



## Sample Authorization Snap-ins

## Table of Contents

Introduction.....	3
Overview .....	3
Authorization CodeGrant Flow.....	4
LDAP .....	4
SAML.....	4
Client Credentials Flow.....	4
Resource Owner Password Credentials Flow.....	4
Installation and Configuration .....	5
Authorization specific Snap-in attributes .....	7
Attributes for Authorization Resources .....	7
Resource Features .....	7
Attribute for a Resource to listen on port 9443 .....	8
Enable Tokenless Access .....	8
Attributes for Authorization Clients .....	9
Authorization Service Address.....	9
Identify TaskDashboard as an Authorization Client .....	10
Use Cases .....	10
Client Credentials Flow.....	10
API Usage .....	11
Resource Owner Password Credentials Flow .....	12
API Usage .....	13
Authorization CodeGrant Flow.....	14
Filters used by the sample snap-ins for Code Grant flow .....	16
API Usage .....	16
Service Attributes used by the filters.....	17
Limitations in Authorization Code Grant Flow .....	18
Logging out users/Invalidating Access Tokens.....	18

## Introduction

Task Dashboard and Task Repository are sample snap-ins that demonstrate authorization capabilities provided by the Avaya Breeze® Authorization Service. The Task Dashboard snap-in shows the use of Authorization Service helper APIs (available via Breeze SDK) to retrieve access tokens for clients and end users. It also demonstrates how these tokens are used to retrieve tasks from the Task Repository. The Task Repository snap-in shows how a snap-in can register itself as an authorization resource, which can be consumed by authorization clients.

## Overview

The Task Repository is analogous to an OAuth 2.0 resource server, installed as a snap-in on a Breeze node. It provides a set of REST APIs to manage tasks of users and these tasks are similar to protected OAuth 2.0 resources, made accessible only if the requestor is sufficiently authorized. Tasks are guarded by the following scopes: “create, update, delete, list, listall”.

HTTP Method	API	Description	Feature Name
POST	/tasks/{user-id}	Creates a new task for a user	create (standard)
DELETE	/tasks/{user-id}/task/{task-id}	Deletes a user's task	delete (standard)
GET	/tasks/{user-id}	Gets list of all tasks for a user	list (standard)
GET	/tasks	Gets a list of all tasks of all users	listall (standard)
PUT	/tasks/{user-id}/task/{task-id}	Updates a user's task	update (standard)

This authorization capability is provided by the Breeze Authorization Service by granting access tokens to clients. The client would then use the granted token and make a request to access a protected resource (here, for example, it may ask the Task Repository to fetch all tasks of a specific user).

This leads us to explore the role of Task Dashboard. The Task Dashboard snap-in is analogous an OAuth 2.0 client whose purpose would be to make token requests to the Authorization Service and use the tokens to manage resources (tasks) by making requests to the Task Repository.

Three OAuth 2.0 grant type flows are demonstrated here:

## Authorization Code Grant Flow

The authorization code grant flow is used to obtain access tokens for Authorization Clients based on a resource owner's grant. Since this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the Authorization Service.

[\[REF\]](#)

This is the preferred flow to be used with web applications because it enables multi-factor authentication and abstracts the Client from the details of how authentication is done.

To support this flow, Task Dashboard redirects the user agent (browser) to the Authorization Service to get back an intermediary token called an "authorization code". The Authorization Service then authenticates the user. Two authentication mechanisms are supported:

### LDAP

Authorization Service presents the user with a login screen. On successful authentication, the browser is redirected back to Task Dashboard with an authorization code.

### SAML

Authorization Service redirects the user to a SAML Identity Provider. On successful authentication, the identity provider redirects the browser back to Authorization Service with a SAML assertion. Then, as with LDAP, Authorization Service redirects the browser back to Task Dashboard with an authorization code.

## Client Credentials Flow

In this flow, an Authorization Client can request an access token using only its Client Credentials (the credentials are in the form of public/private key pairs assigned to clients and managed implicitly by Breeze) when the client is requesting access to the protected resources under its control. [\[REF\]](#)

To support this flow, Task Dashboard acts as a client whose purpose is to show all tasks for all the users, but with no permissions to edit or update any of them.

