|  |
| --- |
| Idies |
| SkyQuery – SciServer integration testing |
| [Type the document subtitle] |

|  |
| --- |
| László Dobos  9/3/2015 |

# SkyQuery / login-portal / Keystone

Basic working mechanism: a user is considered authenticated when the web request contains a keystone token (either in the X-Auth-Token header or in a cookie named X-Auth-Token (path: /)). On the other hand, SkyQuery also establishes a user session based on ASPX session cookies and tries to make sure that the user associated with the session is the same as the user holding the keystone token. This leads to many scenarios to be tested. The ASPX session is established automatically, even when the user holds no valid keystone token. Also, a keystone token can be invalid because it has expired or has been revoked, so token validity needs to be checked regularly.

## Glossary

**Session:** a sequence of page requests associated with the same state object on the IIS server-side. Sessions are established by creating a new session ticket by IIS. A session expires when the ticket becomes invalid. A session can also be forcefully abandoned from the server side. A page request belongs to a session if it contains a session cookie set by IIS and returned by the browser in the request.

**Session ticket:** an encrypted cookie set by IIS to certify the identity of a user. The session cookie might contain information about the user such as username or user id.

**Token:** a voucher issued by Keystone to certify the user’s identity for a limited period of time. A token can expire or can be revoked. A token is a short string that can either be passed as part of the URL, as an HTTP request header or as a cookie. In SkyQuery, the precedence order is cookie, request header, URL. If a token is found in the cookie the rest of the locations are not considered, and so on.

**Token cache:** Keystone tokens are simple string IDs and do not contain any information on the user or expiration time. Hence, tokens need to be validated against the keystone server first. This obviously cannot be done for every single web request as it would result in extremely poor performance. To address the issue, SkyQuery’s keystone client library caches tokens that already have been validated against keystone and the associated user and token expiration time have been figured out. If a request with a token in it comes in, the library first checks the token cache and only turns to keystone if the token is unknown. This logic will need to be extended if we want to support token revocation.

## Algorithm

Token validation and session management is outlined in the figure on the next page. With unit tests, every decision step needs to be tested for both outcomes. This requires at least 2*N* tests where *N* is the number of decision steps. Because Graywulf maintains its internal user database that must reflect Keystone, testing needs to be done with already registered and newly registered users as well. The default configuration is trusted mode, in which case new users found in Keystone are automatically created in Graywulf. The algorithm is implemented in   
graywulf-plugins\src\Jhu.Graywulf.Plugins\Web\Security\KeystoneAuthentication.cs



## Test scenarios

Tests require sending in HTTP headers and cookies so they’re better done from unit tests rather than interactively from the browser.

### Test access to unrestricted page

Make a request without a Keystone token and see if it fails or not.

### Test redirect to login page

Make a request to a restricted web page without a Keystone token, make sure it fails and user is redirected to the login page. Verify that ASPX session is abandoned. This can be done by checking the session cookie reset in the last response or in a follow-up response from the server.

### Redirect from login page

Make sure that users are redirected back to the correct page after successful login. Try with tricky URLs containing parameters. The keystone token must arrive back in the URL.

### Test keystone client token precedence

Submit token in cookie, header or URL and check if the correct one is used.

### Test token cache behavior: new tokens

Pass a new Keystone token to the web server. Make sure it is validated correctly. If the token is valid, SkyQuery will set a cookie with the token in it and set the expiration time of the cookie to the expiration time of the token.

### Test token validation logic

Pass an invalid/non-existing/expired/revoked token to SkyQuery. Make sure it validates it against keystone and fails with an Access Denied exception.

### Test on-demand user creation logic

If a new token is seen, it might belong to a user previously unknown to SkyQuery. In this case, SkyQuery firs validates the token against Keystone then tries to look up the user in the Graywulf registry (which mirrors the central user database to store application specific data). If the user is found then it will be associated with the ASPX session and everything is fine. If the user is missing from the Graywulf registry, it should be automatically created based on user info in Keystone.

To test the correct way of operation, create a new user in Keystone only, request a token, pass it to SkyQuery and make sure the new user is created in the registry with the correct settings.

### Test token cache behavior: known tokens

Pass a token to the web server that has been previously seen. Make sure the user is authenticated without accessing Keystone (this one’s tricky to test though, needs monitoring of network traffic). The goal here is to make sure tokens are not validated at every single request.

