Introduction

The User Tagger is part of the overall Push Notification system of the UniSA Mobile Student Application. It is an executable program which is configured as a scheduled task to run at a certain point in time. The primary purpose of User Tagger is to handle the tagging functionality (addition, removal and setting of tags) in Urban Airship through their API version 3.

The User Tagger handles the tagging functionality in conjunction with the Named Users and Tag Groups features in Urban Airship’s new release. For this purpose the User Tagger uses the API end-point **POST /api/named\_users/tags** which allows the addition, removal and setting of tags on a named user. The request body may contain one or more of the JSON parameters audience, add, remove or set. Tag operations are always associated with a tag group and so the relevant tag group is specified under each operation in the request body.

A typical request may look like the following.

**POST /api/named\_users/tags HTTP/1.1**

**Authorization: Basic <master authorization string>**

**Accept: application/vnd.urbanairship+json; version=3;**

**Content-Type: application/json**

**{**

**"audience": {**

**"named\_user\_id": ["user-1", "user-2", "user-3"]**

**},**

**"add": {**

**"crm": ["tag1", "tag2", "tag3"],**

**"loyalty": ["tag1", "tag4", "tag5"]**

**},**

**"remove": {**

**"loyalty": ["tag6", "tag7"]**

**}**

**}**

Tags that are set by the User Tagger application can be used to group target audiences, thereby we only have to specify the tag name when sending a Push Notification, without having to include all individual channel id’s. Therefore this approach simplifies sending Push Notifications to a target audience.

Design

This section highlights the overall architecture of the User Tagger application. From a use-case point of view, the application is required to connect to a Tag Registry (i.e. a database table) and retrieve all tags that are present in the registry. Then it should associate the correct set of uid’s that are relevant to a particular tag and call the API end-point to post tags to Urban Airship’s store. To address the use-case above, this application is designed to take an event-driven approach.

The event-driven approach has been implemented using a Pub-Sub model as illustrated in the following.

C:\Users\galhenel\Downloads\UserTaggerDesign (1).png

As shown in the diagram above, each tag has a dedicated handler (i.e. Tag1\_Handler, Tag2\_Handler) which identifies the type of tag operation (i.e. addition or removal), generates the relevant list of uid’s (ideally by consulting the relevant repository classes) and hands over the list of uid’s along with the tag name to API Client Worker. The API Client Worker takes care of dealing with the Urban Airship API and notifies the handler whether the operation is complete or not.

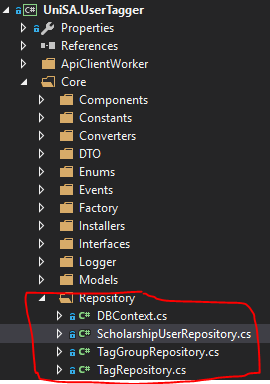
Customizations for New Tags

Whenever we have a requirement to add a new Tag, the User Tagger application requires some development work and needs to be re-deployed. There are several steps involved in order to accommodate changes into User Tagger application. As depicted in its design, User Tagger considers each new tag as a new handler and so we need to create a new handler to cater the new tag.

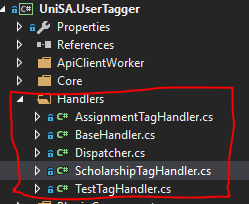
Once we have the new handler, we need to inform the User Tagger about this new handler so the next time it runs it will be able to pick up the handler. To address this issue the User Tagger embodies a plug-in architecture where it takes each new handler as a plug-in.

The following steps outline the process of creating a new handler and registering the handler as a plug-in in User Tagger.

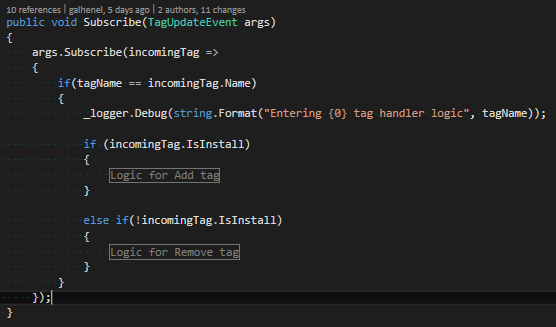
1. First, add the relevant Repository classes for the new tag. The Repository classes are sitting under the namespace UniSA.UserTagger.Core.Repository.