## Resource Owner Password Credentials Flow

The resource owner password credentials grant type is suitable in cases where the resource owner has a trust relationship with the client. This grant type is suitable for

clients capable of obtaining the resource owner's credentials (username and password, typically using an interactive form). [\[REF\]](#)

Use of this flow is discouraged unless the Authorization Code flow is not possible, e.g. due to lack of a web user interface on the Client.

To support this flow, Task Dashboard provides a login screen to allow a user to login. Upon a successful login, it shows the list of tasks the user owns.

## Installation and Configuration

Three snap-ins need to be installed and configured on a Breeze setup:-

- Authorization Service (comes pre-loaded),
- TaskRepository and
- TaskDashboard

To build the TaskRepository and TaskDashboard snap-ins, change directories to:

[Avaya-Breeze-SDK\samples\Authorization\](#)

Compile it using “mvn clean install”. Then load and install the snap-ins on System Manager.

Once the snap-ins are installed, an admin would need to assign grants to the Task Dashboard snap-in by navigating to:

Select Avaya Breeze -> Configuration -> Authorization  
Select TaskDashboard -> Edit Grants -> New

## Authorization Configuration

This page allows you to administer authorization clients, resources and Authorization service instances.

Clients

Resource Servers

Service Instances

Authentication Mechanism

Authorization Clients

Edit Grants

Edit Key

New

Delete

3 Items

	Name	Id
<input type="radio"/>	SomeResource	9oT3uwLMQoWBZ4mIvmXCFg
<input checked="" type="radio"/>	TaskDashboard	AX3zd_7GS2SJg-ehnZOdPQ
<input type="radio"/>	TestAuthorizationClient	vtCQdBG3SMea2BstB65s3w

Select : None

### Create Grant for Authorization Client : TaskDashboard - ClusterA

This page allows you to create/edit an Authorization Grant

\*Resource Name

TaskRepository

\*Resource Cluster

ClusterA

\*Feature

create

\*Values

2 Items

Filter: Enable

Select	Value
<input type="checkbox"/>	none
<input checked="" type="checkbox"/>	standard





Select : All, None

Clicking “Commit”, a “create-standard” scope (feature value pair) gets assigned to TaskDashboard.

Similarly, scopes for “delete-standard”, “update-standard”, “list-standard”, “listall-standard” must be assigned to TaskDashboard. The end result would look something like this:

## Edit Grants for Authorization Client : TaskDashboard - ClusterA

This page allows you to administer grants for an Authorization Client

Grants				
<div> Edit Values  New  Delete</div>				
5 Items 				
	Resource Name	Resource Cluster	Feature	Values
<input type="radio"/>	TaskRepository	ClusterA	create	standard
<input type="radio"/>	TaskRepository	ClusterA	delete	standard
<input type="radio"/>	TaskRepository	ClusterA	list	standard
<input type="radio"/>	TaskRepository	ClusterA	listall	standard
<input type="radio"/>	TaskRepository	ClusterA	update	standard
Select : None				

## Authorization specific Snap-in attributes

### Attributes for Authorization Resources

#### Resource Features

The snap-in properties.xml file can be used to define features associated with the resource snap-in. This is done by using **<resource\_server>** tag containing multiple features as shown below:

```

<resource_server>
  <displayName>TaskRepository</displayName>
  <shortName>authzres</shortName>
  <feature>
    <name>install</name>
    <value>standard,none</value>
  </feature>
  <feature>
    <name>create</name>
    <value>standard,none</value>
  </feature>
  <feature>
    <name>delete</name>
    <value>standard,none</value>
  </feature>
  <feature>
    <name>list</name>
    <value>standard,none</value>
  </feature>
  <feature>
    <name>update</name>
    <value>standard,none</value>
  </feature>
</resource_server>

```

## Attribute for a Resource to listen on port 9443

All inbound requests that have a bearer token in the request header should be bypassed for whitelisting and client Cert challenge. In order to achieve this, the snap-in can listen to such requests on port 9443. For this, the resource snap-in can define a service attribute in its properties.xml file like below:

```

<attribute name="com.avaya.edp.authorization.resource.server">
  <displayName>Identifier to exclude security check</displayName>
  <helpInfo>This attribute is needed so that port
    9443 is open for requests with OAuth tokens</helpInfo>
  <validation name="anyString">
    <type>STRING</type>
  </validation>
  <admin_visible>false</admin_visible>
  <admin_changeable>false</admin_changeable>
  <factory>
    <value>authorizationresource</value>
    <user_changeable>false</user_changeable>
  </factory>
</attribute>

```

## Enable Tokenless Access

The snap-in may choose to allow requests coming through port 9443 that do not have bearer tokens. For this, the resource snap-in can define a service attribute – “com.avaya.authorization.enableTokenlessAccess”.

