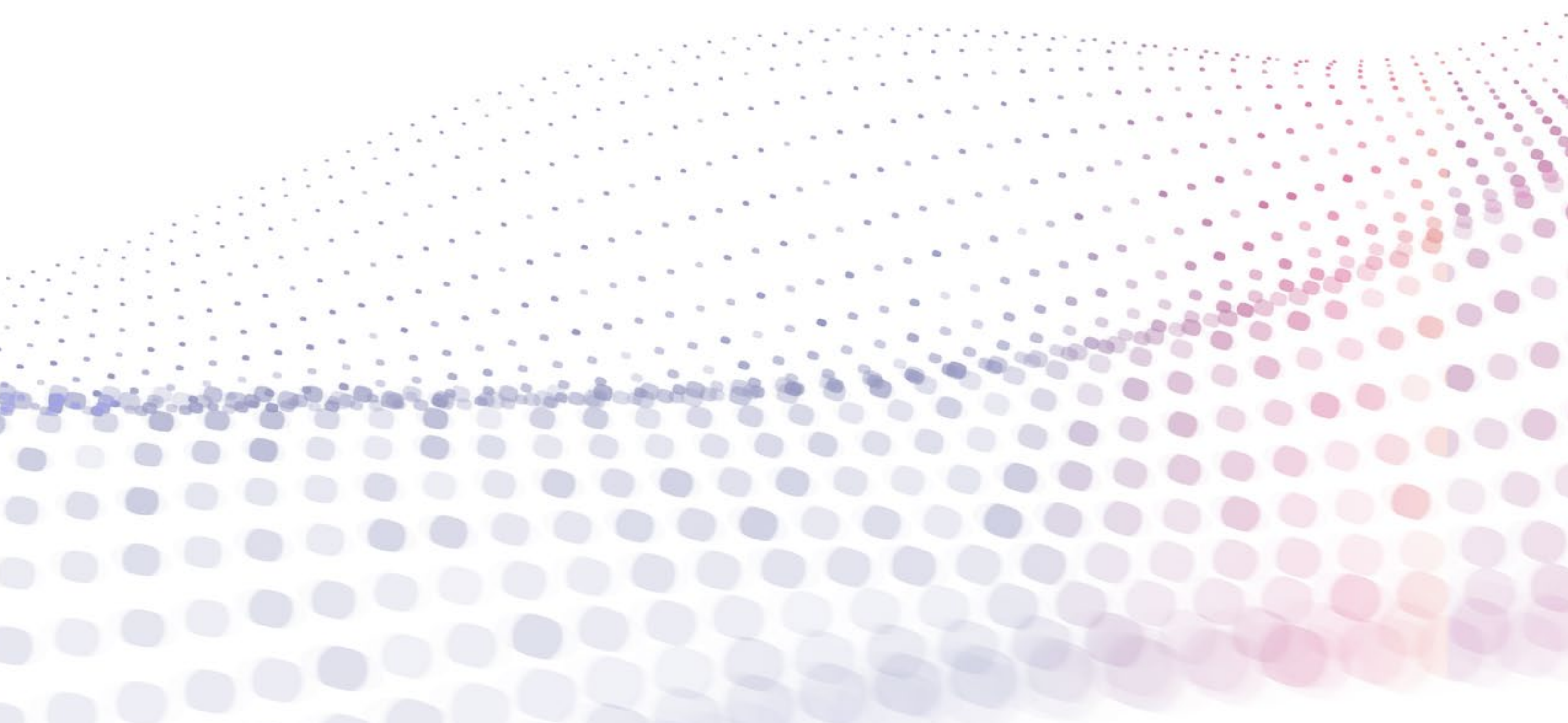
MarkLogic®

MarkLogic Integration with Single Sign-On

MARKLOGIC WHITE PAPER · JUNE 2018

It's easy for digital information to become a lot to manage. Users often find themselves overwhelmed with usernames and passwords when required to submit login credentials each time they use an application—especially across multiple applications. These login verifications may conflict with the security policies of the corporation itself, or simply become an inconvenience for users; as such, many corporations turn to single sign-on (SSO) solutions. We will explain how applications built on MarkLogic can be integrated with SSO solutions at the database layer in order to prevent passwords from being directly sent to MarkLogic. We hope to provide useful insight on how to take full advantage of MarkLogic's flexible security features to better secure access authentication.



