



19.5

Single Sign-On (SSO) Solution

Java, .NET, and PHP

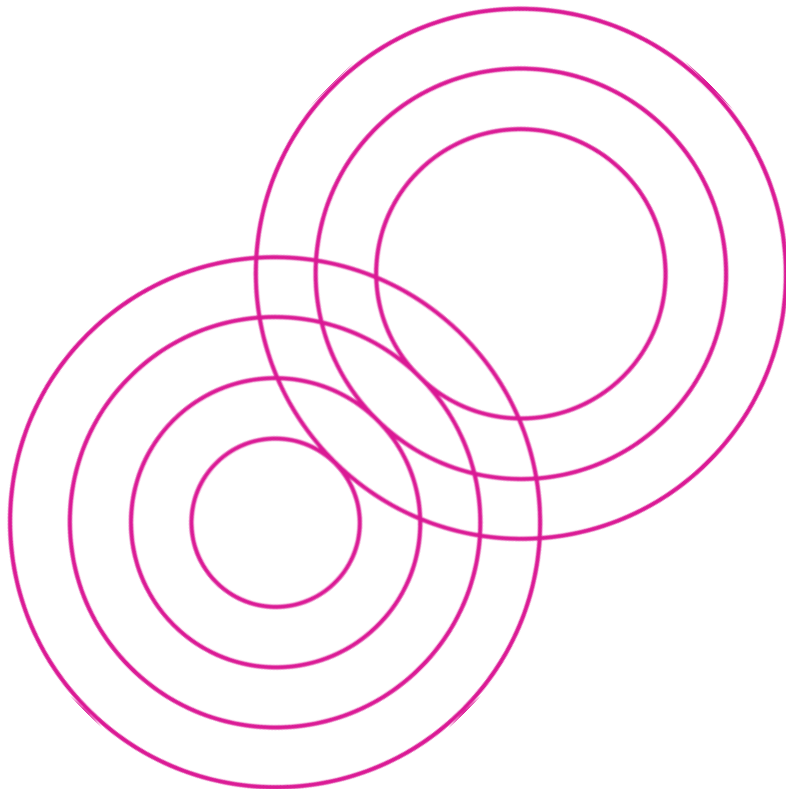




Table of Contents

- LITHIUMSSO OVERVIEW..... 4**

- LITHIUMSSO AUTHENTICATION PROCESS 4**

- CLIENT INTEGRATION 6**

- ENCRYPTION LIBRARY REQUIREMENTS..... 6**
- JAVA REQUIREMENTS..... 6
- .NET REQUIREMENTS 7
- PHP REQUIREMENTS..... 7
- URLS REQUIRED 7**
- CONFIGURING DOMAINS FOR TESTING 7**
- UNDERSTANDING SESSION IP LOCKING 8**

- DEPLOYING THE LITHIUMSSO CLIENT..... 8**

- DEPLOYMENT SAMPLE CODE 12**

- SAMPLE 1: LITHIUMSSO CREATES A COOKIE AND WRITES IT TO THE CLIENT BROWSER..... 12**
- JAVA CODE..... 12
- .NET CODE..... 12
- PHP CODE 13
- SAMPLE 2: PASS THE REQUIRED PARAMETERS AND WRITE THE COOKIE 14**
- JAVA CODE..... 14
- NET CODE..... 14
- PHP CODE 14
- SAMPLE 3: PASS OPTIONAL PARAMETERS USING A HASHMAP (JAVA, .NET) OR AN ARRAY (PHP)..... 15**
- JAVA CODE..... 15
- NET CODE..... 15
- PHP CODE 15
- SAMPLE 4: SET THE COOKIE AS NON-SECURE TO SET UP SSO IF REGISTRATION TO YOUR SITE IS UNDER HTTPS (AN SSL ENVIRONMENT) 16**
- JAVA CODE..... 16
- .NET CODE..... 16
- PHP CODE 16



SAMPLE 5: LITHIUSSO CREATES THE COOKIE, AND THE CLIENT MANUALLY SETS THE COOKIE 16

JAVA CODE 16

.NET CODE..... 17

PHP CODE 17

SAME AS SAMPLE 1 – PHP CODE 17

SAMPLE 6: INITIALIZE LITHIUSSOCOKIE: HEX STRING INSTEAD OF A KEY FILE..... 17

JAVA CODE 17

.NET CODE..... 17

PHP CODE 17

SAMPLE 7: INCLUDE THE “REFERER” PARAMETER IN URL 18

JAVA CODE 18

.NET CODE..... 18

PHP CODE 18

SAMPLE 8: SET UP SIGN-OUT WHEN THE USER COMPLETES A SESSION 19

JAVA CODE 19

.NET CODE..... 19

PHP CODE 19

ABOUT LITHIUM FALLBACK COMMUNICATION 19

CONFIGURING SSO 20

ABOUT BOUNCE SSO 21

ABOUT SINGLE LOG OUT..... 21

JAVADOC AND .NET API REFERENCE..... 22

CLASS LITHIUSSOCLIENT OVERVIEW 22

JAVA REFERENCE..... 25

FIELDS..... 25

CONSTRUCTORS 25

METHODS 27

.NET REFERENCE..... 34

FIELDS..... 34

CONSTRUCTORS 34

METHODS 36



PHP REFERENCE..... 42

CLASS SUMMARY 42

CONSTANTS 43

VARIABLES 43

FUNCTIONS 44

LITHIUMSSO FAQ..... 48



LithiumSSO overview

The Lithium Single Sign-on Solution (LithiumSSO) enables any client user system to integrate its sign-in and registration system with Khoros Community. LithiumSSO creates a seamless sign-in for the end user and enables you to:

- Create a new user account on Community
- Sign a user into Community
- Change a user's personal profile parameters
- Change a user's Community permission levels by assigning or removing a role

Users sign in as usual through the main client site. After signing in, they are forwarded to your Community site and are automatically signed in or registered on Community.

LithiumSSO works with the client system to authenticate users before signing them into Community. When an SSO solution is in place, users who sign into the client system can move freely between the client system and your community without having to sign in again.

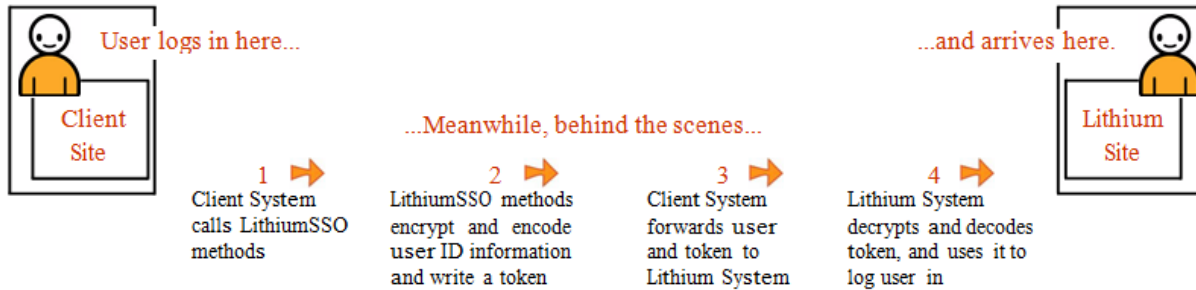
LithiumSSO uses encrypted/encoded HTTP cookies to communicate between user systems. The cookie is encrypted with strong 256-bit Advanced Encryption Standard (AES) cryptography. Secondary "fallback" methods are available in case the user's browser fails to support persistent cookies. Although the user experience is seamless, no direct connection is ever established between the client system and user systems.

- The .Net SSO Client was developed and tested on .Net version 1 and updated for .Net version 2. Although not tested on .Net Versions 3 or 3.5, it should work on these versions, too.
- The Java SSO Client is supported up to JDK 1.8+, but versions can be provided as far back as 1.5.
- The PHP SSO Client has been internally tested up to version 7.2, but should work on the most current version.

