Open the Events App Code in HP Anywhere IDE

To open the code of the Events App in HP Anywhere IDE:

1. Download the <*Events_app*>.zip file from the Dev Zone.
2. Extract the source code from the zip file into your Eclipse workspace folder.
3. Open Eclipse.
4. From the **File** menu, select **Import > Existing Projects into Workspace** and click **Next.**
5. Click the **Browse** button next to **Select root directory**, and browse to the folder into which you extracted the source code.
6. Click **Finish**.

The Events app is now available for use from the Eclipse IDE Plugin.

Note for Eclipse users:

To sync Eclipse dependencies with a third-party dependency in your app's **pom.xml** file:.

1. Open a command line (CLI).
2. Change the directory to the root directory of your app.
3. Run the following command: **%btoa_home%\apache-maven-3.0.4\bin\mvn eclipse:eclipse**
4. In the project tree, right-click the app node and select **Refresh** to refresh the app project in HP Anywhere IDE.

Chapter 4

Recipes

This section describes the basic process to follow when developing apps and contains the following recipes:

All the examples in the recipes are from the **Events** sample app.

How do I start?

This section describes how to implement your new app. Before you begin, make sure to create the new Event app project, so that you are ready to start the implementation.

The first step is to add some mandatory files, optional classes, and configuration files to your project as described in the sections below:

- Implement the Main App Service
- Add the App Descriptor
- Define the Data Source Provider
- Implement the REST Service for the App
- Add Localization Support

Implement the Main App Service (Mandatory)

To register an app in the HP Anywhere framework, the app must implement the **BaseBTOService** interface.

The HP Anywhere framework provides an abstract class (AbstractBTOServiceEE) that must be extended for the app service. The name of the Java Bean is the same as the app service ID that is defined in the **descriptor.xml** file (described in "How do I start?" above

In the example below, EventsApp extends AbstractBTOServiceEE, and the app service ID is **Events**:

```
@Service("events")
public class EventsApp extends AbstractBTOServiceEE {

    @Override
    public Collection<EntryPointDefinition> getEntryPointDefinitionsByContext
(Collection<BTOContext> btoContexts, Collection<EntryPointDefinition> entryP
ointDefinitions) throws CustomException { . . .  }

    @Override
    public Collection<EntryPointDefinition> getSupportedEntryPointDefinition
sNoContext() throws CustomException { . . . }

    @Override
    public Collection<UserProfile> getRelatedUsers(Collection<BTOContext> bt
oContexts) throws CustomException { . . .    }
    @Override
    public DataSourceProvider getDataSourceProvider() { . . .   }
```

```
}
```

Add the App Descriptor (Mandatory)

Each app includes an app descriptor file. This mandatory file includes basic app metadata, such as the form factors the app supports, the web resources relevant per form factor, and the app ID and version.

You define the app descriptor in the **<app-id>-descriptor.xml** file.

> **Note: app-id** must be unique for each app.

This step, along with the step described in "Implement the Main App Service (Mandatory)" on the previous page define a basic app.

For more details on the descriptor file, see the *HP Anywhere API Reference*.

Define the Data Source Provider

You can specify how your app is connected to its legacy backend system using a data source provider. In most cases, the legacy backend is accessed via REST, so the connectivity definition includes properties such as host, port, and protocol.

> **Note:** Not all apps use a backend system. Therefore, this step is not mandatory.

You implement the data source provider in any of the following ways:

- **Use the dataSourceProvider interface**

  HP Anywhere provides a class named **WSDataSourceProvider** that can be extended. At minimum, you must supply the data source name.

- **Add the DataSourceProvider.xml file (which includes the list of properties) to your app**

  Upon deployment of the app on the HP Anywhere server, from the Administrator Console, its data source information is exposed. The admin user then configures the data source values from the Administrator Console, such as the host FQDN, port, and the protocol to access the backend system.

After the data source is configured, the app can compose the URL to its backend, using the dataSourceService that the HP Anywhere framework provides to get the data source information.

For details on the DataSourceProvider.xml, see the *HP Anywhere API Reference*.