If this attribute value is set to “true”, this filter will allow the requests even if there is no bearer token in the Authorization header. This attribute can be defined in the properties.xml file as this:

```
<attribute name="com.avaya.authorization.enableTokenlessAccess">
  <displayName>Enable Tokenless Access</displayName>
  <helpInfo>This is used to allow requests to access resource
    end-points, even
    if there is no Authorization token. Set to false if
    all clients must have
    an Authorization Token. Value should be true or false.
  </helpInfo>
  <validation name="trueORfalse">
    <type>STRING</type>
    <pattern>true|false</pattern>
  </validation>
  <admin_visible>true</admin_visible>
  <admin_changeable>true</admin_changeable>
  <factory>
    <value>false</value>
    <user_changeable>true</user_changeable>
  </factory>
</attribute>
```

## Attributes for Authorization Clients

### Authorization Service Address

This attribute is used to set the FQDN/ IP of the node/cluster where Authorization Service is installed. This will be used by Authorization Client helper library to discover the Authorization Service node/cluster. If this value is not set, then Authorization Client helper library will use the Cluster attribute “Authorization Service Address” which can be found in the Cluster edit page.

```

<attribute name="com.avaya.authorization.service.address">
  <displayName>Authorization Service Address</displayName>
  <helpInfo>FQDN/IP of the node/cluster where Authorization Service is
    installed
  </helpInfo>
  <attr_order>1</attr_order>
  <validation name="anyString">
    <type>STRING</type>
  </validation>
  <group>
    <group_name>Client Attributes</group_name>
    <group_order>1</group_order>
  </group>
  <admin_visible>true</admin_visible>
  <admin_changeable>true</admin_changeable>
  <factory>
    <value></value>
    <user_changeable>true</user_changeable>
  </factory>
</attribute>

```

## Identify TaskDashboard as an Authorization Client

This attribute uniquely identifies the snap-in as an Authorization Client. Change the factory value to the serviceName of the snap-in.

```

<attribute name="com.avaya.authorization.client">
  <displayName>Authorization Identifier</displayName>
  <helpInfo>This is used to uniquely identify this snap-in as an
    Authorization Client
  </helpInfo>
  <validation name="anyString">
    <type>STRING</type>
  </validation>
  <admin_visible>false</admin_visible>
  <admin_changeable>false</admin_changeable>
  <factory>
    <value>TaskDashboard</value>
    <user_changeable>false</user_changeable>
  </factory>
</attribute>

```

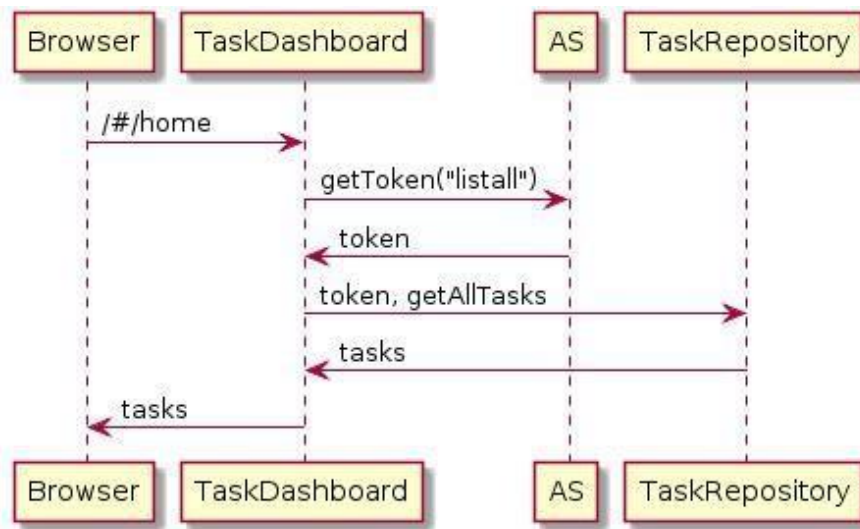
## Use Cases

### Client Credentials Flow

Access the Task Dashboard snap-in UI by pointing web browser to:

<https://<Asset IP of Breeze>/services/TaskDashboard/#/home>

A browser call to this page triggers the application to ask for a token, it asks specifically for scope: “**listall**” - The admin would have already assigned the feature (listall) in the SMGR GUI to TaskDashboard. When the token gets retrieved, TaskDashboard asks the TaskRepository for a list of all tasks owned by all users. TaskRepository validates the token and if it has permission to “listall”, it sends the data. The client renders the data, lists all the tasks grouping by username. All tasks of all users are shown, but none can be edited/removed because the scope "listall" doesn't allow that.



## API Usage

The client JavaScript (JS) makes a Web API call to `/getresource`. This uses the `getAccessToken()` Authorization Helper API, with a specific scope “listall”, meaning that Task Dashboard is requesting Authorization Service for an access token which has “listall” capabilities:

```
accessToken =  
AuthorizationClientHelper.getAccessToken(Arrays.asList("listall"));
```

It then appends the retrieved access token into the Authorization header of an HTTP GET request:

```
httpGetResource.addHeader("Authorization", AUTHORIZATION_BEARER +  
accessToken.toString());
```

Then it fires the request to the resource server (Task Repository):

```
GET <HOST>/services/TaskRepository/tasks  
HTTP/1.1
```

```
Authorization: Bearer  
1uUkSBKFIX7ATndFF5[...].0iJSUzI1NiIsImt[...].GjQip78Al4z1PrFRNi[  
...]
```

```
Content-Type: application/x-www-form-urlencoded
```

```
...
```

The results are then rendered on the GUI.

## Resource Owner Password Credentials Flow

**Note:** To test this flow, comment out the filters section in the TaskDashboard-war web.xml file, re-build and deploy the snap-in.

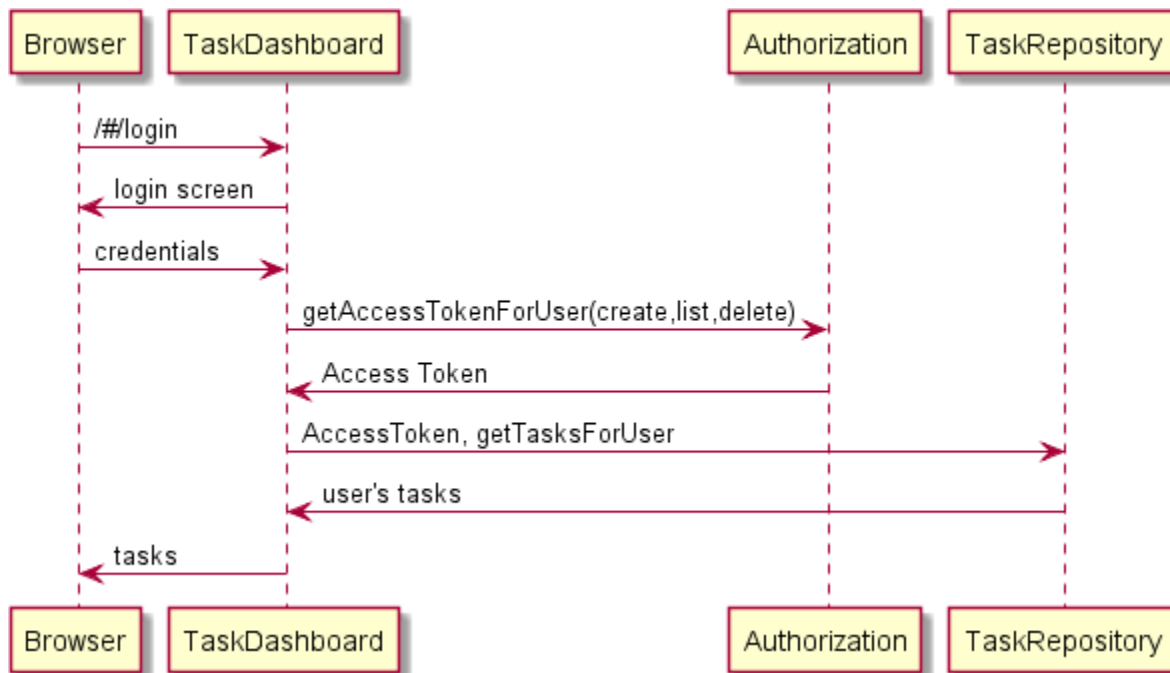
(The `AuthorizationCodeFilter` and `AccessTokenCookieFilter` are meant to be used specifically for Authorization Code Grant Flow)