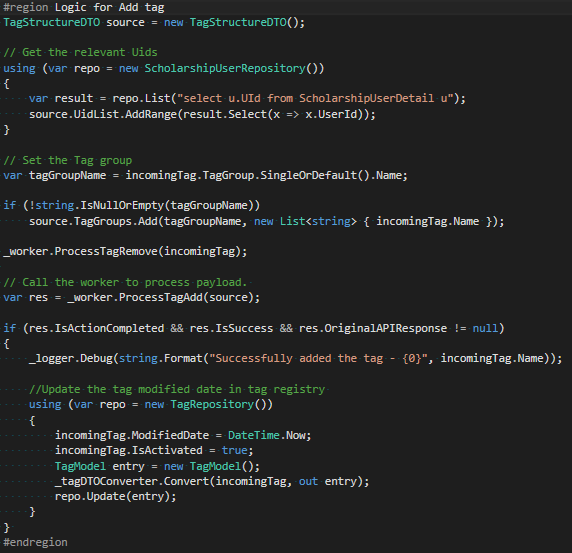
1. Create a handler for the new tag. Handlers are residing under the namespace UniSA.UserTagger.Handlers. A boiler plate handler is provided in TestTagHandler.cs to assist in creating a new one. A more concrete example for a handler can be found in ScholarshipTagHandler.cs



The handler should contain all the logic which is necessary to generate the list of uid’s for a certain tag. This has to go under the *Subscribe* method in handler. For example consider the ScholarshipTagHandler. The Subscribe method should typically have the following structure.



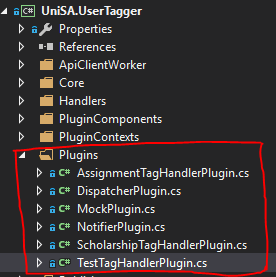
Inside the region ‘*Logic for Add Tag’*, we need to construct the relevant list of uid’s for our tag. A sample logic is provided below for the ScholarshipTagHandler.



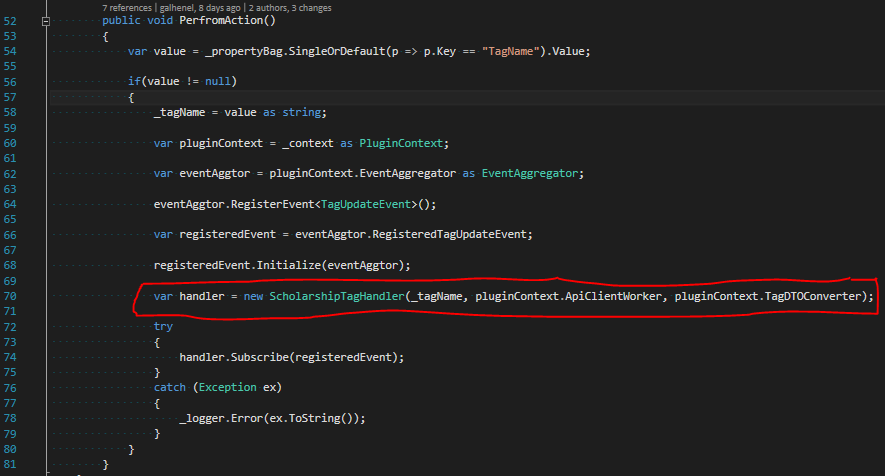
Here the handler is using the repositories we created in Step 1 and in this case ScholarshipUserRepository to create the list of uid’s. But we may have to consult several other Repositories to generate the list and this may vary based on the scenario. For example if we need to get the students who are registered to a particular course module, it might involve several repositories to generate and combine the correct list of student uid’s.

Once we have the list of uid’s we can call the API Client Worker by its method ProcessTagAdd. Calling this method will carry out the task of posting tags and its associated users to Urban Airship. And finally, based in the response from API Client Worker, update the Modified Date and IsActivated properties in the Tag registry.

1. Create a plug-in for the new handler. Once we have a handler in place, it needs to be registered in User Tagger as a plug-in. Plug-ins are residing under the namespace UniSA.UserTagger.Plugins. A Boiler plate code for the plug-in is provided in TestTagHandlerPlugin.cs to assist in creating a plug-in.