In the Events app, the data source provider is defined in the **EventsApp-ds-provider.xml** file. In this file, the **dataSourceName** and **serviceId** parameters are mandatory. You define the remaining parameters as required in the app.

```
<dataSourceProvider xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <dataSourceName>events-DS</dataSourceName>
    <serviceId>events</serviceId>
    <!--Optional:-->
    <configProperties>
        <!--Zero or more repetitions:-->
        <configProperty>
            <name>HostName</name>
            <type>STRING</type>
            <value>localhost</value>
            <isRequired>true</isRequired>
            <propValues></propValues>
        </configProperty>
        <configProperty>
            <name>Port</name>
            <type>INTEGER</type>
            <value>8080</value>
            <isRequired>true</isRequired>
            <propValues></propValues>
        </configProperty>
        <configProperty>
            <name>Protocol</name>
            <type>ENUMERATION</type>
            <value>http</value>
            <isRequired>true</isRequired>
            <propValues>http</propValues>
            <propValues>https</propValues>
        </configProperty>
    </configProperties>
    <!--Optional:-->
        <!--Zero or more repetitions:-->
    <testProperties>
        <!--Zero or more repetitions:-->
        <configProperty>
            <name>string</name>
            <type>string</type>
            <value>string</value>
            <isRequired>true</isRequired>
            <propValues>string</propValues>
        </configProperty>
    </testProperties>
</dataSourceProvider>
```

Each **dataSourceProvider** can define one or more **configProperty** elements. In the example above, you need a URL to access the event legacy backend. Therefore, the properties required are host name, port, and protocol.

Implement the REST Service for the App

For the app client to work with its legacy backend, the app generally includes a REST service that is used for communication between the HP Anywhere server side and client side. For details on HP Anywhere architecture, see the HP Anywhere architecture section in the *HP Anywhere Administrator Guide*.

Example: In the following example, the REST service is responsible for supporting REST APIs that add, update, and delete events. The class is **EventsRestService**.

```
@Service
@Path("/events")
@Produces("application/json;charset=utf-8")
public class EventsRestService {

    @GET
    public List<EventVo> getAll() { . . . }

    @GET
    @Path("{id}")
    public EventVo getEvent(@PathParam("id") String id) { . . . }

    }

    @POST
    @Consumes("application/json;charset=utf-8")
    public EventVo addEvent(EventVo event) { . . .  }

    @PUT
    @Path("{id}")
    @Consumes("application/json;charset=utf-8")
    public EventVo updateEvent(@PathParam("id") String id, EventVo eventVo)
{ . . . }
    @DELETE
    @Path("{id}")
    public void deleteEvent(@PathParam("id") String id) { . . .  }

}
```

Add Localization Support

Each app must provide a resource bundle file that includes a map of all localization keys with their translation per locale. The translation files name is composed of the **serviceId_locale.properties**.

Example: The **events_en.properties** example below displays the translation to the notification type properties in the admin setting xml. These translations appear in the Admin UI settings:

```
sections.notification.types.name=Notification Types

settings.notification.type.new.event.name=New Event

settings.notification.type.new.event.desc=Describe the notification channels
for new event
```

In addition, the HP Anywhere app framework provides the following methods to obtain the translated value of existing keys:

UserInfoService:getLocale	Find the locale of the current user session.
AbstractBTOServiceEE:getString	Retrieve the translated key.

In the following example, the **getString** API is used to retrieve the current locale.

```
@Autowired
private UserInfoService userInfoService;

. . .

// translate the given string according to current user’s locale
   String output = this.getString(userInfoService.getLocale(),"events.servic
e.name");
```

How do I start writing the client side of an app?

I'm developing the client side of a new app. How do I start?

You develop your app in JavaScript. Your app runs on a user's device in an HTML iframe and communicates with HP Anywhere through a JavaScript API. API usage is explained throughout this document.

You do not need to define HTML markup for your app. Instead, you declare your dependencies in a descriptor.xml file, and HP Anywhere injects them into your iframe. Dependencies may be JavaScript files, CSS files, and so on. For performance reasons, it is highly recommended that you provide your app in a minified form. You can use any JavaScript framework (for example Sencha Touch, Enyo JS), CSS pre-processor (for example SCSS, LESS), and other tools (or none at all).

When your app is ready for user interaction, it must call **HPA.Framework.setReady()** with the single Boolean argument, 'true'.

Your app must implement the top-level function—**openEntryPoint (entryPointName, params, callback, scope)**. The first argument, **entryPointName**, is mandatory. **params**, **callback**, and **scope** are optional. HP Anywhere calls this function whenever your app starts running.

A typical implementation of this function includes a switch statement on the given entry point, with a case clause for each possible entry point name. Your app must support a default entry point named **OpenEntryPoint**.

See **Loader.js** for the app's implementation of **openEntryPoint()**.

How do I get current user info?

I want to use the current user info in my app, how can I get it?