### Test token renewal logic

It SkyQuery detects a keystone token in a cookie and the token is already in the token cache, then it can determine whether the token is about to expire. In this case it makes token renewal request to Keystone and sends a new token in the response. To test the behavior, generate a token that’s about to expire (need to figure out how) and pass it to SkyQuery. Check if the token is renewed.

### Test switching users within the same ASPX session

SkyQuery’s Keystone client detects if the user associated with the keystone token differs from the user previously associated with the session. It then automatically abandons the session and starts a new one. To test this behavior, login with one user, make a few requests, so that the session is surely established (check session cookie too). Then switch the keystone token to another user’s. Make sure session is abandoned and SkyQuery reports the new user. Make sure the correct user is reported.

## Additional test

Graywulf takes many settings from the config files which needs to be tested before each release. These include:

* sign in URL
* sign out URL
* user registration URL
* user account details URL

When the user clicks on the sign out link in the top right corner of the page, the keystone token cookie gets deleted and the user is redirected to the logout page of the login portal. Both things should happen or the user doesn’t get logged of completely: either the cookie set by Graywulf remains, or the cookie set by the login portal does. Setting up the portal and the service under the same host name would solve this problem.

# SkyQuery – CasJobs integration

SkyQuery can access any SQL server database and use it as MyDB. A few limitations exists, namely, no user-defined function and stored procedures created in the MyDB are available from SkyQuery. To figure out the MyDB of a particular user, SkyQuery passes the keystone token to CasJobs via its REST web service interface to get the WSID of the user. With this WSID, SkyQuery looks up the connection string to the MyDB directly in the BatchAdmin database.

## Glossary

**MyDB:** a user databases, currently maganaged by CasJobs that serves as a dropbox for user tables, query results, etc. It is available from both SkyQuery and CasJobs, even though they do not share the same set of source catalogs.

**CasJobs:** you know.

**Source catalog or SkyNode:** a SkyQuery database that holds an astronomical catalog, usually from a single observatory or instrument.

## Known issues

There is a bug in CasJobs that prevents returning a user from the web service if the user does not have a MyDB yet. The reason is that a non-existing MyDB is marked with a NULL value in BatchAdmin but the table deserializer modul of CasJobs rejects all rows with NULL values in them. There is currently a work-around implemented in SkyQuery to attempt to create a user when user details are not returned by CasJobs. The work-around tries to create a user and if an error occurs it gets ignored. Then a dummy query is executed to make sure the database is created. Then the web service is queried again for user details.

## Algorithm

When SkyQuery needs to access the user’s MyDB it turns to CasJobs to get a connection string to the database. Once the connection string is figured out it gets cached for further use. MyDB resolution is implemented in a lazy-loading way, so it gets resolved at the first attempt to access it at not when the user signs in to SkyQuery. The following figure illustrates the MyDB lookup algorithm.



CasJobs creates the MyDB the first time a user executes a query. So it is not enough to login to CasJobs to automatically generate a MyDB but a query needs to be submitted. If the query gets scheduled instead of executed synchronously, MyDB is not necessarily created, so any summy queries that force MyDB creation must be very simple (SELECT 1) and submitted to the quick queus.

## Test scenarios

To test the CasJobs integration module, the following scenarios need to be tested

* newly registered user comes to SkyQuery directly, without ever logging in to CasJobs
* existing user comes to SkyQuery after using CasJobs but not running any queries there
* existing user comes to SkyQuery after running some queries from CasJobs

To test if MyDB integration works the following has to be made sure

* the main MyDB information page loads correctly
* MyDB schema shows up correctly in the schema browser
* Any query can be executed and results show up in a new MyDB table

## Known issues

MyDB might not be loaded correctly when accessed for the first time via SkyQuery web services.

# SkyQuery – SciDrive integration

SkyQuery can read files from HTTP and save data by posting files to HTTP. Because many data formats are supported, integration tests should focus on the operation of the feature and not on data formats. SciDrive integration is a simple extension to the more generic HTTP export feature.

## Algorithm

If a known host name is detected in the export URL the keystone token associated with the current user is saved with the table export or import job and passed along with the request to the target server using POST or GET.

## Test scenarios

To test SciDrive integration it enough to import a file from SciDrive and export a table to it, in any simple data format.

## Known issues

SciDrive export does not work if the keystone token expires before the export job gets scheduled for execution. This issue will be solved by implementing long-living trusts instead of tokens.