Access the Task Dashboard snap in UI by pointing web browser to:

<https://<Asset IP of Breeze node>/services/TaskDashboard/#!/user>

This presents the user with a login screen. Authorization Service supports LDAP authentication for use cases following the Resource Owner Password Credentials flow.

A successful authentication of the user triggers the client application to ask for a token, the client app asks for user specific scopes: **create, delete, list**. The admin would have already mapped all these features for the client in the SMGR GUI. When the token gets retrieved, the client simply passes the token to user's browser which is accessible through Javascript (sessionStorage/cookies). The Javascript code makes a REST call to the Resource server to fetch existing tasks for the specific user. The resource server validates the token, retrieves the subject of the JWT (user@domain or user-id) and retrieves all the tasks owned by the user. The Javascript on user's browser renders the result and shows on the screen.



## API Usage

The client JavaScript (JS) makes a Web API call to `/gettokenforuser`. This uses the `getAccessTokenForUser()` Authorization Helper API, with specific scopes “create, list, delete”, meaning that Task Dashboard is requesting Authorization Service for an access token which has capabilities for creating listing and deleting tasks:

```
List<String> accessTokenScopes = Arrays.asList("create", "list", "delete");
AccessToken accessToken = AuthorizationClientHelper.getAccessTokenForUser
    (username, password, accessTokenScopes);
```

The access token is then returned to the JavaScript, where it globally sets the Authorization Header with the token as well as a session cookie in authentication.service.js:

```
if (response.data.accessToken) {
    $http.defaults.headers.common.Authorization = 'Bearer ' + response.data.accessToken;
    $cookies.putObject('dashboard_session', $rootScope.globals, {
        'expires': expiryDate,
        'path': '/services/TaskDashboard/'
    });
}
```