You can retrieve the current user data by calling **HPA.Profile.getInfo()**. The returned object contains the ID, display name, e-mail, job title, origin (internal/external) and several URLs for the user image in different sizes. In the example below, the display name is used.

Example

The following code in the **EventDetails.js** file retrieves the current user's display name to set a new event owner:

```
HPA.Profile.getInfo().displayName
```

How do I add a business entity to an activity context upon a user post?

You can notify the HP Anywhere framework about the context object you are currently working on using the following API:

```
HPA.Framework.setCurrentContext ([context], entryPointObject)
```

where **context** is a single context object or array of context objects, and **entryPointObject** is the entry point into the app.

Each time a user posts a message (collaboration), the framework tries to attach the current context object of the app in focus. There may be no context objects or any number of context objects. You can also add an optional entry point.

To notify the framework, do one of the following:

- Update the framework when the current context of your app changes.
- You can pass an array of one or more context objects to the framework using **HPA.Framework.setCurrentContext()**.

Example

The following code passes an Event item as the current context object:

```
var contextobject = HPA.Framework.createContextObject(eventData.id, 'EVENT',
                                                       eventData.title, '');
HPA.Framework.setCurrentContext([contextobject]);
```

How do I add an app page to an activity context upon a user post?

You can pass an app page to the framework, so that when a user posts a new message, the framework attaches the given app page to the current activity and notifies the framework about the current context of the app.

This is done using the **setCurrentContext** method. This method is called by the app developer whenever he/she wants to notify the framework that the current context has changed.

To do this:

- Use the **HPA.Framework.createEntryPointObject** factory method with the following parameters:

<AppId>	A string specifying the requested app id
<entryPointId>	A string specifying the requested app page id
<displayName>	A title to be presented in the activity's context view

Example

The following code passes the app page name EP_LIST of the Events app as the current context:

```
var entryPointObject = HPA.Framework.createEntryPointObject('events',
                       'EP_LIST', HPA.I18n.localize('EP_LIST'));
                       HPA.Framework.setCurrentContext([], entryPointObject);
```

How do I save a state and reopen the activity later with this state?

I want the app to open at the same location the next time I use the activity. How do I define this and when does it open?

A state may be any JavaScript value. The state is saved per activity, per user, and per entry point (app page). You may save a state at any time. When the user returns to an activity, the last saved state of each entry point is provided to that app page.

A state may include, for example, the context object the user last worked on, or the stage in a workflow the user last completed. In the example below, the state is the view the user was in (list or edit/new), and when performing an edit, the state is the data of the event last edited.

You save the state by calling **HPA.Framework.saveState()** with your state as the parameter.

You retrieve the state in the **openEntryPoint** implementation from **params.state** (params is the second argument).

Example

The following examples show how to save and restore states in an app.

In the **EventContainer.js** file, inside **listEvents**:

```
HPA.Framework.saveState({ viewMode: 'list' });
```

In the **EventContainer.js** file, inside **editEvent**:

```
HPA.Framework.saveState({ viewMode: 'edit', data: eventdata });
```

In the **Loader.js** file, inside **openEntryPoint()**:

```
if ( params.state && !!params.state.viewMode )
   {
     if ( params.state.viewMode === 'list' ){
            myApp.listEvents();
   }
     else if ( params.state.viewMode === 'edit' )
        {
            myApp.editEvent(null, state);
    }
  }
```

How do I dynamically customize the options in the toolbar (Smartphone only)?

How can I add my own action icon to the toolbar?

Pass an object in the following structure:

```
{icon: 'Add', action: "defaultAction", params: "add"},
```

The Action icon can be one of the following predefined icons: **Add, Attach, Delete, Edit, Photo, Remove, Save, Search, Store**.

params is an optional object or string in any format.

For example, the code above adds the Add icon to the toolbar:

How can I add items to the More Options menu?

Specify the text label and pass optional params as required.

Example

```
HPA.Framework.ActionsBar.setActions([
  {icon: 'Add', action: "defaultAction", params: "add"},
  {text: "About", action: "moreAction", params: "Create, edit and view event
s" },
  {text: "Do something", action: "moreAction", params: "Did something..." }
]);
```

How can I revert to the default actions toolbar?

Use the following code:

```
HPA.Framework.ActionsBar.setActions();
```

How do I hide/show the actions toolbar entirely?

Use the following code:

```
HPA.Framework.ActionsBar.setVisible( false );
```

How do I define and use user settings?

