MATRIXMAXX™
Tying It All Together.

This document explains the MatrixMaxx AMS/CRM's approach to Single Sign-On (SSO). It is primarily targeted for external third party consumption at the kickoff of SSO integration efforts. We support 2 standard SSO protocols: Central Authentication Server (CAS) protocol and Oauth 2.0. However, we can work with other options as well.

INTRODUCTION

Single Sign-on (SSO) is the holy grail of system integration. The user logs in one time on one system and then has seamless access to all resources that the association has to offer: Content, Profile Management, Meeting Registration, Library, Career Center, Store, Accreditation System, Education/CEU tracking, etc.

What is Single Sign-on (SSO)

SSO is a situation where users log in once and can then access all related systems without logging in again. That login session remains active across all systems as long as the user is active on at least one, and when the user logs out it (ideally) terminates the session across all systems. (However, it is of note that Single-Sign-Off is in reality harder to achieve than Single-Sign-On and if actually desired needs to be discussed up front as it would potentially incur additional scope/cost in the effort.)

Here's an executive summary of how SSO works:

- The user tries to access a restricted 3rd party system/resource (career center, GR system, testing system, members-only content area, etc.) and is redirected to a login that is controlled by the master system (The AMS, MatrixMaxx)
- The user logs into MatrixMaxx and is then automatically redirected back to the original page/system with a special access token
- The 3rd party system uses this token as a key to access a MatrixMaxx webservice that returns data about that user (name, membership status, member type, lists, committees, demographics, etc.)
- The 3rd party system validates the data packet received from the AMS (MatrixMaxx), creates/updates the user in their own system, and logs them in

What is not SSO

Many people confuse SSO with solutions that involve lesser amounts of integration, such as a single password solution or a 'jump page' solution. While many solutions can achieve a certain level of integration, they are not the same either technically or in user experience. See the "Alternatives" section for more details

Making SSO a success

An SSO project between two or more systems should start with a meeting of the minds, both techno-geek and non-geek. Prior to this meeting, all SSO/API documentation should be shared so that all parties have a chance to review it in advance. Then, during this call/meeting …

- The association should share its vision of the user flow so that all parties are on the same page as to the desired end-user experience.
- The association should confirm the various access levels that they wish to see functional on the various sites/systems
    - Member vs. Nonmember
    - Will the member type come into play?
    - Is there any particular content for former or prospective members or are they treated like non-members?
    - Will committees have their own restricted areas? Are some committee users non-members of the association?
    - Will users fitting into particular lists or demographics have their own restricted areas? Will some of these users be non-members of the association?
- The group should confirm their respective URLs and basic functionality of all systems involved, and confirm whether we are in a cross-domain situation
- The AMS technical team should describe the proposed SSO solution and take feedback/answer questions
- If any of the vendors do not support the proposed solution, then alternate solutions should be discussed at this point
- The technical participants should share relevant development and contact information, including main technical contact, development URLs, IP ranges, etc.
- By the end of the call/meeting, there should be an agreed-upon solution that all involved parties deem technically sound and reasonable to implement.

The MatrixMaxx AMS

MatrixMaxx is an association management system (AMS). MatrixMaxx stores user profile data and transaction data.

- Users include members, former members, and nonmembers (prospects, vendors, hotels, etc.)
- Every user in the database has a username and password

- Email address is NOT REQUIRED, nor is it required to be unique. There are nulls and duplicate emails in our databases, intentionally put there by staff of the associations who use our database.
- This document assumes that the AMS is serving as the database of authority, or “Master” database. This means that that other systems are consuming data from the AMS, not writing to the AMS
- Transaction data can include meeting registrations, store purchases, membership dues, etc.
- The MatrixMaxx system automatically generates its own pages for user interactions with its transactional modules. i.e., The AMS generates a store, a meeting calendar, a meeting registration form, a shopping cart, etc.

MatrixMaxx is not a Content Management System (CMS), but rather is designed to integrate with and complement a CMS.