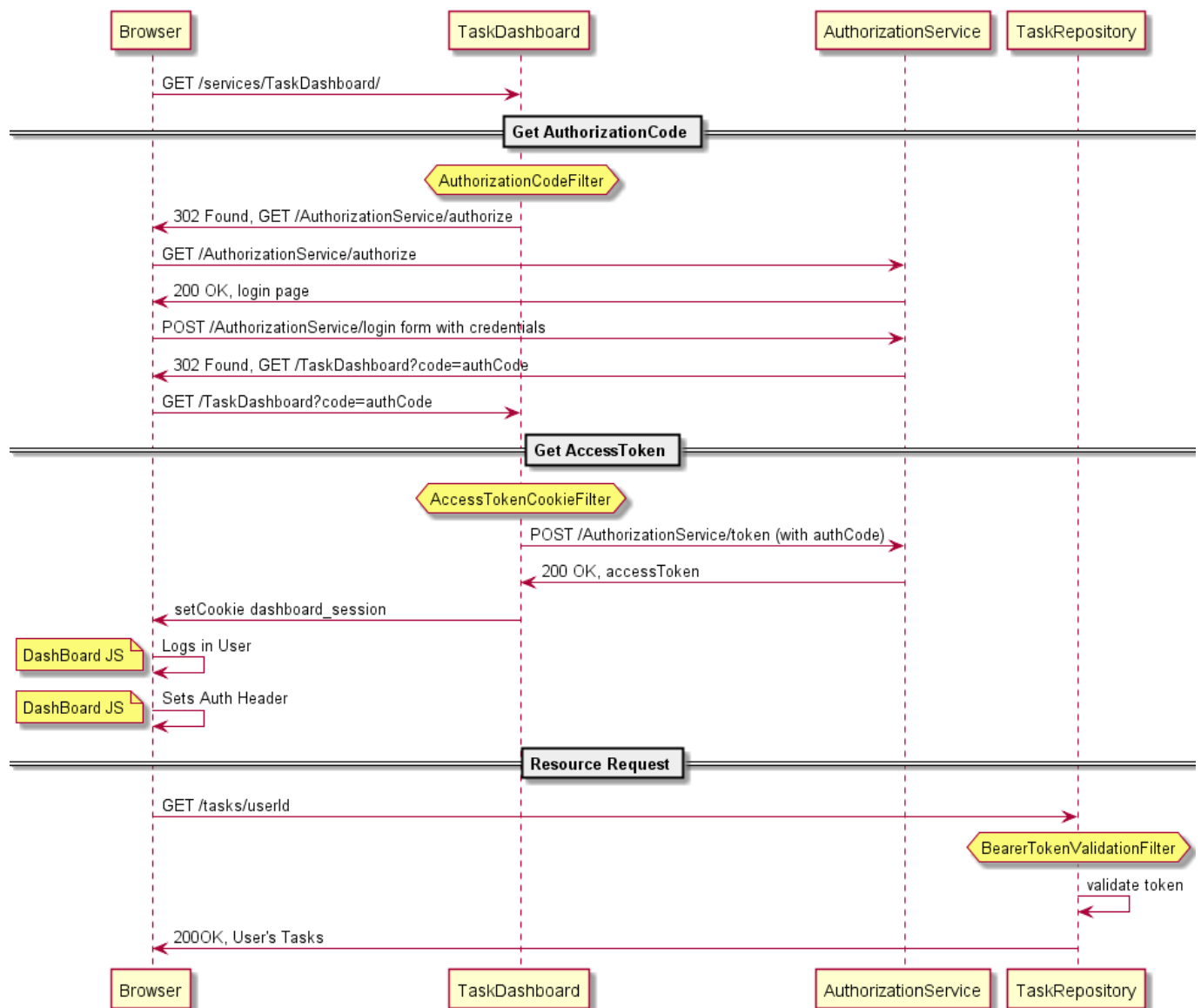
With the access token available, the user's tasks are fetched from the Repository directly:

```
function TaskService($http, $q, $rootScope){
    this.getAll = function(userId){
        return $http
            .get("/services/TaskRepository/ws/sample/tasks/" + userId)
            .then(function(response){
                return response.data;
            });
    };
};
```

The results are then rendered on the UI.

## Authorization Code Grant Flow

The Authorization Code Grant flow exempts the TaskDashboard snap-in from providing a login screen for the user to enter credentials. The user is instead redirected to the Breeze Authorization Service, which authenticates the user using LDAP or SAML. After successful authentication, the browser is redirected to the TaskDashboard with an authorization code. TaskDashboard authenticates with the Authorization Service and exchanges the browser provided authorization code with an access token. The token is then used to make requests to TaskRepository to fetch user's tasks.



To enable Authorization Code Grant Flow for TaskDashboard, go to:

SMGR > Avaya Breeze > Configuration > Attributes > ServiceClusters

Select the cluster where TaskDashboard has been installed and select Service as TaskDashboard. Change the attribute "OAuth2 Grant Type" to AuthCode and Commit changes.

Next, access the TaskDashboard snap-in UI by pointing the browser to:

<https://<Asset IP of Breeze node>/services/TaskDashboard/>

This will redirect the browser to the Authorization Service. Here, depending on the authentication mechanism chosen, a login screen will be presented:

For LDAP, the Authorization Service itself will present a login screen and will authenticate the user against an LDAP server configured on SMGR

For SAML, the Authorization Service will redirect the browser to an Identity Provider configured on SMGR. The user is authenticated by the Identity Provider and redirected back to the Authorization Service.

The Authorization Service then generates an intermediate, short-lived authorization code and redirects the browser back to TaskDashboard along with the code. TaskDashboard then can exchange the authorization code coming from the browse request with an access token granted by the Authorization Service.

### **Filters used by the sample snap-ins for Code Grant flow**

The “Filters” package includes four filters used by the two sample snap-ins for handling Authorization:

#### *AuthorizationCodeFilter*

This filter checks if the request contains an authorization code, if not, it redirects the browser to Authorization Service for getting one. It also allows the request to proceed if a session cookie is already present in the request.