How can I make specific settings available to my users, where each user has his/her own customized settings?

Locate the file **<App ID>-user-settings.xml** in your app's files. This file contains your app's user settings.

Edit this file using an XML editor (or text editor), and add your settings. Make sure you use the UserSettings.dtd file to verify that your XML file is valid and compliant. Some XML editors, such as the one built into the Eclipse IDE, provide a convenient way to build your XML from "compliant elements".

Each setting can be boolean, integer, enumeration and more. You can define default values as well as restrictions, for example, a maximum value.

For each setting, you define a name, a name key and a description key. These keys will be used to retrieve the name and description for a given user language and/or locale. You also provide keys in textual setting options. Settings are grouped into named contexts (sections).

To change their own settings, users can access the Settings section of HP Anywhere by clicking on  .

In your app's xml file, define a section with the app name add the app settings in this section. Each type of setting may be rendered differently, for example a boolean setting may be rendered as a toggle button.

You access the settings of the current user through openEntryPoint's second argument, **params**. **Params.settings** contains your app's user settings with the values selected by the current user.

Example: Defining and using a user setting for events list ordering

In the **events-user-settings.xml** file, define:

```
<enumeration
        enum="events.sortSettings.sortBy.title,events.sortSettings.
               sortBy.date">
                events.sortSettings.sortBy.title
</enumeration>
```

In the **events_en.properties** file, define:

```
events.sortSettings.sortBy.title.name.key=Event Name
events.sortSettings.sortBy.date.name.key=Event Date
```

In the **Loader.js** file, define:

```
var sortOrderSettingValue = params.settings['events.sortSettings']
                                ['events.sortSettings.sortBy'][0];
```

How do I adapt my app for different locales?

I want my app to display user messages according to the current user's locale. How can I provide different text for each locale?

Under the **I10n** folder, create a file for each supported locale with the file name <App ID>_<locale code>.properties. For example, events_en.properties for English texts, events_de.properties for German, and so on. The keys are identical in all files, and the values of the text parameters are in the appropriate language.

HP Anywhere loads only the single file for the current user locale.

You access the value for a given key using **HPA.I18n.localize('key')**.

Example: Defining a key value and using the key to retrieve the value

In the **events_en.properties** file, define:

```
events.listTitle=My Events
```

In the **events_de.properties** file, define:

```
events.listTitle=Meine Ereignisse
```

In the **EventsList.js** file, define:

```
HPA.I18n.localize('events.listTitle') // Returns 'My Events' or
          'Meine Ereignisse', depending on user locale.
```

How do I create a context object and prompt the user to add it to an activity?

My app just generated a context object. How do I prompt the user to add this context object to the current activity or add it to a new activity?

1. Create a context object by calling **HPA.Framework.createContextObject()**.
2. Pass your object's ID, data type, display name and metaData (optional) as parameters to this function. (All these parameters are strings.)

   HPA.Framework.createContextObject() returns a JavaScript object if it created a valid contextObject, otherwise null.
3. If it created a valid context object, you can add it to an activity by calling **HPA.Framework.addContextObjectToActivity()** function, passing your object as a parameter.

The user will be prompted to select whether to add the object to the current activity or add it to a new activity. The user can also dismiss the prompt without selecting an option.

Example: Share an event by creating a context object and adding it to an activity

In the **EventContainer.js** file, inside **shareEvent()**:

```
// Create new context object via HPA.Framework.createContextObject
     factory method
var ctx = HPA.Framework.createContextObject(newData.id, "EVENT",
      newData.title, JSON.stringify(newData));
// Share the newly created context object
ctx && HPA.Framework.addContextObjectToActivity(ctx);
```

How do I read data from the backend system using REST?

In the Events app, the backend can be accessed using the REST API. Using these APIs, you can:

- View all events
- View event (view a single event)
- Create a new event
- Update an existing event
- Delete an event

The backend REST server is represented by the **LegacyEventsEmulatorService** class. This class is added to the app as an emulator for a real backend system.