- A CMS is designed to specifically hold website content, such as new items, press releases, articles, blog posts, etc.
- MatrixMaxx has integrated with a variety of CMS products, including Ektron, SiteFinity, Expression Engine, WordPress, Mira, and a variety of custom-built solutions.

SINGLE SIGN-ON

Goals

The primary purpose for implementing SSO is to make the user experience as seamless as possible across different systems with regard to establishing and ending a login session. The three main aspects to accomplishing this are

- Requiring a user to authenticate only once in order to gain access to all involved systems
- Preventing the user’s login session from expiring due to inactivity as long as the user is actively using any of the related systems
- Terminating the user’s login sessions across all systems once the user performs a logout on any of the systems.

Database of Record

The MatrixMaxx AMS database contains user records, including credentials, for all user accounts. This will be the central database of record for all authentication purposes.

SSO Setup Checklist

When we are setting up SSO with a 3rd party, here are the basic tasks that need to be done ...

- Determine which protocol will be used: CAS, Oauth2.0, or other
- Determine if 3rd party will be wanting info provided in the initial auth response, or if they will be using the API to pull data
    - If initial auth response, the fields will need to be decided upon. [Note: there are limits to what can be sent in this initial response]
    - If API, that documentation will need to be provided, and more discussions will probably be necessary
- MatrixMaxx needs the domain names that we'll be redirecting to after login, so that we can whitelist these (i.e., to prevent a malicious redirect), for both DEV and LIVE on the 3rd party side
    - NOTE: No Firewall rules need to be adjusted; IPs are not necessary to know for SSO, but may be necessary for API
    - for CAS, we simply need the domain name to whitelist for redirect
    - for Oauth2, we need the exact URL to whitelist for redirect
- If CAS, Matrix will Provide domain to 3rd party for use in MatrixMaxx's CAS URL patterns (see below)
    - DEV ... www.CLIENT.maxx.matrixdev.net ... e.g., www.CLIENT.maxx.matrixdev.net/forms/cas/login?service={redirect URL}
    - LIVE/Production ... www.CLIENT.org ... e.g., www.CLIENT.org/forms/cas/login?service={redirect URL}
- If Oath2.0, Matrix will provide credentials similar to the following:
    - Your client id: VENDORNAME
    - Your client secret: abcd12349example5678
        - Authorization endpoint (dev server): http://www.CLIENT.maxx.matrixdev.net/forms/oauth/authorize
        - Token endpoint (dev server): http://www.CLIENT.maxx.matrixdev.net:26054/forms/oauth/token

Supported Protocols

If all client sites/applications reside behind the Matrix firewall, we sometimes accomplish SSO with a simple form-based authentication mechanism that sets a broad domain cookie that all sites can access.

However, if working with 3rd party outside systems is going to occur in either the short or long-term, MatrixMaxx supports two preferred SSO protocols

- Central Authentication Service (CAS) protocol. As of 2016 we are using version 2.0.
- OAuth2 (As of 2018)

CAS (Central Authentication Server) Protocol

As of 2016 we are using **Version 2.0 of the Central Authentication Service (CAS) protocol**.

CAS Flow: https://apereo.github.io/cas/5.0.x/protocol/CAS-Protocol.html

CAS 2.0 Specification:
https://apereo.github.io/cas/5.0.x/protocol/CAS-Protocol-V2-Specification.html

MatrixMaxx's CAS URLs in release 14.2 and later (ie, clients using the built-in Maxx CAS server, which is any client implemented after 2014, plus older clients who have upgraded) are:

Login URL:

- {baseURL}/forms/cas/login?service={redirect URL}
- Example:
  https://www.factweb.org/forms/cas/login?service=https%3A%2F%2Fwww.factweb.org)
- Redirect URL should be URI encoded
- NOTE: If your redirect URL has a different domain than the CAS site, a MatrixMaxx developer will need to enable access to it.