Note: For security reasons, Khoros recommends that you do not modify our SSO client software to enable unsupported features or versions. If you need to modify our SSO client, we recommend opening a Support ticket to investigate the possibility.

LithiumSSO authentication process

This section describes what happens behind the scenes with new or existing users signing in to your system and then automatically into Community.




1. When the user signs in, the client system calls the LithiumSSO library method, which writes a token based on information in the client system's user profile. (Khoros provides a secret key that encrypts the plain text user profile fields and encodes the information before writing it as a cookie to the user's browser. This unique, encrypted and encoded token identifies the user.)

Note: Make sure that the client system does not drop the LithiumSSO cookie until the user visits the community. Dropping the SSO cookie sooner could create problems if the user doesn't visit the community until after the SSO cookie timestamp expires, since the cookie isn't processed until the user hits the community.

2. The client system receives the authentication token and writes it to the user's browser. The authentication token contains the user's unique ID, login name, and email address on the client system. The token can also contain the user's profile and role information.
3. The client's system forwards the user to Community.
4. Khoros picks up the user's authentication token and decrypts and decodes its contents using a secret key that matches the client's. It then processes the clear-text contents of the token, checking to see if the unique user ID passed exists in the system and then does one of the following:
 - *If the user exists in Community*, it checks for updates to the user's information, makes any necessary changes in the user's Community profile, and signs the user in.
 - *If the user is new to Community*, it creates a new user account based on information passed in the SSO token and sets the user's access level based on the user's permissions and roles. If no roles are provided, the user receives the default permissions when they are signed in.

Note: The user is signed in only if the token is valid. If the user is visiting anonymously on the client system and visits a page that requires sign in, the Community sends the user back to the client's registration system with a query string parameter that tells the client's registration system where to send the user in Community after the user finishes signing in.



For example, if an anonymous user tries to post a message, the user is sent to the client's registration system with a query string parameter that contains the URL of the post page. This enables the user to finish posting the message as intended. The name of the query string parameter – “referer” by default - is configurable.

After the SSO process is complete and the user is signed in, users can change their login, email, or other profile settings on the client system.

Client integration

To integrate with LithiumSSO, the client system must:

- Be able to create LithiumSSO tokens from its user system
- Have the LithiumSSO libraries installed
- Have a client-specific encryption key installed

Khoros issues the LithiumSSO libraries and a unique encryption key for each deployment.

Note: (.NET only) The LithiumSSO release assembly is signed with a strong name, which guarantees the uniqueness of a .NET assembly. (Assemblies are the building blocks of .NET applications.) With strong naming, different versions of an assembly can exist side-by-side and be loaded into the Global Assembly Cache (GAC). Strong names also ensure that newer versions of an assembly come from the same publisher. For example, LithiumSSO can come only from Khoros because we have the private key used to sign the assembly.

Encryption library requirements

Environment setup is unique to your platform. This section lists the requirements for Java, .NET, and PHP.

Java requirements

You must download and install the following Java Archive (JAR) files:

- `bcprov-jdkxxx-xxx.jar` and `bcmail-jdkxxx-xxx.jar` files - Cryptography APIs available from http://www.bouncycastle.org/latest_releases.html. Be sure to download the correct file for your version of Java.

- `servlet-api.jar` - available as part of the J2EE SDK from <http://java.sun.com/products/servlet/download.html>. **Be sure to download the correct file for your version of Java.**
- `Guava-jdkxxx-xxx.jar` – Google’s guava library: <https://code.google.com/p/guava-libraries>. **Be sure to download the correct file for your version of Java.**

If you are running JDK 1.3 or earlier, you must also download and install the following JAR files:

- `jce1_2_1.jar` - Java Cryptography Extension (JCE) from Sun
- `sunjce_provider.jar` - Sun's JCE Provider Implementation
- `local_policy.jar/US_export_policy.jar` - JAR containing policy files required by JCE.

Note: These JAR files are included in Sun’s JCE Implementation JAR, which is part of JDK 1.4+. If you already have JCE installed, no further action is required.

.NET requirements

Your .NET environment must include:

- `BouncyCastle.Crypto.dll`
- `ICSharpCode.SharpZipLib.dll`

PHP requirements

Your PHP environment must include:

- `zlib.so` - compression function extension. Download it [here](#).


URLs required

To direct users to specific registration and sign in pages, you must provide Khoros with the URLs. Additionally, to send users to a specific page when they sign out of the community, provide that URL as well.

You can set the sign-in, registration, and sign-out URLs in **Community Admin > System > SSO**.

Configuring domains for testing

Cookie-based SSO requires that both your company server and the Community server be in the same domain. This is not an issue in the normal production environment. However, during testing your Community staging server is located in the `lithium.com` domain, not in your domain. As a result, you must choose a name for the Community staging server in your own



domain and create an alias (CNAME record) on your DNS server that points that local name to the Community staging server inside the lithium.com domain. You must also provide Khoros with the name you have selected so that we can configure the appropriate access.

For example, if the Community staging server is:

`http://yourcompany.stage.lithium.com`

You might choose this as your staging name:

`http://yourcompanystage.yourcompany.com`

Then you must create a DNS alias that points from your local name:

(`http://lithiumstage.yourcompany.com`) to the Community staging server

(`http://yourcompany.stage.lithium.com`).

Note: After you have set up the DNS alias, you must use that alias for SSO to function correctly.

Understanding session IP locking

To prevent malicious third parties from hijacking a user's SSO session, Community locks the session ID to the client IP address. If the client IP address does not match the IP address set in the SSO token, Community prevents the user from signing in.

If your network environment changes the client IP address between the client machine and the SSO server, SSO might not work. For example, SSO might fail under the following circumstances:

- **The SSO server is located behind a proxy server.** In this case, the proxy server changes the client IP address before contacting the SSO server. The SSO server writes the IP address provided by the proxy server in the session cookie before redirecting the client to the community. In this case, the client connects to the community using its true IP address, but has a cookie with a different IP addresses recorded.
- **The client IP address changed mid-session.** For example, this might happen if there are network changes for the client's ISP.

If you have a network environment that might change the client IP address, contact Support before implementing SSO.

Deploying the LithiumSSO Client

To deploy the LithiumSSO client:



1. Deploy the LithiumSSO library containing the LithiumSSO encryption libraries to the application server where the user system runs.
2. Place the encryption key on the user system server.
3. Instantiate the SSO client library (examples are from the “Deployment Sample Code” Sample 1 section below):
 - Java: The call to `LithiumSSOClient.getInstance` in [Sample 1 – Java Code](#)
 - .NET: The call to `SSOClient.init` in [Sample 1 - .Net Code](#)
 - PHP: The call to `new lithium_sso` in [Sample 1 – PHP Code](#)
4. Pass the following parameters to the `LithiumSSOClient`.

Parameter	Description
uniqueID	Identifies the user in both your system and Community. Community uses this ID to identify new and returning users. Note: The uniqueID is case-sensitive. For example, Community sees <code>Wassup_Doc</code> and <code>wassup_doc</code> as two different IDs.
Login (display name)	Sets the name that appears on the messages the user posts in the community. In effect, this is the user’s community identity. The sign-in name: Must be unique in Community. Cannot exceed 15 alphanumeric characters, (including hyphens and underscores). Note: You can ask Professional Services to expand the 15-character limit. However, longer login names might not display properly in some areas. Cannot be an email address or any other personally identifiable information, such as a social security number. Note: Display names are not case sensitive. If the existing user names in your system do not conform to these rules, you must add a Screen Name selection page to the registration process that against the same naming rules. Typically, you would include acceptance of its Terms and Conditions on the same page.
email	Sets the email address Community uses to communicate with the user.

See [Sample 2](#) for sample code that passes the required parameters and writes the cookie.