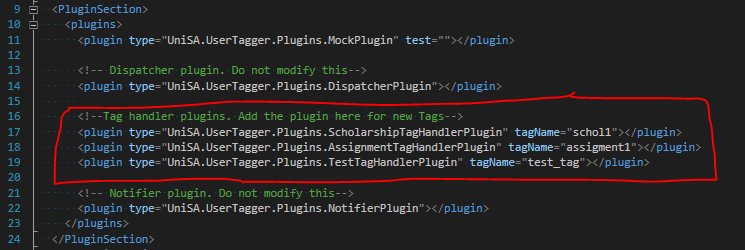


The plugin contains a method named *PerformAction* and this should call the handler we created in Step 2. Provided below is an example for the *PerformAction* method in ScholarshipTagHandlerPlugin.cs



(At line 70 it calls the handler ScholarshipTagHandler with its constructor parameters. )

1. Register the plugin we created in Step 3 inside the App.config. This needs to be done inside the <plugins> section of the App.config.



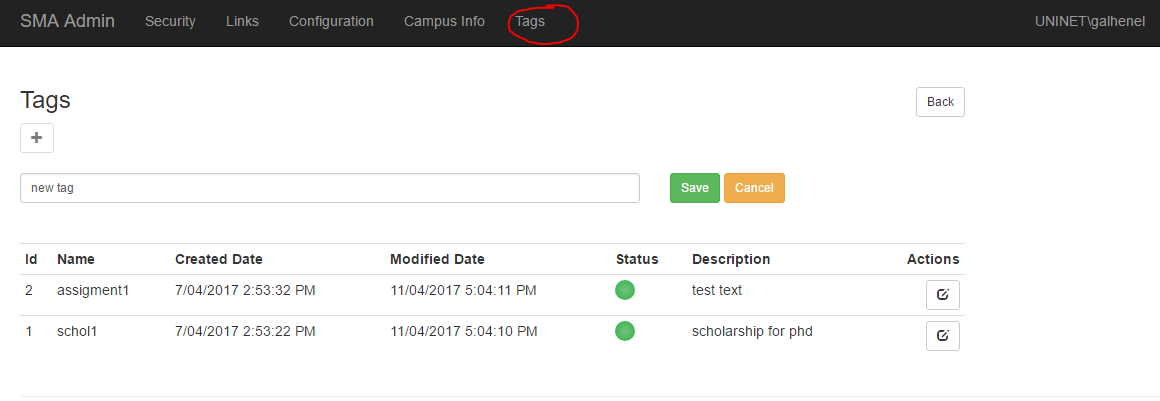
It is important that plugins are registered at the correct location. Do not modify the Dispatcher or Notifier plugins as they are core plugins inside the User Tagger application.

**All plugins that are related to tags should be registered between the Dispatcher and Notifier plugins and this is depicted in the area enclosed within red in the above figure.**

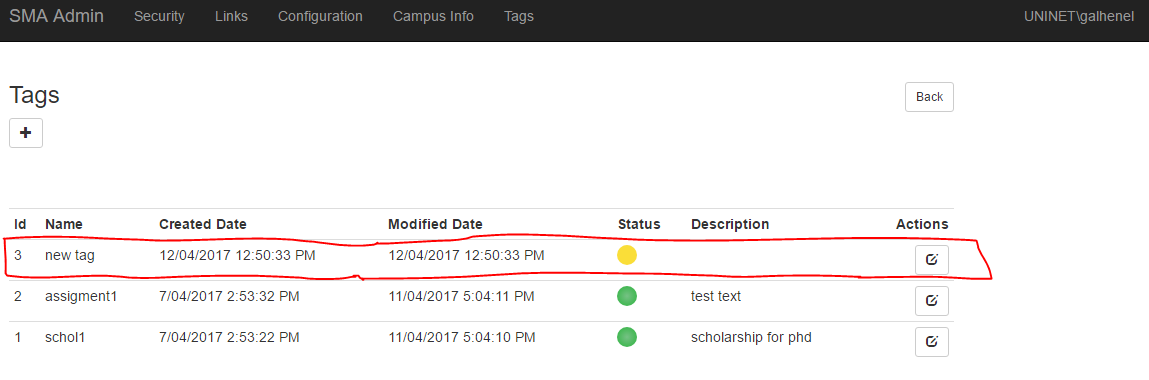
The plugin element takes two attributes namely type and tagName. The type indicates the fully qualified name of the handler plugin we created in Step 3 whereas tagName contains the actual name of the tag. Note that the value we specify here for tagName will be taken by the User Tagger to post it back to Urban Airship.