Logout URL:

- {baseURL}/forms/cas/logout?service={redirect URL}
- Example:
  https://www.factweb.org/forms/cas/logout?service=https%3A%2F%2Fwww.factweb.org)
- The redirect URL is optional and should be URI encoded. If it is omitted, the user will be redirected to the login page.
- NOTE: If your redirect URL has a different domain than the CAS site, a MatrixMaxx developer will need to enable access to it.

Service Validate URL:

- {baseURL}/forms/cas/serviceValidate?service={redirect URL}&ticket={login ticket value}
- Example:
  https://www.factweb.org/forms/cas/serviceValidate?service=https%3A%2F%2Fwww.factweb.org&ticket=LT-1473964398-fgYmjOzTcWKPi1I4m5Xp8LXol9WGiT64
- This request should be made server-side after the client has logged in and has been redirected with login ticket.
- The response will be in XML format and will contain CAS response fields if successful.

FOR OLDER INTEGRATIONS, ONLY : MatrixMaxx's CAS URLs in release 14.1 and prior (ie, clients using the RubyCAS server) are:

- {baseURL}/SSO/login (Eg. https://www.factweb.org/SSO/login)
- {baseURL}/SSO/logout (Eg. https://www.factweb.org/SSO/logout)
- {baseURL}/SSO/validate (Eg. https://www.factweb.org/SSO/validate)
- {baseURL}/SSO/serviceValidate (Eg. https://www.factweb.org/SSO/serviceValidate)

Extra Information

Often additional info about the user is desired: name, title, company, etc.

If extra information about the user is needed (firstName, lastName, organization name, demographics, etc) that could be retrieved in 2 ways:

- By using the read-only API the 3rd party would have access to just about any data they would want.

or

- If the 3rd party does not want to use the API, MatrixMaxx can return *some* extra information as part of the initial authentication response,
    - NOTE: this feature is outside of the standard CAS protocol, at least as of the 2016
    - NOTE: using this approach instead of an API call to get membership status and/or privilege group (authorization) info can result in the 3rd party having old/out-of-date info if the login session is long and/or if they are storing it.

As of 2016, we can return the following extra attributes in the serviceValidate response (with xml namespace 'maxx'). The exact list of what is returned depends on our XML settings file (the 'CASServer_authResponseAttributes' setting):

- firstName
- lastName
- CompanyName
- email
- id
- individualNum
- title
- prefix
- middleInitial (can contain a full middle name)
- suffix
- aliases (this is the user's 'nickname' field)
- username
- personalStatus (indicates whether the person is active user)
- PreferredPhone (looks for the 'best' number for a person. Individual workPhone first, then mainAddress phone, then current company's preferred phone)
- personalBackground (this is the bio text field in individual demographics)
- LinkToPicture
- MainAddressString (individual first, then company, unless they've explicitly specified their company address as their main address. line 1-3, city, state, prov, zip, country but NOT phones)
- JobFunctionStr (User's job function)
- MembershipStatusText (a string value combining the user's Membership type and Status (active/inactive)
- CompanyTypeStr
- CompanyLogo
- PrivilegeGroupStr (semicolon ';' delimited string of user's privilege groups)

Sample xml serviceValidate responses with extra parameters

Simple response with a few fields

```
<cas:serviceResponse xmlns:cas="http://www.yale.edu/tp/cas"
xmlns:maxx="http://www.w3.org/XML/">
 <cas:authenticationSuccess>
  <cas:user>matrixgroup</cas:user>
  <maxx:firstName>Matrix</maxx:firstName>
  <maxx:lastName>Group</maxx:lastName>
  <maxx:CompanyName/>
  <maxx:email>test@matrixmaxx.com</maxx:email>
 </cas:authenticationSuccess>
</cas:serviceResponse>
```