5. (Optional) Pass the following information via the authentication token:

- User profile information, such as first name, last name, and location. See [Sample 3](#) for sample code that passes optional parameters using a hashmap.
- The user's role, which specifies the permissions the user has after registering and logging in to Community. This enables your system to control access to specific forums or features.

In the SSO token, you can include any profile field. The most commonly used profile fields include:

Field	Description
profile.biography	The user's profile biography
profile.birthday	The user's birthday in UTC epoch time
profile.im_id_aim	The user's AIM ID
profile.im_id_icq	The user's ICQ ID
profile.im_id_msn	The user's MSN ID
profile.language	The default community site language to use for this user.
profile.location	The user's location
profile.name_first	The user's first name
profile.name_last	The user's last name
profile.remember_password	Indicates whether the community should set an auto-login cookie to remember users when they return to the community after the session has expired
profile.signature	The user's message signature
profile.url_homepage	The user's homepage URL
profile.url_icon	The URL for the user's avatar
profile.url_icon_h	Height in pixels for the user's avatar
profile.url_icon_w	Width in pixels for the user's avatar
roles.grant roles.remove	<p>Comma-delimited list of the roles to grant or remove for a user</p> <p>Note: To prevent confusion and having roles reset each time the user authenticates via SSO, we recommend you manage each role using <i>either</i> SSO or Admin, <i>but not both</i>.</p> <p>For example, you might choose to manage roles with smaller groups of users (Administrator, Moderator, and Superuser) via Community Admin and manage</p>



	<p>roles for larger populations (Employee, Partner, and Customer) via SSO, which is much more manageable.</p> <p>So, if you're passing an Employee role via SSO, don't remove it via the Admin; instead, stop passing it in roles.grant and start passing it roles.remove. If you also remove it via the Admin, and the IDP is still passing the role in roles.grant, they'll be granted the role each time they sign in.</p>
--	--

Based on the required and optional information, the LithiumSSOClient can:

- Return the value of the encryption string to be used in the cookie or as a parameter in the fallback HTTP GET or POST request. See [About Lithium fallback communication](#) for more information.
- Return a cookie "object" that you can use to write the cookie to the browser/user-agent.
- Write the cookie directly to the browser/user-agent if the request includes the `HttpServletResponse` object. (Java only)

After the SSO token is created and written, your system can redirect users to any page on Community. This redirection need not be forced; users can make their way to Community on their own at any time after logging in to your system. We recommend redirecting users to the URL provided as a query string parameter when the user is sent to the client system.

For example, if an anonymous user tries to post a message, the user is sent to the client's registration system with a query string parameter that contains the URL of the post page. This enables the user to finish posting the message. The name of the query string parameter—"referrer" by default—is configurable. See [Sample 7](#) for sample code that shows how to get the "referrer" parameter in the URL.

To sign a user out using an SSO token, write the SSO token using a unique login ID that represents an anonymous user, as shown in [Sample 8](#).

Deployment sample code

The following sections provide sample deployment code for Java, .NET, and PHP.

Note: When choosing which fields to include in the SSO token, be mindful of how long the SSO cookie will get. The standard maximum allowed size for all cookies for a domain is 4093 bytes. (This is the limit set by most modern browsers.) Some fields, such as the referrer URL shown in [Sample 7](#), can be quite long. As such, you might need to make some adjustments to accommodate for all the information you want to include in the SSO cookie.

Sample 1: LithiumSSO creates a cookie and writes it to the client browser

Java code

```
String keyPath = "C:\\usr\\local\\www\\web-  
inf\\lithium\\companyx.key";  
  
LithiumSSOClient ssoClient = LithiumSSOClient.getInstance(keyPath,  
"companyx", ".companyx.com", "serverid");  
  
// get Lithium cookie value from client values:  
// uniqueId, login, email, settingString, request, response  
  
String settingString =  
"profile.name_first=Lia|profile.name_last=Thium";  
  
ssoClient.writeLithiumCookie(uniqueId, login, email, settingString,  
request, response);
```

.NET code

```
// init SSOClient class (to be called during server init, 1 time only)  
String keyPath = "C:\\usr\\local\\www\\web-  
inf\\lithium\\companyx.key";  
SSOClient.init(keyPath, "companyx", ".companyx.com");  
  
// get Lithium cookie value from client values:  
// uniqueId, login, email, settingString, request, response  
String settingString =  
"profile.name_first=Lia|profile.name_last=Thium";  
SSOClient.writeLithiumCookie(uniqueId, login, email, settingString,  
request, response);
```

PHP code

```
// Hexidecimal string provided by Khoros (encryption key)
$key = "9D1DD509A2E1529E73DC3026D455D391";

// Initialize the Khoros php library
require_once("lithium_sso.php");
$lithiumSSOClient = new lithium_sso("companyx", ".companyx.com",
    $key);

// Setup parameters
$uniqueId = "231412341"; // unique identifier for this user
$login = "testuser234"; // login name for this user
$email = "testuser234@companyx.com"; // email address for this user

// (Optional) Additional user profile settings to pass to Lithium
$settings = array();

// Example: Set the user's homepage URL
$settings["profile.url_homepage"] = "http://www.customerhomepage.com";

// Example: Grant the user the Administrator role
$settings["roles.grant"] = "Moderator";

// Generate the actual token
$ssoValue = $lithiumSSOClient->get_auth_token($uniqueId, $login,
    $email, $settings);
```

PHP5:

```
setrawcookie($lithiumSSOClient->lithium_cookie_name . clientid,
    $ssoValue, 0, '/', clientdomain);
```

PHP4:

```
header("Set-Cookie: ".$lithiumSSOClient-
    >lithium_cookie_name.clientid,"=".$ssoValue.";Domain=".clientdomain.";
    Path=/");
```

Sample 2: Pass the required parameters and write the cookie

Java code

```
// unique id
String uniqueId = "167865";

// display name
String login = "janmon04";

// email
String email = "jane.monet@mycompany.com";

//write cookie
lithiumSSOClient.writeLithiumCookie(uniqueId, login, email,
settingsMap, request, response);
```

NET code

```
// unique id
String uniqueId = "167865";

// display name
String login = "janmon04";

// email
String email = "jane.monet@mycompany.com";

//write cookie
SSOClient.writeLithiumCookie(uniqueId, login, email, settingsMap, request,
response);
```

PHP code

```
// unique id
$uniqueId = "167865";

// display name
$login = "janmon04";

// email
$email = "jane.monet@mycompany.com";

//write cookie
$ssoValue = $lithiumSSOClient->get_auth_token($uniqueId, $login, $email,
$settings_array);
header("Content-Type: text/html; charset=utf-8");
header("Set-Cookie: ".$lithiumSSOClient-
>lithium_cookie_name.$clientId."=".$ssoValue."; Path=/;Domain=".$domain);
```

Sample 3: Pass optional parameters using a hashmap (Java, .NET) or an array (PHP)

Java code

```
Map<String, String> settingsMap = new HashMap<String, String>();
settingsMap.put("profile.name_first", "Jane");
settingsMap.put("profile.name_last", "Monet");
settingsMap.put("profile.im_id_aim", "janem04");

//write cookie
lithiumSSOClient.writeLithiumCookie(uniqueId, login, email,
settingsMap, request, response);

//write cookie
lithiumSSOClient.writeLithiumCookie(uniqueId, login, email,
settingsMap, request, response);
```

NET code

```
Hashtable settingsMap = new Hashtable();
settingsMap.Add("profile.name_first", "Jane");
settingsMap.Add("profile.name_last", "Monet");
settingsMap.Add("profile.im_id_aim", "janem04");

//write cookie
SSOClient.writeLithiumCookie(uniqueId, login, email, settingsMap, request,
response);
```

PHP code

```
$settings_array = new array();

$settings_array["profile.name_first"] = "Jane";
$settings_array["profile.name_last"] = "Monet";
$settings_array["profile.im_id_aim"] = "janem04";

//write cookie
$ssoValue = $lithiumSSOClient->get_auth_token($uniqueId, $login, $email,
$settings_array);

header("Content-Type: text/html; charset=utf-8");

header("Set-Cookie: ".$lithiumSSOClient-
>lithium_cookie_name.$clientId."=".$ssoValue."; Path=/;Domain=".$domain);
```


Sample 4: Set the cookie as non-secure to set up SSO if registration to your site is under HTTPS (an SSL environment)

Note: In the following examples, if the community URL you are sending the request to after setting the SSO cookie to is HTTPS, then set "Secure: True", but if it is not HTTPS, set it to "Secure: False".