#### *AccessTokenCookieFilter*

This filter fetches an access token from Authorization Service by using the authorization code present in the request. It then sets the token as a session cookie and redirects the control to a predefined path. This filter also allows the request to proceed if a session cookie is already present in therequest.

#### *AccessTokenHeaderFilter*

This filter fetches an access token from Authorization Service by using the authorization code present in the request. It then adds the token as a Bearer Authorization header to the request and allows the control to pass.

#### *BearerTokenValidationFilter*

This filter validates a bearer token. Upon successful validation, allow the request to reach the resource.

### **API Usage**

The sample filters described in the above section use the Authorization Helper APIs provided by the Breeze SDK. The specific APIs used by each of the filters is described in this section.

#### *AuthorizationCodeFilter*

This filter uses the helper API to build the redirect URI to which the browser request needs to be sent to, if it doesn't contain an authorization code.

```
String authorizationEndpoint = AuthorizationClientHelper.  
    getAuthorizationEndpoint(servletRequest);  
  
httpResponse.sendRedirect(authorizationEndpoint);
```

#### *AccessTokenCookieFilter*

The filter retrieves the authorization code present in the request, uses it to get an access token.

```
AccessToken accessToken = AuthorizationClientHelper.  
    getAccessTokenForUser(servletRequest);  
  
Cookie sessionCookie = new Cookie(..);  
httpServletResponse.addCookie(sessionCookie);  
httpServletResponse.sendRedirect(clientRedirectUri.toString());
```

#### *BearerTokenValidationFilter*

Token validation is done by using the following helper API call:

```
if (AuthorizationResourceHelper.isValid(accessToken))  
{  
    httpRequest.setAttribute(BEARER_TOKEN, accessToken);  
    filterChain.doFilter(servletRequest, servletResponse);  
}
```

### Service Attributes used by the filters

*com.avaya.authorization.sessionCookieName*

This attribute is used by the AccessTokenCookieFilter to know the name of the cookie when creating one. The TaskDashboard JavaScript would use look up the same name in the HTTP request when trying to log the user in. The factory value is implementation specific.

```
<attribute name="com.avaya.authorization.sessionCookieName">
  <displayName>Session Cookie Name</displayName>
  <helpInfo>Access token cookie filter uses this attribute to set
    set the user session's cookie name.
  </helpInfo>
  <validation name="anyString">
    <type>STRING</type>
  </validation>
  <admin_visible>true</admin_visible>
  <admin_changeable>true</admin_changeable>
  <factory>
    <value>dashboard_session</value>
    <user_changeable>true</user_changeable>
  </factory>
</attribute>
```

#### *com.avaya.authorization.clientRedirectionPath*

This attribute is used by AccessTokenCookieFilter to redirect the request to another path after setting the cookie session. The factory value is implementation specific.

```
<attribute name="com.avaya.authorization.clientRedirectionPath">
  <displayName>Client Redirection Path</displayName>
  <helpInfo>Access token cookie filter uses this path attribute to
    redirect the browser to.
  </helpInfo>
  <validation name="anyString">
    <type>STRING</type>
  </validation>
  <admin_visible>true</admin_visible>
  <admin_changeable>true</admin_changeable>
  <factory>
    <value>/services/TaskDashboard/</value>
    <user_changeable>true</user_changeable>
  </factory>
</attribute>
```

## Limitations in Authorization Code Grant Flow

### Logging out users/Invalidating Access Tokens

The Authorization Service doesn't yet support invalidating access tokens. TaskDashboard has been demonstrated to maintain a logged-in user session by use of browser cookies with a lifetime equal to the granted access token's lifetime. To log a user out, a stop gap solution would be to invalidate the "dashboard\_session" cookie in the browser and direct the user to a logout page. This however doesn't clear the session set either by the Authorization Service or the Identity Provider, so in all probabilities, if the user tries to log in again to TaskDashboard, the already authenticated session will be used and a new access token will be granted to TaskDashboard by the Authorization Service.

In order to test the flow and have the browser get a login screen again provided either by the Identity Provider or the Authorization Service, browser cookies need to be cleared manually.