Response for an active Member of type International

```
<cas:serviceResponse xmlns:cas="http://www.yale.edu/tp/cas"
xmlns:maxx="http://www.w3.org/XML/">
<cas:authenticationSuccess>
<cas:user>TestIntCommissioner</cas:user>
<maxx:firstName>Testfirst</maxx:firstName>
<maxx:lastName>Testlast</maxx:lastName>
<maxx:CompanyName>Test International Company</maxx:CompanyName>
<maxx:email/>
<maxx:id>6C700000366</maxx:id>
<maxx:individualNum>15797</maxx:individualNum>
<maxx:personalStatus>Active Staff</maxx:personalStatus>
<maxx:prefix/>
<maxx:middleInitial/>
<maxx:suffix/>
<maxx:aliases>Test</maxx:aliases>
<maxx:user.username>TestIntCommissioner</maxx:user.username>
<maxx:JobFunctionStr/>
<maxx:MembershipStatusText>Active International
Member</maxx:MembershipStatusText>
<maxx:CompanyTypeStr/>
<maxx:user.PrivilegeGroupStr>Member-Only; Member: International
Member</maxx:user.PrivilegeGroupStr>
</cas:authenticationSuccess>
</cas:serviceResponse>
```

Response for a nonmember individual who is on the staff of the association

```
<cas:serviceResponse xmlns:cas="http://www.yale.edu/tp/cas"
xmlns:maxx="http://www.w3.org/XML/">
<cas:authenticationSuccess>
<cas:user>trainingmaxxstaff</cas:user>
<maxx:firstName>Test</maxx:firstName>
<maxx:lastName>Staff</maxx:lastName>
<maxx:CompanyName/>
<maxx:email/>
<maxx:id>6C6000009B9</maxx:id>
<maxx:individualNum>15796</maxx:individualNum>
<maxx:personalStatus>Active Staff</maxx:personalStatus>
<maxx:prefix/>
<maxx:middleInitial/>
<maxx:suffix/>
<maxx:aliases>Test</maxx:aliases>
<maxx:user.username>trainingmaxxstaff</maxx:user.username>
<maxx:JobFunctionStr/>
<maxx:MembershipStatusText>NonMember</maxx:MembershipStatusText>
<maxx:CompanyTypeStr/>
<maxx:user.PrivilegeGroupStr>Staff</maxx:user.PrivilegeGroupStr>
</cas:authenticationSuccess>
</cas:serviceResponse>
```

Embedded Login Forms

Sometimes clients want to include a mini-login form on their homepage or in their header. In order to implement this they need a way to get a valid loginTicket as a hidden field in that form. MaxxCAS implemented a method to make it easy to get such a loginTicket either server-side (before rendering the mini-login form) or client-side (via javaScript). The service should be provided as a field to this method, and a loginTicket will be returned:

- {baseURL}/forms/cas/loginTicket (Eg. https://www.factweb.org/forms/cas/loginTicket)

Oauth2 Protocol

As of 2018, MatrixMaxx also supports SSO via OAuth2. For integrations with third party systems, we support basic Authorization Code grant for general web applications, or Authorization Code with PKCE for mobile, single page, and native apps.

For further details about OAuth2 integration, please refer to the following guide: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

MatrixMaxx Devs may find detailed instructions about the implementation here: https://wiki.matrixgroup.net/index.php/MaxxOAuth2

Important Notes

- All requests to the Maxx OAuth2 endpoints should be https
- Any URLs that are going to be used as the redirect_uri field will need to be configured by a Maxx developer. Unlike CAS, OAuth does not allow us to whitelist the full domain. The redirect uri needs to match exactly.
- If using the basic Authorization Code grant, client secret should never be exposed in javascript/html or in a native application where the source code is exposed to the client. PKCE should be used instead for these types of integrations.

Authorization Code Grant (standard)

This is the default grant for use with a third-party system **where the client_secret can be secured**. This covers the majority of use-cases where the user's browser will be making requests to a protected server. The authorization code grant uses a multi-step process where the client first requests an authorization code. Upon receipt of the authorization token, the client should pass the code to the /token endpoint and receive an access token. When the access token is received, the client service knows that the authentication is successful.