Java code

```
Cookie ssoCookie = getLithiumCookie(uniqueId, login, email, settings,
request);
ssoCookie.setSecure(false);
response.addCookie(ssoCookie);
```

.NET code

```
HttpCookie cookie = SSOClient.getLithiumCookie(uniqueId, login, email,
request, settings);
ssoCookie.set_Secure(false);
response.AppendCookie(ssoCookie);
```

PHP code

```
//simply don't set the Secure header parameter
$ssoValue = $lithiumSSOClient->get_auth_token($uniqueId, $login, $email,
$settings_array);

header("Content-Type: text/html; charset=utf-8");
header("Set-Cookie: ".$lithiumSSOClient-
>lithium_cookie_name.$clientId."=".$ssoValue."; Path=/;Domain=".$domain);
```

Sample 5: LithiumSSO creates the cookie, and the client manually sets the cookie

Java code

```
String keyPath = "D:\\usr\\local\\www\\web-
inf\\lithium\\companyx.key";

LithiumSSOClient ssoClient =
LithiumSSOClient.getInstance(keyPath, "companyx", ".companyx.com",
"serverid");

// get Lithium cookie value from client values:
// uniqueId, login, email, settingString, request
String settingString =
"profile.name_first=Lia|profile.name_last=Thium";
```

```
Cookie ssoCookie =
ssoClient.getLithiumCookie(ssoClient.getLithiumCookieValue(uniqueId,
login, email, settingString, request));
response.addCookie(cookie);
```

.NET code

```
// init SSOClient class (to be called during server init, 1 time only)
String keyPath = "D:\\usr\\local\\www\\web-inf\\lithium\\companyx.key";
SSOClient.init(keyPath, "companyx", ".companyx.com");

// get Lithium cookie value from client values:
// uniqueId, login, email, settingString, request
String settingString = "profile.name_first=Lia|profile.name_last=Thium";
String lithiumSSOvalue = SSOClient.getLithiumCookieValue(uniqueId, login,
email, settingString, request);

HttpCookie cookie = new HttpCookie("lithiumSSO:companyx", lithiumSSOvalue);
cookie.Path = "/";
cookie.Domain = ".companyx.com";
response.AppendCookie(cookie);
```

PHP code

Same as [Sample 1 – PHP Code](#)

Sample 6: Initialize LithiumSSOCookie: Hex String instead of a key file

Java code

```
String keyString = "04A7D2234230C506047F3BAF67B5DAC2";
Key ssoKey = new Key(keyString);
LithiumSSOClient.getInstance(ssoKey.getRaw(), "companyx",
".companyx.com", "serverid")
```

.NET code

```
byte[] ssoKeyData = KeyManager.convertHexString(ssoKey);
SSOClient.init(ssoKeyData, "companyx", ".companyx.com", "serverid");
```

PHP code

Same as [Sample 1 – PHP Code](#)

Sample 7: Include the “referrer” parameter in URL

Java code

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String refererUrl = request.getParameter("referrer");
    // do something, such as logging the user in or out using the
LithiumSSOClient
    // ...
    // send the user back to the page of the forums they came from
response.sendRedirect(refererUrl);
}
```

.NET code

```
private void Page_Load(object sender, System.EventArgs e)
{
    String refererUrl = Request.QueryString["referrer"];

    // do something, such as logging the user in or out using the
LithiumSSOClient
    // ...

    // send the user back to the page of the forums they came from
response.Redirect(refererUrl);
}
```

PHP code

```
<?php
if ($_POST != null) {
    $referrerUrl = $_POST['referrer'];

    // do something, such as logging the user in or out using the
LithiumSSOClient
    // ...

    // send the user back to the page of the forums they came from
header("Location: ".$referrerUrl);
}
?>
```

Sample 8: Set up sign-out when the user completes a session

Java code

```
String uniqueId = LithiumSSOClient.ANONYMOUS_UNIQUE_ID;
ssoClient.writeLithiumCookie(uniqueId, login, email, settingString,
request, response);
```

See [Javadoc reference](#) for information on syntax.

.NET code

```
String uniqueId = SSOClient.ANONYMOUS_UNIQUE_ID;
SSOClient.writeLithiumCookie(uniqueId, login, email, request, response);
```

PHP code

```
$uniqueId = $lithiumSSOClient->ANONYMOUS_UNIQUE_ID;
$ssoValue = $lithiumSSOClient->get_auth_token($uniqueId, $login, $email,
$settings);
```

PHP5:

```
setrawcookie($lithiumSSOClient->lithium_cookie_name . clientid, $ssoValue, 0,
'/', clientdomain);
```

PHP4:

```
header("Set-Cookie: ".$lithiumSSOClient->
>lithium_cookie_name.clientid, "=".$ssoValue.";Domain=".clientdomain.";
Path=/");
```

About Lithium fallback communication

The LithiumSSO system supports a fallback method for communicating between the client system and Community. In addition to the SSO cookie, you can pass the encrypted value generated by the LithiumSSOClient class to Community through either an HTTP GET or POST request.

You make the GET or POST request to the SSO sign-in page URL. For example:

```
http://myforum.mydomain.com/mycommunity/sso
```

Include the following parameters in your GET or POST request:

Parameter Name	Sample Value
sso_value	~30asldivjha3093wjhafkefjaow3r934uefjo349aur9w03jhoas0aw349aj0f9
referer	http://forums.mydomain.com

You can change the name of the `referer` parameter as part of the configuration for your community. If you do not specify a referer URL, the user is taken to the community front page.

Configuring SSO

To configure SSO settings for your community:

1. Go to **Community Admin > System > SSO**
2. Configure the SSO settings:

Setting	Description
Turn on Lithium Single Sign-On (SSO)	Enable or disable LithiumSSO.
Bounce URL (Optional)	URL to bounce the first request of a session to. Used to determine the login state. Used in bounce SSO scenarios only. Leave blank to disable bounce.
Allow SSO user Email changes	Enable SSO users to change the email associated with their account.
Language Parameter Name	Name of the query string parameter to pass to the host system.
Return Value Parameter Name	Name of the query string parameter to pass to the host system.
Redirect Reason Parameter Name	Name of the query string parameter to pass to the host system.
URL to registration page	Direct users to this URL when they register.
URL to login page	Direct users to this URL at sign-in.
URL to logoff page	Direct users to this URL at sign-out.
Single Log Out (optional)	Enable logging users out of the community from the client's system.
Enable Auto Login for Fallback SSO	Enables auto login via fallback SSO if cookie-based login fails.

Auto generate SSO User Login	When login is enabled in SSO Complete Registration form and this field is checked, the XML Text Generator for SSO User login (which must also be checked) generates a user login and populates the login field in the SSO Complete Registration form. Users can still change the pre-populated values. If login is disabled in the SSO Complete Registration form, this field has no effect.
XML text generator for SSO user login	When login is enabled in SSO Complete Registration form and the Auto generate SSO User Login field is checked, the XML Text Generator for SSO User login generates a user login and populates the login field in the SSO Complete Registration form when this field is checked. Users can still change the pre-populated values. If login is disabled in the SSO Complete Registration form, this field has no effect.

3. Click **Save**.

About Bounce SSO

Bounce SSO ensures that a user’s community session is synced with the user’s session on the client system. When user Community sessions times out, Bounce SSO redirects them to a client-specified URL which the client can set up to redirect users back to the originating community pages with the SSO token to sign them back in.

Bounce syncs your company website with Community. When enabled, users who are already signed in on the customer side can be seamlessly signed into Community when they get to the community via a link or bookmark.

Bounce hits the configured bounce URL once per session on the user’s first visit to the community, if the user is not signed in. To support this, you need to implement a new page that checks to see if the user is signed in (no UI is required). If they are signed in, set an SSO cookie. Then, in either case, redirect back to the referrer.

About Single log out

You use Single Log Out when Single Sign-On is enabled and a user signs out on your company website. You can set an SSO cookie with the Community anonymous user ID (refer to Sample 8). This ensures that the user is not signed in when they visit the community.

Javadoc and .NET API reference

This section provides the summary and detailed code information for Java and .NET.

Class LithiumSSOClient overview