LegacyEventsEmulatorService exposes the following REST API:

1. View all events

Method	GET
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator
Body	Empty

2. View event

Method	GET
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator/{id}
Body	Empty

3. Create new event

Method	POST
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator
Body	JSON representation of EventVo

For example:

```
{
"id":"1362474882964",
"owner":"admin admin",
"date":"2013-01-01",
"importance":"LOW",
"title":"my event6"
}
```

4. Update existing event

Method	PUT
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator/{id}
Body	JSON representation of EventVo

For example:

```
{
"id":"1362474882964",
"owner":"admin admin",
"date":"2013-01-01",
"importance":"LOW",
"title":"my event6"
}
```

5. Delete event

Method	DELETE
URL	http://backend.fqdn:8080/events-web/services/backendLegacyEventsEmulator/{id}
Body	empty

Reading Data from the Backend Legacy Server

To read data from the backend legacy server, follow the steps below:

1. Get the Events app data source information to create backend base URL

The data source information is managed by the administrator via the Admin UI.

> **Note:** The data source provider must be implemented, see " Define the Data Source Provider" on page 13.

To retrieve the data source information, the HP Anywhere framework provides the **DataSourceService**, which includes the following **getDataSourceConfig** method:

```
@Autowired
    DataSourceService dataSourceService;

. . .
//retrieve data source configuration from the HPA server
        DSConfiguration dataSource = dataSourceService.getDataSourceConfig(e
ventsApp.getID() + "-DS", null);
```

where:

dataSourceName is the name of the data source to retrieve, usually the same as the app service ID.

instanceName is the name of the data source instance to retrieve. If null, method returns the first data source instance for the App.

> **Note: instanceName** is currently not implemented.

This method returns **DSConfiguration** which is a key-value map of data source for the current app. The keys are according to the data source provider .xml that the app provided.

The following example displays how to build the base URL in events app using the **DataSourceService**:

```
private String getBackendDatasourceBaseUrl() throws
```

```
InvalidConfigurationException {
        //retrieve data source configuration from the HPA server
        DSConfiguration dataSource = dataSourceService.getDataSourceConfig(e
ventsApp.getID() + "-DS", null);

        //build the base url to access the backend according to the value of
known data source properties
        String protocol = (String) dataSource.getPropertyValue(DataSourceCon
sts.PROTOCOL_KEY_NAME);
        String hostname = (String) dataSource.getPropertyValue(DataSourceCon
sts.HOSTNAME_KEY_NAME);
        String port = (String) dataSource.getPropertyValue(DataSourceConsts.
PORT_KEY_NAME);

        String requestUrl = protocol + "://" + hostname + ":" + port + "/";

        return requestUrl;
    }
```

Use the DSConfiguration **getPropertyValue** method to retrieve the data source properties values.

2. Implement the REST request URL to access the relevant resource in the backend

Update the request URL to access the relevant resource.

To access the backend REST Server use Spring's RestTemplate. RestTemplate is a Spring central class for client-side HTTP access. It simplifies communication with HTTP servers, and enforces RESTful principles. It handles HTTP connections, leaving application code to provide URLs (with possible template variables) and extract results.

To use RestTemplate, all you need to do is to inject RestTemplate bean to your REST Service class, for example:

```
@Service
@Path("/events")
@Produces("application/json;charset=utf-8")
public class EventsRestService {

    @Autowired
    RestTemplate restClient;
. . .
```

After you have the backend base URL, append the relevant resource path. For example:

```
@GET
    public List<EventVo> getAll() {
```

```
        List<EventVo> events = new ArrayList<EventVo>();
        String requestUrl = null;

        try {
            requestUrl = getBackendDatasourceBaseUrl() + SERVICE_SUFFIX_PATH;
        } catch (InvalidConfigurationException e) {
            logger.error("Failed to retrieve data source configuration detai
ls, with exception:" + e);
            //error occurred, returning empty events list
            return events;
        }

        HttpEntity requestEntity = createRequestEntityBySessionCookie(MediaT
ype.APPLICATION_JSON);

        ResponseEntity<EventVo[]> response = null;

        try {
            response = restClient.exchange(requestUrl, HttpMethod.GET, reque
stEntity, EventVo[].class);
        } catch (RestClientException e) {
            logger.warn("get all events failed with RestClientException", e);
        }

        if (response != null) {
            if (response.getStatusCode() == HttpStatus.OK) {
                logger.debug("received successful response status ");
            } else {
                logger.error("response status is:" + response.getStatusCode(
));
            }
            return Arrays.asList(response.getBody());
        } else {
            return events;
        }

    }
```

3. Get the session security cookie for backend authorization

In order to build the HTTP request, use the private method in the EventsRestService class.

```
private HttpEntity createRequestEntityBySessionCookie(MediaType mediaType)
```

1. Use LWSSOService to retrieve the security cookie. For example:

```
Cookie sessionCookie = null;

        try {
            sessionCookie = securityService.getSecurityCookie(userInfoSer
vice.getUserName());
            logger.debug("Success to get cookie from session , cookie nam
e : " + sessionCookie.getName() + " , Cookie value: " + sessionCookie.get
Value());
        } catch (Exception e) {
            logger.error("failed to retrieve session cookie", e);
        }
```

2. Use **UserInfoService getUserName** to retrieve the user name of the user currently logged in.
3. Add relevant headers, cookie and body to pass them to the request.