Contents

Introduction

To facilitate user authentication, out-of-the-box MarkLogic can integrate directly with a Lightweight Directory Access Protocol (LDAP) directory like Active Directory or a Security Assertion Markup Language (SAML) based authentication system. This type of direct integration either requires end users to type in their login information or requires the passwords to be passed to MarkLogic. Single sign-on (SSO) is a common solution that enterprises adopt that allows a user to type login credentials once to access multiple applications. Whether for convenience, or because an organization's security policies prohibit passwords from being sent to downstream applications, MarkLogic applications can be integrated with SSO solutions so that passwords are not retype or sent to MarkLogic, as detailed below.

Where MarkLogic Resides in your System Architecture

Before integrating MarkLogic with an SSO system, we first need to understand where MarkLogic resides in the overall system architecture. There are three common areas that MarkLogic can reside within a system architecture.

1. MarkLogic receives requests directly, or requests go through a load balancer and hit MarkLogic directly. All logic that would need to be secured would need to be within MarkLogic.
2. MarkLogic resides behind a reverse proxy, indicating that most logic is within MarkLogic. Reverse proxies can be used to serve up static content (e.g. images, CSS, JavaScript) as well as intercept requests in order to add information for security or be used for an added layer of security. For example, you can configure a reverse proxy to only allow requests to MarkLogic from the reverse proxy's IP address.
3. Another server handles business logic and queries MarkLogic through one of the API connectors such as the Java API[1] . Most—if not all—of the logic is handled in a system outside of MarkLogic, and MarkLogic simply executes the queries sent to it.

SSO Integration at the Database Layer

The best way to integrate a MarkLogic application with an SSO solution is to do it directly with MarkLogic so that security is controlled at the database layer. This is accomplished using the rewriter which intercepts all the requests sent to an application server and dispatches the requests to the desired module file based on code instruction or configuration. There are two types of rewriters in MarkLogic: the Declarative XML Rewriter[2] (released as part of version 8), and the Interpretive Rewriter[3].

If you are not using the out-of-the-box REST APIs, use the Interpretive Rewriter; this will allow you to execute code that integrates with the SSO system and send the user's information to MarkLogic in the

1 https://developer.marklogic.com/products/java

2 https://docs.marklogic.com/guide/app-dev/XMLrewriter

3 https://docs.marklogic.com/guide/app-dev/rest

HTTP headers. While a frontend application can send user information directly, there more often is a middle layer that exists between MarkLogic and the user's request, such as a load balancer, reverse proxy, or middle-tier. When you need to pass more information to MarkLogic than a load balancer can easily do, use a reverse proxy or a middle-tier.

Once MarkLogic has received the user information, read the user information and give the current user more privileges using xdmp:login[4]. The xdmp:login function's fourth parameter takes role names that will get added to the user you login with, which elevates the user's privileges for the current session. Note that xdmp:login requires you to be using application level authentication and does not work with XDBC servers.

```
(:
gets the header userID and gives them an extra role if its there
This is a basic read example below about how to securely use xdmp:login
:)

let $userID := xdmp:get-request-header('userID')

let $roles :=
   if (fn:exists($userID)) then (
      ("secret","public")
   ) else ("public")

return
 xdmp:login(xdmp:get-current-user(), (), fn:false(), $roles)
```

Securing the Integration

By passing MarkLogic the user's information and using xdmp:login (in effect elevating privileges), you are essentially adding a point of attack. We can secure the potential vulnerability by using a few of MarkLogic's security features, including:

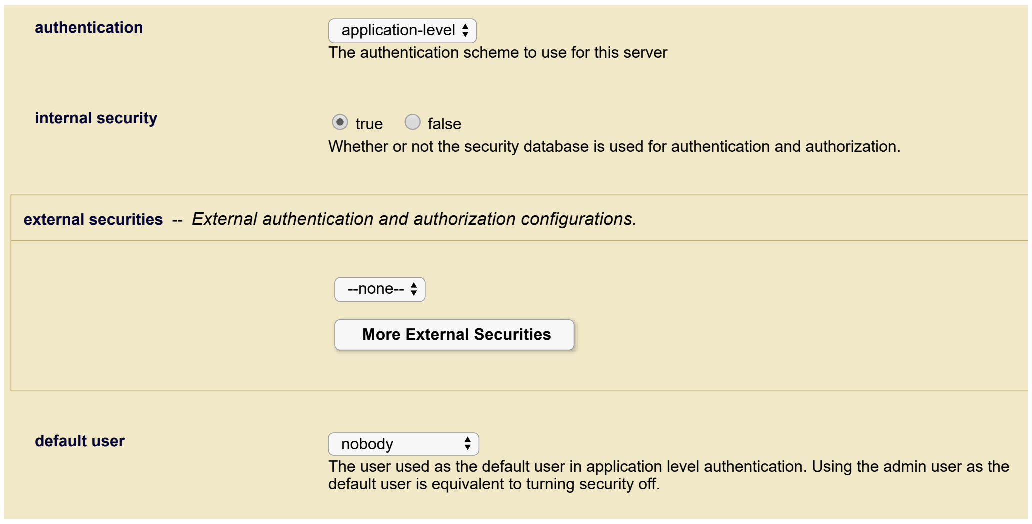
- Limiting the default user of the Application Server[5]
- Limiting the roles that can be used during xdmp:login function
- Using LDAP to lookup the roles
- Using an amp[6] when calling the xdmp:login function
- Restricting the application server requests to only machines that have certificates that we have previously authenticated

The first step in securing the SSO integration is to give the default user of the application server as little privileges as needed. Using xdmp:login requires setting the application server authentication type to "application-level," and also requires a default user. The default user is the user that will run before we call xdmp:login and needs to have read-access to the rewriter file and modules database, as well as any library files that it references. The default user will also need privileges to any functions you are calling that require them, such as xdmp:login.

4 https://docs.marklogic.com/xdmp:login

5 https://docs.marklogic.com/guide/security/authentication#id_60432

6 https://docs.marklogic.com/guide/admin/security#id_81246



authentication	application-level The authentication scheme to use for this server
internal security	● true ○ false Whether or not the security database is used for authentication and authorization.

external securities -- *External authentication and authorization configurations.*

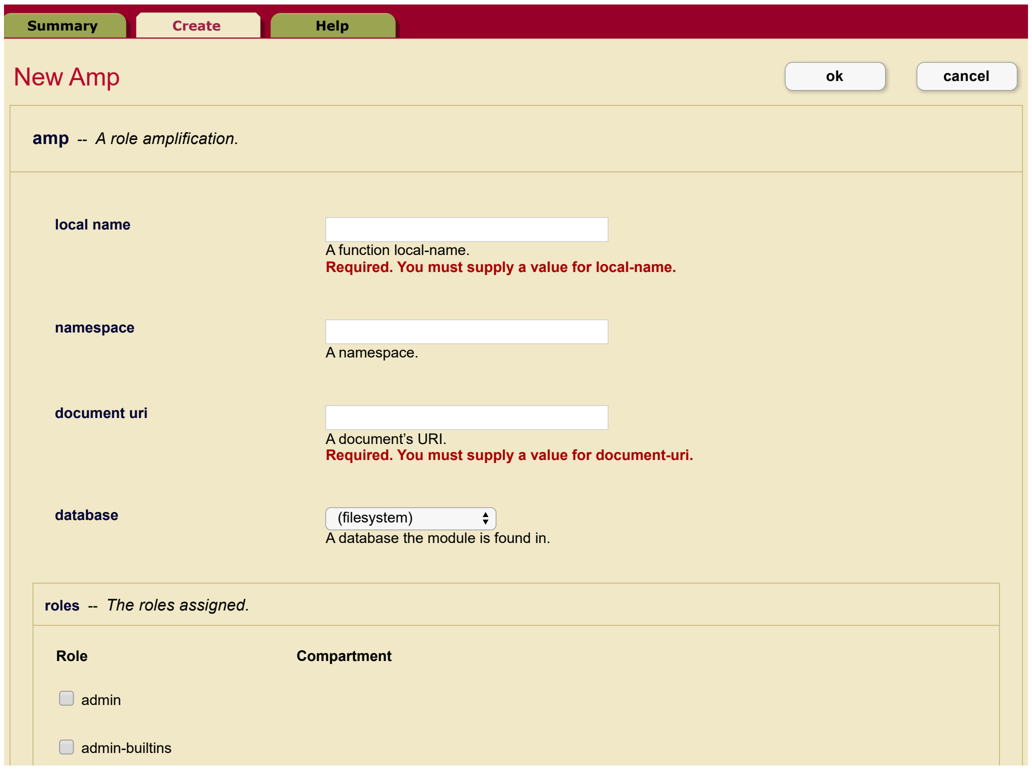
--none--

More External Securities

default user	nobody The user used as the default user in application level authentication. Using the admin user as the default user is equivalent to turning security off.