```
lithium.user.sso.LithiumSSOClient  
public class LithiumSSOClient  
extends java.lang.Object (Java only)
```

Class used to generate encrypted cookies, used for creating authentication tokens used by the LithiumSSO system. The cookies are encrypted using a private key, which is provided by Lithium. The default name of the key is `cookie.prod.key` and it resides in the current directory. You can change the file path to the key by specifying the `keyPath` during the call to `getInstance()` (Java) or `init()` (.NET).

If you are using the SMR system, your `private.key` must reside in the same location as `cookie.prod.key`.

If possible, call `getInstance()` (Java) or `init()` (.NET) during server initialization to minimize the response time for the first user. Initialization times may vary depending on key size and server CPU speeds. If the client is using the SMR functionality, you must call `initSMR(String)` prior to the initial `getInstance()` (Java) or `init()` (.NET) method to initialize the `private.key` file.

You can pass the authentication token to the client in three ways:

- Call `writeLithiumCookie()`, which creates a `Cookie` object and writes it directly to the client via the response object.
- Call `getLithiumCookie()`, which returns a `Cookie` object, and then write it to the client manually.
- Call `getLithiumCookieValue()`, which returns an encrypted string that can be used to create the lithiumSSO token.

Note: Cookie name must be `lithiumSSO:<clientID>` where `clientID` is the unique ID assigned to you by Khoros. You pass this value in the call you make to first instantiate the LithiumSSO client.

To process a user account the LithiumSSO system requires at least 3 parameters:

- **uniqueId** - A unique identifier which is used to represent the user. (String)
`uniqueId` is the same as the SSO ID and contain up to 120 single-byte characters. The only length restrictions are the size of the database columns for these fields in the users table. `uniqueId` cannot be `null` or empty string ("") or contain delimiter token (|).
- **login** - User's display name for Community. (String)
This name appears on all messages the user posts. It can contain up to 255 single-byte characters, cannot be null or empty string (""), and cannot contain delimiter token (|).
- **email** - User's email address for Community. (String)
It can contain up to 255 single-byte characters and be null, but if email is specified, then it cannot contain delimiter token (|)

Note: Many customers opt to have the community capture login and/or email when the user first authenticates to the community via SSO. If the community is configured this way, the login and/or email passed in the SSO token doesn't matter, since this information is captured by Community during the registration flow.


You can also provide an optional hash of user settings.

The unique identifier must be the same as the one used on the client system, unless the client system uses personally identifiable information, such as a social security number. This ID is usually in the form of a unique integer, but can be any arbitrary string (for example, a user name can be used if it is unique on the system).

For security purposes, three additional parameters are required:

- **reqUserAgent** - [`HttpServletRequest.getHeader ("User-Agent")`] used for security identification information
- **reqReferer** - [`HttpServletRequest.getHeader ("Referer")`] used for security identification information
- **reqRemoteAddr** - [`HttpServletRequest.getRemoteAddr ()`] used for security identification information

You can pass the optional settings (Map) object in the token with additional fields to be synchronized with the Community servers. You can use the settings hash to pass profile information about the user or to set up the user's permissions on Community. Permissions are determined with the `roles.grant` setting entry. Roles are used on Community to grant or deny permissions, and are created and managed by the client. If no Role is specified in the



settings hash, the system assigns the default role associated with a new system user. (Example:
`roles.grant = "Moderator"`)

Valid settings include:

- **roles.grant:** A comma-delimited list of Roles to grant this user
- **profile.name_first:** The user's first name
- **profile.name_last:** The user's last name
- **profile.location:** The user's geographic location

Java reference

Fields

This section contains the summary and detailed information for fields.

Field
ANONYMOUS_UNIQUE_ID <code>public static final java.lang.String ANONYMOUS_UNIQUE_ID</code>
COOKIE_NAME <code>public static final java.lang.String COOKIE_NAME</code>
COOKIE_NAME_PREFIX <code>static java.lang.String</code>
SEP_TOKEN <code>public static final char SEP_TOKEN</code>
VERSION <code>public static final java.lang.String VERSION</code>

Constructors

This section contains the summary and detailed information for constructors.

Constructor
getInstance (keyPath, clientId, clientDomain, serverId) Initializes the SSOClient with an encryption key, client id, the client's domain, and a server id. A path to the key file is passed in and used to find the key file and load the key. The form of the domain name is set by RFC 2109. Support provides the key and client id when the encryption key request is made. This method must be called first, prior to initial calls to either <code>getLithiumCookie()</code> or <code>writeLithiumCookie()</code> are made. Calling this method during server initialization is highly recommended to minimize response times for the first user. The initialization optionally takes the <code>serverId</code> parameter. When the SSO API issues cookies, it also issues a one-time-use ID to prevent cookies from being reused. This ID is specific to the instance of the SSO API. Whether or not the <code>newServerId</code> parameter is passed, a random string will be used in the creation of the server ID to ensure that each instance of the SSO Client has a unique server ID. This method can also be called at any time to reload the key, usually after a new key is generated. Parameters: keyPath the byte array that contains the encryption key.

clientId a unique identifier assigned to the client.

clientDomain the client's domain.

serverId the ID of the client's server. If Null or an empty String is passed in for `serverId`, it will default to the ip address of the client.

Returns:

an initialized `LithiumSSOClient` object.

Throws:

`SSOException` - if initial settings are invalid.

```
public static synchronized LithiumSSOClient getInstance
(java.lang.string keyPath,
java.lang.String cliendId,
java.lang.String clientDomain,
java.lang.String serverId) throws SSOException
```

getInstance (keyData, clientId, clientDomain, serverId)

Initializes the `SSOClient` with an encryption key, client id, the client's domain, and a server id. Key is passed in as a byte array. Clients can use the `Key` class to generate the byte array from a hex String. The form of the domain name is set by RFC 2109. Both the key and client id are provided by Khoros. This method must be called first, prior to initial calls to either `getLithiumCookie()` or `writeLithiumCookie()` are made. Calling this method during server initialization is highly recommended to minimize response times for the first user.

The initialization optionally takes the `serverId` parameter. When the SSO API issues cookies, it also issues a one-time-use ID to prevent cookies from being reused. This ID is specific to the instance of the SSO API. Whether or not the `newServerId` parameter is passed, a random string will be used in the creation of the server ID to ensure that each instance of the SSO Client has a unique server ID.

This method can also be called at any time to reload the key, usually after a new key is generated.

Parameters:

keyData the byte array that contains the encryption key.

clientId a unique identifier assigned to the client.

clientDomain the client's domain.

serverId the ID of the client's server. If Null or an empty String is passed in for `serverId`, it will default to the IP address of the client.

Returns:

an initialized `LithiumSSOClient` object

```
public static synchronized LithiumSSOClient getInstance
```

```
(byte[] keydata,  
java.lang.String cliendId,  
java.lang.String clientDomain,  
java.lang.String serverId) throws SSOException
```

Methods

This section contains the summary and detailed information for methods.

Method detail
getKeyPath () Returns the path to the LithiumSSO encryption key. <code>public java.lang.String getKeyPath()</code>
getServerId () Returns the Server ID. <code>public java.lang.String getServerId()</code>
getClientDomain () Returns the Client Domain used to set the Cookie domain. <code>public java.lang.String getClientDomain()</code>
getLithiumCookieName () Returns the name that is used for cookies containing an SSO token. <code>public java.lang.String getLithiumCookieName()</code>
getClientId () Returns the Client ID. <code>public java.lang.String getClientId()</code>

getLithiumCookieValue (uniqueId, login, email, settingString, reqUserAgent, reqReferer, reqRemoteAddr)

Returns an encrypted String to be used for an authentication token.

The `getInstance()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **settingString** - name/value pairs for user profile fields. (optional)
- **reqUserAgent** - [`HttpServletRequest.getHeader("User-Agent")`] used for security identification information
- **reqReferer** - [`HttpServletRequest.getHeader("Referer")`] used for security identification information
- **reqRemoteAddr** - [`HttpServletRequest.getRemoteAddr()`] used for security identification information

Throws: `SSOException` if initial settings are invalid.