   In this example the server accepts JSON media type and also requires CSRF header:

```
   HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        headers.add("X-CSRF-HPMEAP", "FROM-EVENTS-APP");
```

4. Add the security session cookie that is retrieved earlier to the HttpHeaders:

```
if (sessionCookie != null) { headers.add("Cookie", sessionCookie.getName() + "=" +
sessionCookie.getValue() + ";" + sessionCookie); } else { logger.error("session cookie is
missing, return requestEntity without cookie"); }
```

5. Create request HttpEntity with the headers and request body object:

```
HttpEntity request;
        if (requestObject == null) {
            request = new HttpEntity(headers);
        } else {
            request = new HttpEntity(requestObject, headers);
        }

        return request;
```

4. Invoke the REST call and analyze the return value

To invoke the REST call, use the REST template's exchange method.

For example, follow the exchange method on the Delete action:

```
@DELETE
    @Path("{id}")
    public void deleteEvent(@PathParam("id") String id) {
        String requestUrl = null;

        try {
            requestUrl = getBackendDatasourceBaseUrl() + SERVICE_SUFFIX_PATH
+ "/{id}";
        } catch (InvalidConfigurationException e) {
           logger.error("Failed to retrieve data source configuration detail
s, with exception:" + e);
        }

        HttpEntity requestEntity = createRequestEntityBySessionCookie(MediaT
ype.APPLICATION_JSON);


        ResponseEntity<EventVo> response = null;
        Map<String, String> uriParams = new HashMap<String, String>();
        uriParams.put("id", id);

        try {
            response = restClient.exchange(requestUrl, HttpMethod.DELETE, re
questEntity, EventVo.class, uriParams);
        } catch (RestClientException e) {
            logger.warn("delete event failed with RestClientException", e);
        }

        if (response != null) {
            if (response.getStatusCode() == HttpStatus.OK) {
                logger.debug("received successful response status ");
            } else {
                logger.error("response status is:" + response.getStatusCode(
));
            }
        }
    }
```

In the code example above, you compose the requestUrl that is the URL to the legacy backend server resource, then build the requestEntity that is HttpEntity that includes the relevant headers and the session cookie.

You then invoke the REST call to the backend by using Spring RestTemplate exchange method:

restClient.exchange(requestUrl, HttpMethod.DELETE, requestEntity, EventVo.class,uriParams)

From this point, the app can perform its business logic according to the response.

How do I create a new activity and send notifications (triggered by event)?

There are cases in which an app needs to create new activity and send notification to the activity's participants. In other cases the app needs to send notifications about an existing activity. This can be done by triggering an event with a context object using the Events REST API.

For details, see the *HP Anywhere API Reference*.

In the Events app example, to demonstrate the handle event feature, invoke the event REST call when there is a new event added by the user.

URL	http://localhost:8080/diamond/rest/api/V2/apps/{appId}/events
Method	POST
URI params	appId = events

Body	<pre>{
	"id": "unique-id",
	"contextObjects": [
	{
	"objectId": "135893211785089",
	"dataType": "event",
	"displayName": "Calendar Event",
	"metaData": "user@mycompany.com",
	"app": {
	"id": "eventsApp"
	}
	}
	],
	"message": "New activity created by triggering app event",
	"users": [
	{ "id": "john.smith@mycompany.com", "isNotify": true},
	{ "id": "frank.lee@mycompany.com", "isNotify": true}
	],
	"visibility": "PRIVATE",
	"notificationType": "new.event.notification",
	"subject": "Welcome to HPAnywhere"
	}</pre>

In this example, the app sends an event in order to create new activity.

When you invoke the Events REST API:

1. A new activity is created with the subject: "Event Activity Subject"
2. A system post is added to the activity timeline: "New activity created by triggering app event"

3. The activity includes the following participants: john.smith@mycompany.com, frank.lee@mycompany.com
4. The visibility of the activity is set as PRIVATE, which means only a participant in the activity can see it and search for it.
5. A context object of type event and id 135893211785089 is added to the activity.