Figure 1: Setting the Application Server Authentication Type and Default User in the Admin UI

To make it more secure, the privileges for the default user can be elevated in an amp— a feature in MarkLogic that temporarily gives the user additional roles while executing a function. By doing this, we can create a role that will have the needed privileges to call functions like xdmp:login and xdmp:get-request-header[7], preventing the default user from being used in an unexpected way (like in qconsole[8]) as well as gain the xdmp:login privileges.



New Amp

amp -- *A role amplification.*

local name	A function local-name. **Required. You must supply a value for local-name.**
namespace	A namespace.
document uri	A document's URI. **Required. You must supply a value for document-uri.**
database	(filesystem) A database the module is found in.

roles -- *The roles assigned.*

Role	Compartment
☐ admin	
☐ admin-builtins	

Figure 2: Creating a New Amp in the Admin UI

7 https://docs.marklogic.com/xdmp:get-request-header

8 https://docs.marklogic.com/guide/qconsole/walkthru

Since xdmp:login can take any role we give it, it is particularly important to be cautious of giving users too much power by limiting the roles that can be sent to it.

We suggest limiting user roles by creating a set of users with different roles based on the necessary permissions. If you are unable to maintain local MarkLogic users and are unable to use this method, try assigning the roles directly, but do not pass them in via the fourth argument of xdmp:login. For example, you might hard-code roles with if-or switch statements, or there might be a role mapping document that maps user ID to MarkLogic role. You might have something that passes in the role to xdmp:login from the middle layer.

Since MarkLogic has so many roles, it's best to only allow roles that you expect. This can easily be done by putting a predicate on the fourth parameter of xdmp:login.

```
(: only allows roles that are expected :)

let $expectedRoles := (“public”,”classified”,“secret”,”top-secret”)

(: would likely be passed in or dynamic:)
let $roles := (“admin”,”public”)

return
   xdmp:login(“default-user”, (), fn:true(), $roles[. eq $expectedRoles])
```

To further secure the integration, you can have an LDAP/AD server own the role assignment. During the authentication part of the rewriter, you can use xdmp:ldap-lookup[9]. You'll need credentials in order to authenticate to the LDAP server, which is typically done with a service account. You'll then pass some way to identify the user that needs to be authenticated into MarkLogic, e.g. their LDAP Common Name (CN). This will give you results of user payloads, and if you use a unique ID like the CN, there should only be one result. The payload has memberOf LDAP attribute elements. You can xpath to these and filter out the ones that you want to add as roles. You would then take those roles and pass them into the xdmp:login function.

Hint: Use a prefix when naming roles, such as marklogic-role-name.

Here is some sample code for what that would look like:

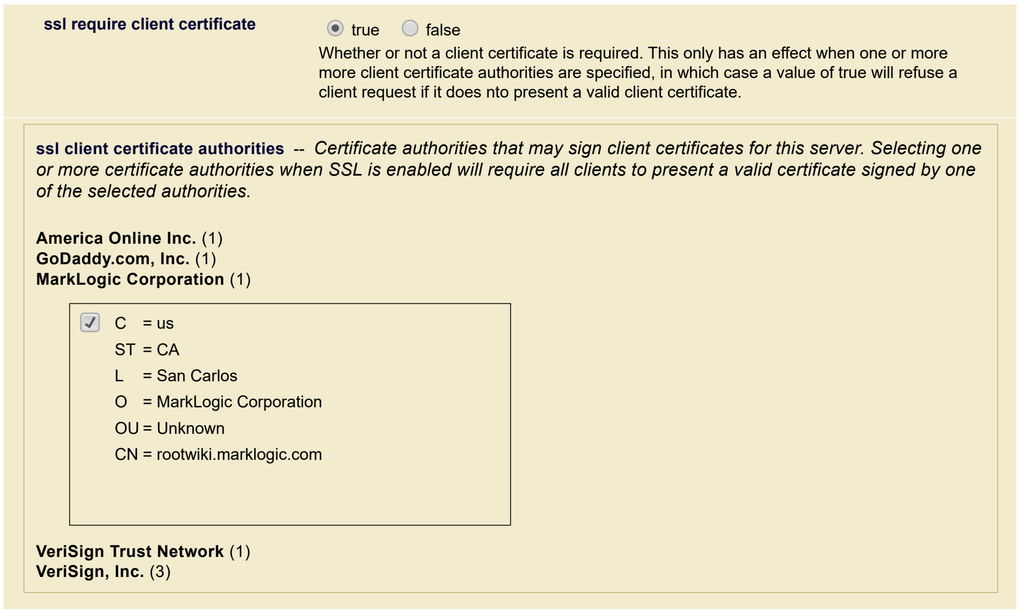
```
(:
$cn would be passed from the SSO system, but could come through as a header that marklogic would
look up
You would need to change the path, username password and server-uri
:)

$ldapResults :=
xdmp:ldap-lookup(
   "CN=”|| $cn || “,CN=Users,DC=MLTEST1,DC=LOCAL",
   <options xmlns="xdmp:ldap">
      <username>admin</username>
      <password>admin</password>
      <server-uri>ldap://dc1.mltest1.local:389</server-uri>
   </options>)

$groups := $ldapResults/ldap-attribute[@id eq “memberOf"]
```

9 https://docs.marklogic.com/xdmp:ldap-lookup

```
(: gets the groups that just have the prefex we want :)
$MLgroups:= $groups[fn:starts-with(., “CN=marklogic-”)][1]

(: only takes the group cn :)
let $roles := $MLgroups[fn:substring-after(fn:substring-before(., “,”), “CN=marklogic-”)]

(: only allows roles that are expected :)
let $expectedRoles := (“public”,”classified”,“secret”,”top-secret”)

return
   xdmp:login(“default-user”, (), fn:true(), $roles[. eq $expectedRoles])
```

Securing the Connection

Once the amount of elevated permissions has been secured, the connection should be secured in order to ensure that the correct machines are sending the requests. The requests can be restricted to machines that are allowed to send requests to the application server running the code. This can be done by using SSL and setting “ssl require client certificate[10]” to true on the application server configuration. Typically, an SSL certificate has a Fully Qualified Domain Name, but you can also give it IP addresses. This allows you to add another layer of security on top of the SSL certificate in case the certificate is compromised. This will require intruders to also perform IP spoofing over HTTPS with 2-way SSL.



ssl require client certificate [✓ true] false

Whether or not a client certificate is required. This only has an effect when one or more more client certificate authorities are specified, in which case a value of true will refuse a client request if it does nto present a valid client certificate.

ssl client certificate authorities -- *Certificate authorities that may sign client certificates for this server. Selecting one or more certificate authorities when SSL is enabled will require all clients to present a valid certificate signed by one of the selected authorities.*

America Online Inc. (1)
GoDaddy.com, Inc. (1)
MarkLogic Corporation (1)

✓ C = us
ST = CA
L = San Carlos
O = MarkLogic Corporation
OU = Unknown
CN = rootwiki.marklogic.com

VeriSign Trust Network (1)
VeriSign, Inc. (3)

Figure 3: Setting “ssl require client certificate” Attribute for the Application Server in the Admin UI

10 https://help.marklogic.com/Knowledgebase/Article/View/350/0/configuring-marklogic-https-application-servers-with-ssl-client-authentication-for-a-trusted-ca-certificate-without-a-named-organization-field-in-the-subject
https://docs.marklogic.com/guide/security/SSL#id_37057

Figure 4 provides an example architecture diagram where the connection is secured. The request starts in the browser and gets sent to the server that intercepts it, i.e. the load balancer or the reverse proxy. This server has its own certificate and certificates of the MarkLogic servers. When the server directs the requests to the MarkLogic server, it sends its certificate and verifies the MarkLogic server's certificate. The MarkLogic server will validate the sent certificate along with the request coming in, which assures that the request is coming from an authorized machine.

