

How To Access Data With Java

The following Java code example demonstrates how to configure a connection to download data from an Earthdata Login enabled server. Note that you will need a secure way to configure the Earthdata Login username and password.

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.CookieHandler;
import java.net.CookieManager;
import java.net.CookiePolicy;
import java.net.URL;
import java.net.HttpURLConnection;
import java.util.Base64;

public class test
{
    /*
     * Prefix used to identify redirects to URS for the purpose of
     * adding authentication headers. For test, this should be:
     * https://uat.urs.earthdata.nasa.gov for test
     */
    static String URS = "https://urs.earthdata.nasa.gov";

    /*
     * Simple test.
     */
    public static void main( String[] args )
    {
        String resource = "<url of resource>";
        String username = "<URS user ID>";
        String password = "<URS user password>";
        System.out.println("Accessing resource " + resource);
        try
        {
            /*
             * Set up a cookie handler to maintain session cookies. A custom
             * CookiePolicy could be used to limit cookies to just the resource
             * server and URS.
             */
            CookieHandler.setDefault(
                new CookieManager(null, CookiePolicy.ACCEPT_ALL));

            /* Retrieve a stream for the resource */
            InputStream in = getResource(resource, username, password);
            /* Dump the resource out (not a good idea for binary data) */
            BufferedReader bin = new BufferedReader(
                new InputStreamReader(in));
            String line;
            while( (line = bin.readLine()) != null)
            {
                System.out.println(line);
            }
            bin.close();
        }
        catch( Exception t)
        {
            System.out.println("ERROR: Failed to retrieve resource");
            System.out.println(t.getMessage());
            t.printStackTrace();
        }
    }

    /*
     * Returns an input stream for a designated resource on a URS
     * authenticated remote server.
     */
}
```

```

*
*/
public static InputStream getResource(
    String resource, String username, String password) throws Exception
{
    int redirects = 0;
    /* Place an upper limit on the number of redirects we will follow */
    while( redirects < 10 )
    {
        ++redirects;
        /*
         * Configure a connection to the resource server and submit the
         * request for our resource.
         */
        URL url = new URL(resource);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setInstanceFollowRedirects(false);
        connection.setUseCaches(false);
        connection.setDoInput(true);

        /*
         * If this is the URS server, add in the authentication header.
         */
        if( resource.startsWith(URS) )
        {
            connection.setDoOutput(true);
            connection.setRequestProperty(
                "Authorization",
                "Basic " + Base64.getEncoder().encodeToString(
                    (username + ":" + password).getBytes());
        }

        /*
         * Execute the request and get the response status code.
         * A return status of 200 means is good - it means that we
         * have got our resource. We can return the input stream
         * so it can be read (may want to return additional header
         * information, such as the mime type or size ).
         */
        int status = connection.getResponseCode();
        if( status == 200 )
        {
            return connection.getInputStream();
        }

        /*
         * Any value other than 302 (a redirect) will need some custom
         * handling. A 401 from URS means that the credentials
         * are invalid, while a 403 means that the user has not authorized
         * the application.
         */
        if( status != 302 )
        {
            throw new Exception(
                "Invalid response from server - status " + status);
        }

        /*
         * Get the redirection location and continue. This should really
         * have a null check, just in case.
         */
        resource = connection.getHeaderField("Location");
    }

    /*
     * If we get here, we exceeded our redirect limit. This is most likely
     * a configuration problem somewhere in the remote server.
     */
    throw new Exception("Redirection limit exceeded");
}

```