```
public java.lang.String getLithiumCookieValue
(java.lang.String uniqueId,
    java.lang.String login,
    java.lang.String email,
    java.lang.String settingString,
    java.lang.String reqUserAgent,
    java.lang.String reqReferer,
    java.lang.String reqRemoteAddr)
    throws SSOException
```

See also: `getInstance(String keyPath, String clientId, String clientDomain, String serverId)`

getLithiumCookieValue (uniqueId, login, email, settingString, request)

Returns an encrypted String to be used for an authentication token.

The `getInstance()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **settingString** - name/value pairs for user profile fields. (optional)
- **request** - `HttpServletRequest` used for security identification information

Throws:

`SSOException` if initial settings are invalid.

```
public static java.lang.String getLithiumCookieValue
    (java.lang.String uniqueId,
     java.lang.String login,
     java.lang.String email,
     java.lang.String settingString,
     javax.servlet.http.HttpServletRequest request)
    throws SSOException
```

See Also: `getInstance(String keyPath, String clientId, String clientDomain, String serverId`

getLithiumCookieValue (uniqueId, login, email, settings, request)

Returns an encrypted String to be used for an authentication token.

The `getInstance ()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **settings** - name/value pairs for user profile fields. (optional)

- **request** - `HttpServletRequest` used for security identification information

Throws: `SSOException` - if initial settings are invalid.

```
public java.lang.String getLithiumCookieValue
    (java.lang.String
     uniqueId,
     java.lang.String login,
     java.lang.String email,
     java.util.Map<java.lang.String,java.lang.String> settings,
     javax.servlet.http.HttpServletRequest request)
    throws SSOException
```

See Also: `getInstance(String keyPath, String clientId, String clientDomain, String serverId)`

getLithiumCookie (lithiumSSOValue)

Helper method returns an `HttpCookie` containing an encrypted authentication token that can be passed to the client (usually via `HttpResponse`) for seamless login and registration on the Community servers. The `getInstance()` method must be called before this method is accessed.

Parameters:

lithiumSSOValue - Result of calling `getLithiumCookieValue()` [lithium SSO token]

Throws: `SSOException` if initial settings are invalid.

```
public Cookie getLithiumCookie(java.lang.String lithiumSSOValue)
    throws SSOException
```

See Also: `getInstance(String KeyPath, String ClientId, String ClientDomain)`

getLithiumCookie (uniqueId, login, email, settings, request)

Returns an `HttpCookie` containing an encrypted authentication token that can be passed to the client (usually via `HttpResponse`) for seamless login and registration on the Community servers. The `getInstance()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on the Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on the Community
- **settings** - name/value pairs for user profile fields. (optional)
- **request** - `HttpServletRequest` used for security identification information

Throws: `SSOException` if initial settings are invalid.

```
public Cookie getLithiumCookie
    (java.lang.String uniqueId,
     java.lang.String login,
     java.lang.String email,
     java.util.Map<java.lang.String,java.lang.String>
     settings,
     HttpServletRequest request)
    throws SSOException
```

See Also: `getInstance(String keyPath, String clientId, String clientDomain, String serverId)`

writeLithiumCookie (lithiumSSOValue, response)

Helper method writes an encrypted authentication token cookie directly to the client via the `HttpResponse` object for seamless login and registration on the Community servers. The `getInstance()` method must be called before this method is accessed

Parameters:

- **lithiumSSOValue** - Result of calling `getLithiumCookieValue()` [lithium SSO token]
- **response** - `HttpServletResponse` to set the cookie with

Returns: `true` if the token was written successfully.

Throws: SSOException if initial settings are invalid.

```
public boolean writeLithiumCookie (java.lang.String  
    lithiumSSOValue,  
    HttpServletResponse response)  
    throws SSOException
```

See Also: getInstance(String keyPath, String clientId, String clientDomain, String serverId)

writeLithiumCookie (uniqueId, login, email, settings, request, response)

Writes an encrypted authentication token cookie directly to the client via the HttpServletResponse object for seamless login and registration on the Community servers. The getInstance () method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **settings** - name/value pairs for user profile fields. (optional)
- **request** - HttpServletRequest used for security identification information
- **response** - HttpServletResponse to set the cookie with

Returns: true if the token was written successfully

Throws: SSOException if initial settings are invalid.

```
public boolean writeLithiumCookie  
    (java.lang.String uniqueId,  
    java.lang.String login,  
    java.lang.String email,  
    java.util.Map<java.lang.String,java.lang.String>  
    settings,  
    HttpServletRequest request,
```

```
HttpServletResponse response)
throws SSOException
```

See Also: `getInstance(String keyPath, String clientId, String clientDomain, String serverId)`

removeLithiumCookie (request, response)

Removes the LithiumSSO cookie from the user's cookie cache. Call this method when a user signs out from the client system.

Parameters:

- **request** - `HttpServletRequest` used for security identification information
- **response** - `HttpServletResponse` to remove the cookie with

```
public void removeLithiumCookie(HttpServletRequest request,
                                HttpServletResponse response)
```

initSMR (smrKeyPath)

Initializes the SMR private key.

Note: PrivacyGuard is a separate Khoros product that enables Community to send email messages to community members without actually knowing what the user's email address is. Contact your Customer Success Manager (CSM) for more information about PrivacyGuard.

Parameters:

smrKeyPath - SMR private key path.

Throws: `SSOException`

```
public void initSMR(java.lang.String smrKeyPath)
throws SSOException
```

getSMRField (param)

Returns passed parameter encrypted via the SMR private key.

Note: PrivacyGuard is a separate Khoros product that enables Community to send email messages to community members without actually knowing what the user's email

address is. Contact your Customer Success Manager (CSM) for more information about PrivacyGuard.

Parameters:

param – field to encrypt.

Throws: `SSOException` - if SMR initialization has failed.

```
public java.lang.String getSMRField(java.lang.String param)
    throws SSOException
```

.NET reference

Fields

This section contains the summary and detailed information for fields.

Field
ANONYMOUS_UNIQUE_ID Public static read-only String ANONYMOUS_UNIQUE_ID
COOKIE_NAME public static read-only System.String COOKIE_NAME
SEP_TOKEN public static read-only char SEP_TOKEN
VERSION public static read-only System.String VERSION

Constructors

This section contains the summary and detailed information for constructors.

Constructor
init (newKeyPath, newClientId, newClientDomain, newServerId) Initializes the SSOClient with an encryption key, client id, the client's domain, and a server id. A path to the key file is passed in and used to find the key file and load the key. The form of the domain name is set by RFC 2109. Both the key and client id are provided by Khoros. This method must be called first, prior to initial calls to either <code>getLithiumCookie()</code> or <code>writeLithiumCookie()</code> are made. Calling this method during server initialization is highly recommended to minimize response times for the first user.

The initialization optionally takes the `newServerId` parameter. When the SSO API issues cookies, it also issues a one-time-use ID to prevent cookies from being reused. This server ID is used in the creation of those one-time-use IDs. Whether or not the `newServerId` parameter is passed, a cryptographically-random string will be used in the creation of the server ID to ensure that the SSO Client has a unique server ID. This method can also be called at any time to reload the key. **Only call this method once, after a new key is generated.**

Parameters:

`newKeyPath` the path to the file that contains the encryption key.

`newClientId` a unique identifier assigned to the client.

`newClientDomain` the client's domain.

`newServerId` the ID of the client's server. If Null or an empty String is passed in for `serverId`, it will default to the ip address of the client.