Send notification

In addition to the activity creation, the app can send a notification. push or email messages to several channels as defined in the app admin settings. This is part of the Events REST API.

To send a notification you must:

- " Invoke the Events REST API"
- "Define notification type in the Admin Setting xml"

Invoke the Events REST API

The following code sample invokes the Events REST API:

```
private void invokeEvent(EventVo vo) {
. . .
        JSONObject notificationEventJson = buildNotificationEvent(UUID.rando
mUUID().toString(), currentUserId, vo, EVENT_NOTIFICATION_MESSAGE, EVENT_NOT
IFICATION_TYPE, DEFAULT_VISIBILITY);

        sendNotificationEvent(notificationEventJson);
    }
```

To invoke a notification event, you must:

1. Compose the JSON body for the event with the relevant values.
2. Send the event:
   a. Compose the HP Anywhere server URL with the event REST API to invoke.
   b. Invoke the Events REST API using the REST client (similar flow to invoking the backend REST API).

   In the following example,to compose the JSON body for the event, use the EventRestService:buildNotificationEvent method:

```
private JSONObject buildNotificationEvent(String eventId, String currentU
serId, EventVo vo, String message, String notificationType, String
```

```
visibility) throws JSONException
```

Finally, here is how we actually invoke REST API using the REST client (similar flow to invoking backend REST API):

```
private void sendNotificationEvent(JSONObject notificationEventJson) {

        String requestUrl = null;
        try {
            requestUrl = getHPAServerEventRestUrl();
        } catch (MissingSettingException e) {
            logger.error("Failed to send notification, due to failure whe
n retrieving the HPA server URL" + e);
            return;
        }
        HttpEntity requestEntity = createRequestEntityBySessionCookie(Med
iaType.APPLICATION_JSON, notificationEventJson.toString());

        ResponseEntity<EventVo> response = null;
        Map<String, String> uriParams = new HashMap<String, String>();
        uriParams.put("appId", eventsApp.getID());
        try {
            response = restClient.exchange(requestUrl, HttpMethod.POST, r
equestEntity, EventVo.class, uriParams);
        } catch (RestClientException e) {
            logger.warn("send notificationupdate event failed with RestCl
ientException", e);
        }

        if (response != null) {
            if (response.getStatusCode() == HttpStatus.OK || response.get
StatusCode() == HttpStatus.NO_CONTENT) {
                logger.debug("received successful response status");
           } else {
                logger.error("response status is:" + response.getStatusCo
de());
            }
        }
    }
```

Define notification type in the Admin Setting xml

The notification channels are defined for the “new.event.notification”. Enter the notification type in the <string> </string> parameter. This may be FRONTPAGE, PUSH_NOTIFICATION, EMAIL or NONE. In this example, the notification is sent to the HP Anywhere Front Page:

```
<!-- App Supported Notification Types                       -->
    <!-- supported values are: FRONTPAGE, EMAIL, PUSH_NOTIFICATION, NONE
-->

    <setting name="new.event.notification"
             sectionKey="sections.notification.types.name"
             nameKey="settings.notification.type.new.event.name"
             descKey="settings.notification.type.new.event.desc"
             refreshRate="Immediate"
             settingType="global"
             required="true"
             displayInUI="true">
        <string>FRONTPAGE</string>
    </setting>
```

How do I define and use admin settings?

There are cases in which an app requires configurations that impact the app behavior of the app. You can provide a specific Admin Settings XML file for your app that includes these configuration. Upon deployment of the app, the app settings are exposed in the admin console, and the admin user can then configure its values.

The app can then run these values in runtime, using the AdminSettingService of the HP Anywhere framework.

For more details on the admin settings see the *HP Anywhere API Reference*.

In the example the admin setting XML is defined in events-admin-settings.xml.

```
<setting name="auto.create.activity"
             sectionKey="sections.event.logic.name"
             nameKey="settings.auto.create.activity.name"
             descKey="settings.auto.create.activity.desc"
             refreshRate="Immediate"
             settingType="global"
             required="true"
             displayInUI="true">
        <boolean>true</boolean>
    </setting>
```

In the event app example, use the admin setting to decide whether the app needs to create new activity on add event. To get the admin setting value, use the AdminSettingService: getAdminSetting(String,String), for example:

```
 private boolean isAutomaticCreateActivityEnabled() throws MissingSettingExc
eption {
        return Boolean.valueOf(adminSettingsService.getAdminSetting(eventsAp
p.getID(), Consts.AUTO_CREATE_ACTIVITY_KEY));
    }
```

How do I implement All Apps (entry points)?