1. Finally add an entry to Tag Registry. This is done using the Student Mobile Administration web application.

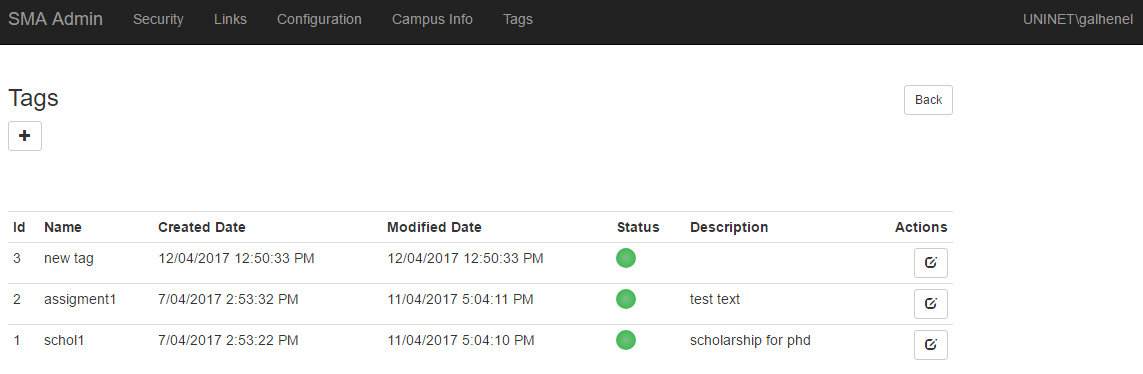
Note that the value for this entry should be the same as the value entered for tagName attribute in Step 4 above.



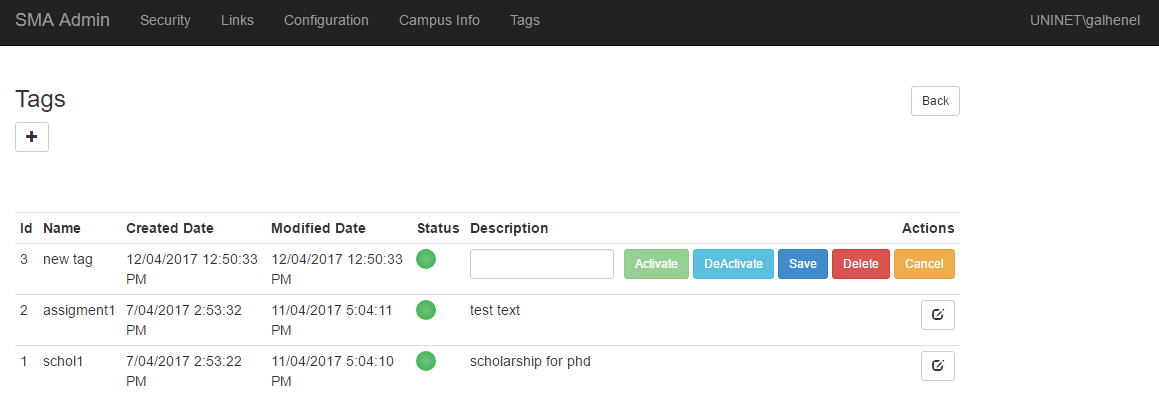
Once a new tag entry has been added, it will be shown in the tags list with Status as ‘Waiting for Activation’.



Once after the User Tagger application has completed its run, the new entry will be displayed as ‘Activated’. This conveys the idea that the tag has been added to Urban Airship and hence it is active.



Alternatively the Student Mobile Administration allows the overall management of tags including addition, deletion, updating as well as Activation and De-Activation of tags.



De-Activation allows the user to remove an already active tag from Urban Airship, and so the User Tagger removes the tag during its next run. However it does not delete the entry and allows to activate it at a later point if the user wishes so.

Deleting a tag results in removing the tag from Urban Airship as well as from the Tag registry. Note that this is a soft delete.

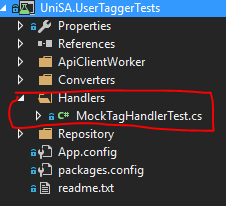
The edit mode only allows the tag description to be edited. This is because the Push Notification system works on the assumption that tag name should be unique.

1. At the end of these steps deploy the modified User Tagger Application to its target environment.

Troubleshooting

It is always recommended that after adding a handler for a new tag, that handler needs to be unit tested.

In order to troubleshoot handlers we created a Unit Test for a MockHandler and this can be found under the Unit Test Project ‘*UniSA.UserTaggerTests*’ in the Visual Studio solution.



The MockTagHandlerTest class encompasses a MockTagHandler which emulates a typical tag handler where we can assert the uid list for different scenarios.