```
public static void init(  
    System.String newKeyPath,  
    System.String newCliendId,  
    System.String newClientDomain,  
    System.String newServerId) throws SSOException
```

Init (newKeyPath, newClientId, newClientDomain)

Initializes the SSOClient with an encryption key, client id, and the client's domain. A path to the key file is passed in and used to find the key file and load the key. The form of the domain name is set by RFC 2109. Both the key and client id are provided by Khoros. This method must be called first, prior to initial calls to either `getLithiumCookie()` or `writeLithiumCookie()` are made. Calling this method during server initialization is highly recommended to minimize response times for the first user.

When the SSO API issues cookies, it also issues a one-time-use ID to prevent cookies from being reused. A server ID is used in the creation of those one-time-use IDs.

When this method is called, a cryptographically-random string will be generated to create a unique server ID.

This method can also be called at any time to reload the key, **Only call this method once, after a new key is generated.**

Parameters:

`newKeyPath` the path to the file that contains the encryption key.

`newClientId` a unique identifier assigned to the client.

`newClientDomain` the client's domain.

```
public static void init(  
    System.String newKeyPath,  
    System.String newCliendId,  
    System.String newClientDomain) throws SSOException
```

Init (newKeyData, newClientId, newClientDomain, newServerId)

Initializes the SSOClient with an encryption key, client id, the client's domain, and a server id. A path to the key file is passed in and used to find the key file and load the key. The form of the domain name is set by RFC 2109. Both the key and client id are provided by Khoros. This method must be called first, prior to initial calls to either `getLithiumCookie()` or `writeLithiumCookie()` are made. Calling this method during server initialization is highly recommended to minimize response times for the first user.

The initialization optionally takes the `newServerId` parameter. When the SSO API issues cookies, it also issues a one-time-use ID to prevent cookies from being reused. This server ID is used in the creation of those one-time-use IDs. Whether or not the `newServerId` parameter is passed, a cryptographically-random string will be used in the creation of the server ID to ensure that the SSO Client has a unique server ID. This method can also be called at any time to reload the key. **Only call this method once, after a new key is generated.**

Parameters:

`newKeyData` the byte array that contains the encryption key.

`newClientId` a unique identifier assigned to the client.

`newClientDomain` the client's domain.

`newServerId` the ID of the client's server. If Null or an empty String is passed in for `serverId`, it will default to the IP address of the client.

```
public static void init(  
    byte[] newKeyData,  
    System.String newCliendId,  
    System.String newClientDomain,  
    System.String newServerId) throws SSOException
```

Methods

This section contains the summary and detailed information for methods.

Method detail

getKeyPath ()

Returns the path to the LithiumSSO encryption key.

```
public System.String getKeyPath()
```

getServerId ()

Returns the Server ID.

```
public System.String getServerId()
```

getClientDomain ()

Returns the Client Domain used to set the Cookie domain.

```
public System.String getClientDomain()
```

getClientId ()

Returns the Client ID.

```
public System.String getClientId()
```

getLithiumCookieValue (uniqueId, login, email, settingString, reqUserAgent, reqReferer, reqRemoteAddr)

Returns an encrypted String to be used for an authentication token.
The `init()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **settingString** - name/value pairs for user profile fields. (optional)
- **reqUserAgent** - [`HttpServletRequest.getHeader("User-Agent")`] used for security identification information
- **reqReferer** - [`HttpServletRequest.getHeader("Referer")`] used for security identification information
- **reqRemoteAddr** - [`HttpServletRequest.getRemoteAddr()`] used for security identification information

Throws: `SSOException` if initial settings are invalid.

```
public System.String getLithiumCookieValue(  
    System.String uniqueId,  
    System.String login,  
    System.String email,  
    System.String settingString,  
    System.String reqUserAgent,  
    System.String reqReferer,  
    System.String reqRemoteAddr) throws SSOException
```

See Also: `init(String newkeyPath, String newClientId, String newClientDomain)`

getLithiumCookieValue (uniqueId, login, email, settingString, request)

Returns an encrypted String to be used for an authentication token.
The `init()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **settingString** - name/value pairs for user profile fields. (optional)
- **request** - `HttpServletRequest` used for security identification information

Throws: `SSOException` if initial settings are invalid.

```
public static System.String getLithiumCookieValue(  
    System.String uniqueId,  
    System.String login,  
    System.String email,  
    System.String settingString,  
    System.Web.HttpRequest request) throws SSOException
```

See Also: `init(String newkeyPath, String newClientId, String newClientDomain)`

getLithiumCookieValue (uniqueId, login, email, settings, request)

Returns an encrypted String to be used for an authentication token.
The `init()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **settings** - name/value pairs for user profile fields. (optional)
- **request** - `HttpRequest` used for security identification information

Throws: `SSOException` - if initial settings are invalid.


```
public static System.String getLithiumCookieValue(
    System.String uniqueId,
    System.String login,
    System.String email,
    System.Collections.Hashtable,
    System.Web.HttpRequest request) throws SSOException
```

See Also: `init(String newKeyPath, String newClientId, String newClientDomain)`

getLithiumCookie (uniqueId, login, email, request)

Returns an `HttpCookie` containing an encrypted authentication token that can be passed to the client (usually via `HttpResponse`) for seamless login and registration on the Community servers. The `init()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **request** - `HttpRequest` used for security identification information

Throws: `SSOException` - if initial settings are invalid.

```
public static System.Web.HttpCookie getLithiumCookie(
    System.String
    System.uniqueId,
    System.String login,
    System.String email,
    System.Web.HttpRequest request)
```

writeLithiumCookie (uniqueId, login, email, settings, request, response)

Writes an encrypted authentication token cookie directly to the client via the `HttpResponse` object for seamless login and registration on the Community servers. The `init()` method must be called before this method is accessed.



Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community
- **settings** - name/value pairs for user profile fields.
- **request** - `HttpRequest` used for security identification information
- **response** - `HttpResponse` used to write the SSO cookie

Returns: `true` if the token was written successfully

Throws: `SSOException` if initial settings are invalid.

```
public static Boolean writeLithiumCookie(  
    System.String uniqueId,  
    System.String login,  
    System.String email,  
    System.Collections.Hashtable,  
    System.Web.HttpRequest request,  
    System.Web.HttpResponse response)
```

See Also: `init(String newKeyPath, String newClientId, String newClientDomain)`

writeLithiumCookie (uniqueId, login, email, request, response)

Writes an encrypted authentication token cookie directly to the client via the `HttpResponse` object for seamless login and registration on the Community servers. The `init()` method must be called before this method is accessed.

Parameters:

- **uniqueId** - unique identified used on the client User system
- **login** - User's login display name to be used on Community. The following characters are NOT allowed: < > () [] \ / ' "
- **email** - User's email address to be used on Community

- **request** - `HttpRequest` used for security identification information
- **response** - `HttpResponse` used to write the SSO cookie

Returns: `true` if the token was written successfully

Throws: `SSOException` if initial settings are invalid.