The authorization code endpoint will pass the user to the MatrixMaxx login page if they are not already logged-in. After the user logs in, they will be redirected to the redirect_uri with the auth code appended to the URL as a querystring parameter.

Sample Authorization Code Request

```
https://{sitedomain}/forms/oauth/authorize
?client_id=(clientid)
&redirect_uri=(redirect_uri)
&response_type=code (should always be code for this grant type)
&state=(state can be any value, and you should check that you get the
same value back on the response)
```

Sample Authorization Code Response

```
https://(redirect_uri
address)/?code=64097588-6883-446e-b5bc-73af351ba22c&state=(state
should match value in request)
```

After the authorization code is received, a server side POST request should be made to the Token endpoint to receive the access token.

Sample Access Token Request (using curl)

```
curl -X POST https://{sitedomain}/forms/oauth/token
-d "code=64097588-6883-446e-b5bc-73af351ba22c (the access code we
received above)
&grant_type=authorization_code (should always be authorization_code
for this grant type)
&client_id=(client id)
&client_secret=(protected client secret)
&redirect_uri=(redirect_uri)
```

Authorization Code Grant (using PKCE)

This is the default grant for use with a third-party system **where the client_secret CANNOT be secured**. This should be used when implementing OAuth on a single page application where the token request is made from the browser, or in a native/mobile app where the source code can be decompiled. This type of grant is similar to the basic Authorization Code Grant, except we do not use a client secret in the Token request since it cannot be protected.

Instead, the client should encrypt a string into a hash using SHA256 encryption. The encrypted hash (known as the **code challenge**) will be passed in with the Authorization request, and the original string (known as the **code verifier**) will be passed into the Token request. The code verifier should be a string of random characters of length between 43 and 255. The oauth server will hash the code verifier and check that it matches the code challenge string that was received by the Authorization request. If this check fails, the response will include an error message indicating a bad client secret.

For more details on PKCE, you may refer to the RFC (https://tools.ietf.org/html/rfc7636)

Sample PKCE Authorization Code Request

```
https://{sitedomain}/forms/oauth/authorize
?client_id=(clientid)
&redirect_uri=(redirect_uri)
&response_type=code (should always be code for this grant type)
&state=(state can be any value, and you should check that you get the
same value back on the response)
&code_challenge=(code challenge is the code_verifier passed through
SHA256 encryption)
```

Sample Authorization Code Response

```
https://(redirect_uri
address)/?code=64097588-6883-446e-b5bc-73af351ba22c&state=(state
should match value in request)
```

After the authorization code is received, a client side POST request should be made to the Token endpoint to receive the access token. The **code verifier** string that was used to generated the code challenge should be passed here so that we can ensure that we are sending the token back to the

Sample Access Token Request (using curl)

```
curl -X POST https://{sitedomain}/forms/oauth/token
-d "code=64097588-6883-446e-b5bc-73af351ba22c (the access code we
received above)
&grant_type=authorization_code (should always be authorization_code
for this grant type)
&client_id=(client id)
&code_verifier=(code challenge should be a string of random
characters in length from 43-255)
&redirect_uri=(redirect_uri)
```

Token Response

The token endpoint will return some JSON data with a 200 HTTP code if everything is correct. Here is an example of what the JSON data will return by default. If additional data needs to be included in the response, it can be added with custom code.

```
{"access_token": "c9fba119-d2e1-4cc4-8930-0fa5d9ef1eb3",
"token_type": "Bearer", "username": "maki", "userid": "14D00000000",
"integrationid": "14D00000000"}
```

- **access_token** is a unique key to pass to the server to request access. In the future, these will be passed to an API to request protected resources.
- **token_type** is the type of token. This will always be "Bearer".

- **username** of the MatrixMaxx user
- **userid** is the unique ID for this user in the MatrixMaxx database
- **integrationid** will return the same as userid by default, but it can be mapped to a custom id value if necessary

Getting User Profile in OAuth2 flow

There are several options for getting a user's profile after completing the above steps to authenticate the user.