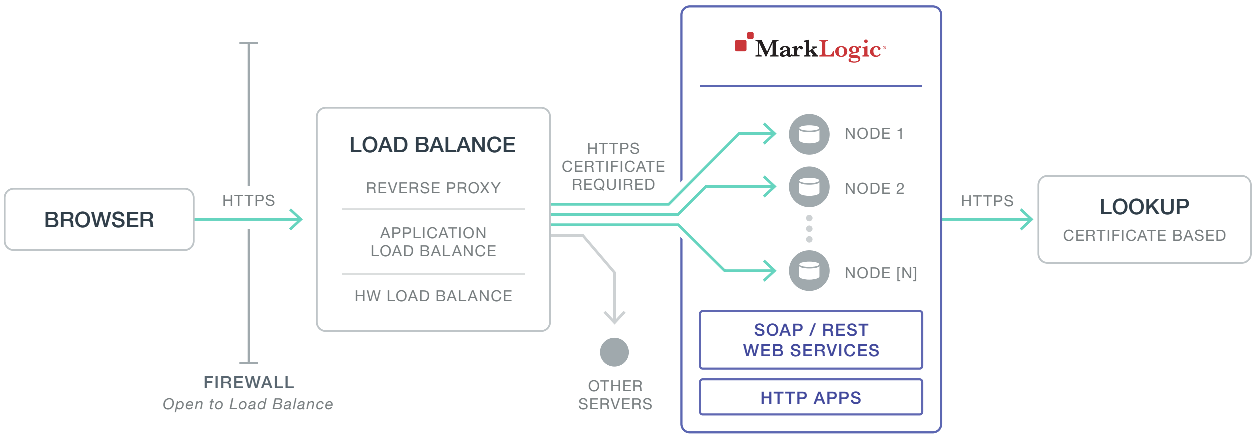


Figure 4: Example Architecture Diagram where the Connection is Secured

The Set-Session Parameter in xdmp:login

It can be time-consuming to login with every request; in order to save time, you can have MarkLogic save sessions for the users. By default, xdmp:login does not save the login user and roles. You can tell xdmp:login to do that with the third parameter, the set-session parameter. The set-session parameter defaults to false, but if you set it to true, it will keep the role information in a session. This session is tied to whichever MarkLogic server handled the request.

If you are going to save the sessions, you'll want to update the login code to check to see if there is a session. You can do this with xdmp:set-session-field[11] and xdmp:get-session-field[12]. These functions require extra execute privileges, so you'll want to update the role that is being amp to have them. You will also want to configure your load balancer to have sticky sessions, ensuring that all requests within a session are sent to the same MarkLogic server.

11 https://docs.marklogic.com/xdmp:set-session-field

12 https://docs.marklogic.com/xdmp:get-session-field

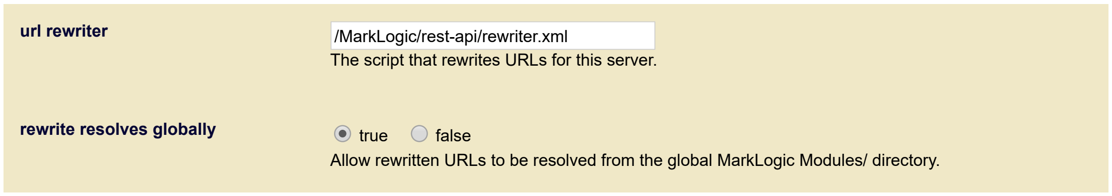
Note that keeping the sessions saved across multiple connections creates a higher security risk. The session could be intercepted and then it would have the extra permissions. It's important to make sure that the sessions are secured when using this option by using 2-way SSL.

```
(: checks to see if there is a loggin session before loggin in :)

if (xdmp:get-session-field("loggin")) then (

) else (
   let $expectedRoles := ("public","classified","secret","top-secret")

   (: would likely be passed in or dynamic:)
   let $roles := ("admin","public")

   (: sets the loggin session so we don't have to login again :)
   let $_ := xdmp:set-session-field("loggin", fn:true())

   return
   xdmp:login("default-user", (), fn:true(), $roles[. eq $expectedRoles])

)
```

Declarative XML Rewriter and Dealing with Out-of-the-Box REST Endpoints

The process to integrate MarkLogic with an SSO system using the Declarative XML Rewriter is very similar to the Interpretive Rewriter with a few extra steps. First, you need to know what dispatch element you want to have integrated. Most of the time, people want to integrate the out-of-the-box REST Endpoints[13] as well as any custom extensions[14].