```
public static Boolean writeLithiumCookie(  
    System.String uniqueId,  
    System.String login,  
    System.String email,  
    System.Web.HttpRequest request,  
    System.Web.HttpResponse response)
```

See Also: `init(String newKeyPath, String newClientId, String newClientDomain)`

removeLithiumCookie (request, response)

Removes the LithiumSSO cookie from the user's cookie cache. Call this method when a user logs out from the client system.

Parameters:

- **request** - `HttpRequest` used for security identification information
- **response** - `HttpResponse` used to remove the SSO cookie with

```
public static void removeLithiumCookie(  
    System.Web.HttpRequest request,  
    System.Web.HttpResponse response)
```

PHP reference

This section is a PHP reference of summary and detailed code information.

Class summary

`Lithium_SSO`

Constants

This section contains constants information.

Constants summary
\$lithium_separator = " " The character used as a separator
\$lithium_version = "LiSSOv1.5" The current version of the LithiumSSO
\$lithium_cookie_name = "lithiumSSO:" The name of the LithiumSSO cookie

Variables

This section contains variables information.

Variables summary
\$client_id The client or community ID to create an SSO token for. This is typically the same as community ID unless you have multiple communities. In that case, this is a value that you and Khoros decide on.
\$client_domain The domain name for this token, used when transporting via cookies.
\$sso_key The 128-bit or 256-bit secret key, represented in hexadecimal
\$pg_key The client's secret PrivacyGuard key (128-bit or 256-bit) Note: PrivacyGuard is a separate Khoros product that enables Community to send email messages to community members without actually knowing what the user's email

Variables summary

address is. Contact your Customer Success Manager (CSM) for more information about PrivacyGuard.

Functions

This section contains functions information.

Functions summary

lithium_sso

Initializes the LithiumSSOClient.

get_auth_token

Returns a Khoros authentication token for the given user parameters.

Get_auth_token_value

Returns a Khoros authentication token for the given user parameters.

encode

Returns an encrypted representation of the specified string.

get_random_iv

Returns a random initialization vector for AES with the specified length. The returned string is URL-safe.

get_token_safe_string

Returns a token-safe representation of the specified string. Used to ensure that the token separator is not used inside a token.

init_smr

Initializes the PrivacyGuard key, if PrivacyGuard is used.

Note: PrivacyGuard is a separate Khoros product that enables Community to send email messages to community members without actually knowing what the user's email address is. Contact your Customer Success Manager (CSM) for more information about PrivacyGuard.

get_smr_field

Returns the encrypted PrivacyGuard token.

Functions summary

get_server_var

Returns the \$_SERVER variable by the specified name.

Functions detail

lithium_sso

```
lithium_sso($client_id, $client_domain, $sso_hex_key)
```

Initializes the LithiumSSOClient.

Parameters:

\$client_id – The client or community ID for which to create an SSO token. Your client ID is assigned by Khoros and uniquely identifies your company or your community. If you have only one community, the client ID is likely to be an abbreviated form of your company name. If you have multiple communities, each community must have a unique ID. In that case, you and Khoros decide on the IDs for each community.

\$client_domain – The domain name for this token, used when transporting via cookies

\$sso_hex_key – **The 128-bit or 256-bit secret key, represented in hexadecimal**

get_auth_token

```
get_auth_token($unique_id, $login, $email, $settings_array)
```

Returns a Khoros authentication token for the given user parameters.

Parameters:

\$unique_id - A non-changable ID used to uniquely identify this user globally. This should be a non-reusable integer or other identifier. E-mail addresses can be used, but are not recommended as this value cannot be changed.

\$login – The login name or screen name for this user. This is usually a publicly visible field, so should not contain personally identifiable information. This name appears on all messages the user posts in the community.

\$email – The e-mail address for this user.

Functions detail

\$settings_array - An associative array of profile settings => value pairs.

Examples of settings include:

roles.grant = Moderator (grants the Moderator role to user)

profile.name_first = John (sets first name to John)

get_auth_token_value

```
get_auth_token($unique_id, $login, $email, $settings_array,  
$req_user_agent, $req_referer, $req_remote_addr)
```

Returns a Khoros authentication token for the given user parameters.

Parameters:

\$unique_id A non-changable ID used to uniquely identify this user globally. This should be an non-reusable integer or other identifier. E-mail addresses can be used, but are not recommended as this value cannot be changed.

\$login – The login name or screen name for this user. This is usually a publicly visible field, so should not contain personally identifiable information. This name appears on all messages the user posts in the community.

\$email – The e-mail address for this user.

\$settings_array An associative array of profile settings => value pairs.

Examples of settings include:

roles.grant = Moderator (grants the Moderator role to user)

profile.name_first = John (sets first name to John)

\$req_user_agent = The user agent of the browser making the request. Used for security identification information.

\$req_referer = The "referer" HTTP Header of the current request. Used for security identification information.

\$req_remote_addr = The remote IP Address of the current request. Used for security identification information, specifically to determine that the IP Address in the sso token matches the IP Address of the request (to prevent spoofing of the SSO token).

Functions detail

encode

```
encode($string, $key)
```

Returns an encrypted representation of the specified string.

Parameters:

\$string – The string to encode

\$string – The key to use

get_random_iv

```
get_random_iv($length)
```

Returns a random initialization vector for AES with the specified length. The returned string is URL-safe.

Parameters:

\$length – The length of the IV to return, in bytes

get_token_safe_string

```
get_token_safe_string($string)
```

Returns a token-safe representation of the specified string. Used to ensure that the token separator is not used inside a token.

Parameters:

\$string – The string for which a token-safe representation is being returned

init_smr

```
init_smr($pg_hex_key)
```

Initializes the PrivacyGuard key.

Note: PrivacyGuard is a separate Khoros product that enables Community to send email messages to community members without actually knowing what the user's email address is. Contact your Customer Success Manager (CSM) for more information about PrivacyGuard.

Functions detail

Parameters:

\$pg_hex_key – The 128-bit or 256-bit PrivacyGuard key, represented in hexadecimal

get_smr_field

```
get_smr_field($string)
```

Returns the encrypted PrivacyGuard token.

Parameters:

\$string – The string for which a PrivacyGuard encrypted token is being returned

get_server_var

```
get_server_var($name)
```

Returns the `$_SERVER` variable by the specified name.

Parameters:

The initialization optionally takes a server ID. When the SSO API issues cookies, it also issues a one-time-use ID to prevent cookies from being reused. This ID is specific to the instance of the SSO API. By default, this is the IP address of the system the API is on. If there are multiple SSO API instances (e.g. multiple JVMs running on a single computer), a more specific ID is needed (e.g. IP and port). The server ID can be any string that is unique to each SSO API instance. It is important that this value maintain its uniqueness for the life of the SSO API instance. If you choose a value other than internal IP:port, Khoros strongly recommends that you maintain a mapping of server ID to the SSO API that generated it. A mapping of server ID to SSO API instance aid in troubleshooting of SSO authentication errors.

\$name – The name of the `$_SERVER` variable to return

LithiumSSO FAQ

What type of encryption does the LithiumSSO client use?

The LithiumSSO client uses 256-bit Advanced Encryption Standard (AES) cryptography.



Should we use a different clientID for stage vs. production?

Yes. Community uses the `clientId` in the cookie name it creates (and tries to delete it after processing the SSO token), so if you use different client IDs for stage and production, the correct SSO cookies are deleted right after they are processed and thus won't be read again when processing SSO cookies. We recommend using this client format: `<community>.<phase>`. For example, Khoros's stage `clientId` would be `khoros.stage` and its production `clientId` would be `lithium.prod`.

For reference, by `clientId`, we are referring to one of the values you pass in when you create the SSO client instance – in Java it's the second parameter passed into this call:

```
LithiumSSOClient.getInstance(key, clientId, clientDomain, serverId);
```

We have several instances on our production environment. Is it okay to set the same server ID for all instances or should server IDs be unique?

You should *always use unique server IDs for each instance* of the SSO client that you create. Otherwise, there could be clashes between tokens generated by each instance. We do have code in the Java to prevent this from happening with multiple instances using the same server ID, but you should use a different server ID for each instance. Additionally, it helps you identify if a particular SSO client instance is configured wrong from the SSO token.