In order to implement All Apps entry points, you need to implement the method:

AbstractBTOServiceEE:**getSupportedEntryPointDefinitionsNoContext**

This method is called by the HP Anywhere framework, in order to show all available entry points in your app that can be opened without a given context object.

In the Events app example, two entry points without context are supported: View Events list and Add Event.

In the example below from the EventApp class, notice how we use the factories to create the list of the app-supported entry points without context:

```
@Override
    public Collection<EntryPointDefinition> getSupportedEntryPointDefinition
sNoContext() throws CustomException {

        Collection<EntryPointDefinition> output = new HashSet<EntryPointDefi
nition>();
        Locale userLocale = this.getLocale();

        App app = AppFactory.create(getID(), getLocalizedServiceName(userLoc
ale));

        EntryPoint ep = EntryPointFactory.create(Consts.EP_LIST, this.getStr
ing(userLocale, Consts.EP_LIST));
        EntryPointDefinition def = EntryPointDefinitionFactory.create(app, e
p);
        output.add(def);

        EntryPoint ep2 = EntryPointFactory.create(Consts.EP_ADD, this.getStr
ing(userLocale, Consts.EP_ADD));
        EntryPointDefinition def2 = EntryPointDefinitionFactory.create(app,
ep2);
        output.add(def2);


        return output;

    }
```

EntryPointDefinition consists of entryPoint and its owner app. Each object consists of an id and name.

The name will be displayed in the user interface, therefore it should be localized.

Note: If you do not implement this method, the default implementation returns NULL.

How do I implement Recommended Apps (entry points)?

In order to implement recommended apps feature, you need to implement the method:

AbstractBTOServiceEE: **getEntryPointDefinitionsByContext**

This method is called by the HP Anywhere framework, with a list of context objects and opened entry points that exist in current activity. According to these lists, the app should perform its own business logic and return recommended entry points.

In the Events app example, we recommend the Edit entry point if a context exists with a type that is one of the supported types.

In the example below from the EventApp class, notice how we validate the context object to apply our own recommendation logic:

```
for (BTOContext context : btoContexts) {
    if (Consts.DATA_TYPE.equals(context.getDataType())) {
        App app = AppFactory.create(getID(), getLocalizedService-
Name(userLocale));
        EntryPoint ep = EntryPointFactory.create(Consts.EP_EDIT,
this.getString(userLocale, Consts.EP_EDIT));
        EntryPointDefinition def = EntryPointDefinitionFactory.create(app,
ep);
        output.add(def);
        break;
    }
}
```

A similar logic of checking the list of the context objects can be applied to the list of open entry points where entry points are recommended according to what is currently opened.

Note: If you do not implement this method, the default implementation returns NULL.

How do I implement Recommended Participants?

To support recommended participants, the app should return a list of user ids (as defined in the user repository).

In order to implement the recommended participants feature, you need to implement the method:

AbstractBTOServiceEE: **getRelatedUsers**

This method is called by the HP Anywhere framework, with a list of context objects and open entry points that exist in the current activity. According to these lists, the app should perform its own business logic and return a list of the recommended participants.

The following example is from the EventApp class. Notice how **UserProfileFactory.createProfile** is used to compose the list of recommended users:

```
@Override
    public Collection<UserProfile> getRelatedUsers(Collection<BTOContext> bt
oContexts) throws CustomException {

        Collection<UserProfile> output = new HashSet<UserProfile>();
       // ADD USERS BY CONTEXT
        if (btoContexts != null) {


            for (BTOContext btoContext : btoContexts) {

                if (Consts.DATA_TYPE.equalsIgnoreCase(btoContext.getDataType
())) {
                    // todo: fetch the events of the event
                    UserProfile profile = UserProfileFactory.createProfile(b
toContext.getMetaData());
                    output.add(profile);
                    return output;
                }
            }
        }

        UserProfile profile = UserProfileFactory.createProfile(Consts.SUPPOR
T_EMAIL);
        output.add(profile);

        return output;
    }
```

In the EventApp example, when the app saves the context object, it also saves possible recommended users in the context object metadata field. When the app is called, it retrieves the recommended users from the context object.

This method was used in order to save time of retrieving the relevant users from the backend according to context Id.

Note that you can implement a different method for producing recommend users, by accessing the backend data source, or recommending according to the given list of open entry points.

Note: If you do not implement this method, the default implementation returns NULL.


Java™
COMPATIBLE