A global XML rewriter configuration file is shipped with MarkLogic, but changing this file directly is strongly discouraged. Instead, copy the global XML configuration to your project code, edit it there, and set the application server's rewriter to read the project configuration XML. The global XML rewriter configuration file is located at [MarkLogic-Install-Directory]/Modules/MarkLogic/rest-api/rewriter.xml. Simply edit the dispatch elements that you want to change. Change the URL rewriter to a local module file that does the same code as before but also does the login logic mentioned above in the "SSO Integration at the Database Layer" section.



url rewriter	/MarkLogic/rest-api/rewriter.xml The script that rewrites URLs for this server.
rewrite resolves globally	[✓ true] false Allow rewritten URLs to be resolved from the global MarkLogic Modules/ directory.

Figure 5: Editing the URL rewriter in the Admin UI

13 https://docs.marklogic.com/guide/rest-dev
https://docs.marklogic.com/REST

14 https://docs.marklogic.com/guide/rest-dev/extensions

For example, if you want to change the custom extensions and the out-of-the-box REST endpoints, copy the global rewriter, locate the dispatch element you want to update by looking for the matches attribute, and change the dispatch element to your own code:

```
<match-path matches="^/(v1|LATEST)/resources/([^/]+)/?$">
   <match-query-param name="database">
      <set-database checked="true">$0</set-database>
   </match-query-param>
   <add-query-param name="name">$2</add-query-param>
   <match-method any-of="GET HEAD POST">
      <match-query-param name="txid">
         <set-transaction>$0</set-transaction>
         <set-transaction-mode>query</set-transaction-mode>
      </match-query-param>
      <dispatch>/MarkLogic/rest-api/endpoints/resource-service-query.xqy</dispatch>
   </match-method>
   <match-method any-of="PUT DELETE">
      <match-query-param name="txid">
         <set-transaction>$0</set-transaction>
         <set-transaction-mode>update</set-transaction-mode>
      </match-query-param>
      <dispatch>/MarkLogic/rest-api/endpoints/resource-service-update.xqy</dispatch>
   </match-method>
</match-path>

(...)

<math-path matches="^/(v1|LATEST)/documents/?$">
  <match-query-param name="database">
     <set-database checked="true">$0</set-database>
  <match-query-param>
  <match-method any-of="POST">
    <match-query-param name="txid">
      <set-transaction>$0</set-transaction>
     <set-transaction-mode>update</set-transaction-mode>
   </match-query-param>
    <match-content-type any-of="application/x-www-form-urlencoded">
      <dispatch>/etc/sso/resources-wrapper.xqy</dispatch>
    </match-content-type>
  </match-method>
  <match-method any-of="GET HEAD OPTION">
    <match-query-param name="txid">
      <set-transaction>$0</set-transaction>
     <set-transaction-mode>query</set-transaction-mode>
    </match-query-param>
    <dispatch>/etc/sso/resources-wrapper.xqy</dispatch>
   </match-method>
  <match-method any-of="PUT POST DELETE PATH">
    <match-query-param name="txid">
      <set-transaction>$0</set-transaction>
     <set-transaction-mode>update</set-transaction-mode>
    </match-query-param>
    <dispatch>>/etc/sso/resources-wrapper.xqy</dispatch>
 </match-method>
<math-path>
```