- **MatrixMaxx API** - After you receive the token response which validates that the user is successfully authenticated, you can pass the userid to the MatrixMaxx API to request user data. The API does not currently use the access token, but we can whitelist a server's IP to make these requests.
- **User Info endpoint** - The access token can be passed to an endpoint that will check if the token is valid and non-expired and will return user profile information if successful. This request can be made client side or server side and generally returns the same fields that are returned in the CAS response listed above. The data is returned in JSON format. `https://{sitedomain}/forms/oauth/userinfo?token={access_token}`

- **Open ID Connect JWT** This is not currently active for all MatrixMaxx implementations, but can be enabled as requested. For clients that have JWTs enabled, the token response will return an **id_token** field. The data will be an ID token in JWT format which can be decoded to get claims about the user's identity. For more information about OIDC, see the section below.

Open ID Connect

OpenID Connect (OIDC) is an identity layer built on top of the OAuth 2.0 framework. It allows third-party applications to verify the identity of the end-user and to obtain basic user profile information. OIDC uses JSON web tokens (JWTs). For more information about OIDC and JWTs see the following resources:

- https://openid.net/connect/
- https://jwt.io/introduction

MatrixMaxx can be configured to return an OIDC JWT as part of the token response. The JWT can be decoded to expose a number of basic user profile claims about the logged-in user. The token's signature can and should be validated against our server by using the public key that we expose. The URL for the public key will be:

```
https://{sitedomain}/.well-known/jwks.json
```

MatrixMaxx also exposes an endpoint which can be used for the OIDC Discovery process.

```
http://{sitedomain}/.well-known/openid-configuration
```

SECURITY CONSIDERATIONS

Communication Over HTTPS

All web sites involved in the SSO protocol should communicate and serve all content and web services exclusively over HTTPS. This helps prevent session hijacking, which is where an unauthorized user gains access to the system by using the identity of a user already authenticated to the system.

As of June 2018, this must be done using TLS 1.2 or higher protocol.

Session Inactivity Timeout

Additionally, Matrix Group recommends that login sessions expire after 2 hours of inactivity. This measure helps protect users who don't logoff the site, especially those where computers are shared with others, from having their login session used by unauthorized users.

ALTERNATIVES

We prefer CAS or OAuth2. However, there is more than one way to skin the proverbial cat.

Single Password via nightly export/import of user data

It is possible to achieve a single-password situation with another system via a nightly export of user data from the MatrixMaxx system, sent directly (usually via sFTP) to the remote system. This system is not as secure or foolproof, but will get the job done in most cases. There are numerous drawbacks to this approach, including …

- If a user changes their credentials in the AMS mid-day, then tries to go to the remote system, they will not be able to log in with their new password, which could lead to user confusion even if an abundance of help text is provided
- The association's data is now being held in two places rather than just one, making it twice as susceptible to intrusion or security compromises
- For hashed passwords, the third party vendor will have to use the same hashing algorithm when verifying passwords. We store hashed passwords as

$6$salt$encryptedPassword, with a 16-character salt. They should use a 'crypt' library function (http://man7.org/linux/man-pages/man3/crypt.3.html).

Single Login via a ‘Jump’ page

If the association wants all traffic to a remote system to funnel through a particular page on the main website, it is possible for MatrixMaxx to generate a special authenticated page that serves no other purpose than to prompt the user for a login and then immediately ‘jump’ them to the remote site, with relevant data in the URL string. This method has numerous drawbacks, including ...

- security concerns regarding the 'replay' capability of these posts
- inability for users to bookmark URLs on the restricted remote system and return to it successfully
- the remote system would need to be 'locked down' or provide some sort of redirect to send users back to the main site/page

Custom

Anything is possible with enough time and money. If another solution is desired by either the association or one of the other vendors, the MatrixMaxx AMS team will work with all parties to craft a solution that is agreeable to all. Please talk to your Matrix project manager to start the process.