Be sure to update the login code and add to whatever else it was already doing, like the example below:

```
xquery version "1.0-ml";

import module namespace ssolib = "example.marklogic.com/sso-lib" at "/ext/sso/lib.xqy";

declare private variable $MODULE-ENDPOINT-RESOURCES-READ := "/MarkLogic/rest-api/endpoints/
resource-services-query.xqy";
declare private variable $MODULE-ENDPOINT-RESOURCES-UPDATE := "/MarkLogic/rest-api/endpoints/
resource-services-update.xqy";

declare private variable $MODULE-ENDPOINT-DOCUMENTS-READ := "/MarkLogic/rest-api/endpoints/
document-item-query.xqy";
declare private variable $MODULE-ENDPOINT-DOCUMENTS-UPDATE := "/MarkLogic/rest-api/endpoints/
document-item-update.xqy";

declare private variable $MODULE-ENDPOINT-SEARCH-READ := "/MarkLogic/rest-api/endpoints/search-
list-query.xqy";
declare private variable $MODULE-ENDPOINT-SEARCH-UPDATE := "/MarkLogic/rest-api/endpoints/search-
list-update.xqy";

declare private variable $MODULE-ENDPOINT-SUGGEST := "/MarkLogic/rest-api/endpoints/suggest.xqy";
declare private variable $MODULE-ENDPOINT-TRANSACTIONS := "/MarkLogic/rest-api/endpoints/
transaction-item-default.xqy";

(: logic that we have shown before :)
let $_ := ssolib:login-User()

let $context := map:map()
let $params := map:map()

(: puts all the rs feilds in the params map :)
let $_ :=
   for $field in xdmp:get-request-field-names()[fn:starts-with(.,'rs:') or . eq ("uri")]
   let $key := replace($field, "rs:", "")
   return map:put($params, $key, xdmp:get-request-field($field))

let $method := fn:lower-case(xdmp:get-request-method())
return
   if(fn:starts-with($original-url,'/v1/resources/')) then
      if($method = "put" or $method = "delete") then
         xdmp:invoke($MODULE-ENDPOINT-RESOURCES-UPDATE)
      else
         xdmp:invoke($MODULE-ENDPOINT-RESOURCES-READ)

      else if(fn:starts-with($original-url,'/v1/documents/')) then

         if($method = "put" or $method = "post" or $method = "delete" or $method = "patch") then
            xdmp:invoke($MODULE-ENDPOINT-DOCUMENTS-UPDATE)
         else
            xdmp:invoke($MODULE-ENDPOINT-DOCUMENTS-READ)

         else if(fn:starts-with($original-url,'/v1/search/')) then

            if($method = "delete") then
               xdmp:invoke($MODULE-ENDPOINT-SEARCH-UPDATE)
            else
               xdmp:invoke($MODULE-ENDPOINT-SEARCH-READ)

            else if(fn:starts-with($original-url,'/v1/suggest/')) then
               xdmp:invoke($MODULE-ENDPOINT-SUGGEST)

   else if(fn:starts-with($original-url,'/v1/transactions/')) then
      xdmp:invoke($MODULE-ENDPOINT-TRANSACTIONS)
```

When upgrading MarkLogic, keep in mind that the application uses a copy of the out-of-the-box global XML rewriter and you will need to merge any changes.

Integrating in the Middle Layer

Integration at the MarkLogic layer is strongly encouraged in order to maintain security at the database layer. However, if your resources are dedicated to your middle layer, or if all of your business logic is handled in the middle layer, integration in the middle layer is possible. Make an eval[15] call to MarkLogic before any other calls are made to it. In this eval call, call the xdmp:login function with the extra roles and have the set-session parameter set to true to create a session with the extra privileges. Be sure to have the load balancer configured to have all requests for each session sent to the same MarkLogic server. You will also want to do all the steps discussed in the “Securing the Integration” section.

Summary

Many organizations look to SSO solutions to improve both user experience and system security. When incorporating an application built on top of MarkLogic into an SSO architecture, we recommend leveraging MarkLogic’s built-in security features and integrating at the database layer.

Key Resources

MarkLogic Security Guide – Authenticating Users:
https://docs.marklogic.com/guide/security/authentication

MarkLogic Security Guide – External Security:
https://docs.marklogic.com/guide/security/external-auth

Application Developer’s Guide – Declarative XML Rewriter
https://docs.marklogic.com/guide/app-dev/XMLrewriter

White Paper – Building Security Into MarkLogic
https://www.marklogic.com/resources/building-security-marklogic/

15 https://docs.marklogic.com/9.0/REST/POST/v1/eval
https://docs.marklogic.com/guide/java/resourceservices#id_70532

MARKLOGIC CORPORATION
999 Skyway Road, Suite 200 San Carlos, CA 94070
+1 650 655 2300 | +1 877 992 8885 | www.marklogic.com | sales@marklogic.com