Shibboleth IdP UI Deployment Instructions

Notes

The ShibUI is built as a Java Spring Boot (https://spring.io/projects/spring-boot) application. It can be run as a standalone web application that has Tomcat embedded in it. The same WAR file can be deployed into an external servlet container (standalone Tomcat etc). It can also be deployed using a Docker image. And in the Docker realm, the project also provides a full "testbed environment" that includes a database, an IdP, a LDAP server, etc.

The ShibUI currently supports three distinct roles:

- **ROLE_ADMIN** -- No limits, can do anything the Shibboleth UI supports. Currently, Metadata Provider configuration (and filter configuration) requires **ROLE_ADMIN** access
- **ROLE_USER** -- These users can only add individual metadata sources (single entityID SP metadata file), and modify metadata sources that they created. And when creating a new SP entry, that SP metadata will not be active until after a user with **ROLE_ADMIN** approves it.
- **ROLE_ENABLE** – These are "enhanced" ROLE_USER users that have the ability to **enable/publish** SP metadata but cannot access/configure Metadata Provider (and filter) configurations

Requirements

- The ShibUI requires a backing relational database for storage. Out of the box, you can run the application using an in memory to get familiar with the application, but you will need a permanent data store in order to retain configurations
- Java 11+ - the Docker version of the deployment includes the needed Java environment to run the application

Downloads

- WAR releases available at: https://github.internet2.edu/TIER/shib-idp-ui/releases
- Docker image available at: https://hub.docker.com/r/i2incommon/shib-idp-ui/tags (version 1.11.0 released 07/26/2022)

Configuration Notes

The following are some things that are useful to consider and know regardless of which deployment model you choose to go with.

Much of the behavior of the ShibUI can be set and controlled through properties files, which can be in one or both of the following formats:

- A Spring property file, which one would be used to using with the Shibboleth IdP – a simple text file with a property name, equals sign, and the property value, one per line. This file is named: *application.properties*
- a YAML format file, named *application.yml*

The ShibUI comes with a basic example of both, with the example *application.properties* file having the core settings for authentication, database connection information, users file, directory/location settings for where the ShibUI should write out the metadata files and metadata-providers.xml file it manages, etc. The example *application.yml* file contains all the settings that impact the information, options, list elements, etc. that are actually shown in the UI. There is no reason that you need to keep that distinction; you could manage everything through a single properties or YAML-format file if you wanted. On the other hand, it can be a convenient distinction to keep the core "internal/baked-in settings" distinct from the "front-end/UI" settings. Details on the properties that can/should be configured are detailed later in this document.

> You really do need to make at least a few changes to the "out-of-the-box" properties file (by creating either the .properties or .yml file) for the ShibUI to be useful at all - regardless of which deployment model you choose to go with.

Deployment Instructions

Deployment via Docker

Code and files for the TIER Shibboleth UI Project

This repository contains both:

- The needed artifacts to build (including auto-builds through Jenkins) a Docker image of the Shibboleth IdP UI,
- A "testbed" that provides the full environment one needs to explore and gain experience with the Shibboleth IdP UI.

The Docker image of the Shibboleth IdP UI follows the TIER Docker packaging standards, utilizing CentOS7, the Zulu JDK, supervisor, and the TIER Beacon configuration.

Deployment Steps

- Install Docker. These instructions require version 17.03.1 or higher.
- i2incommon has the latest image for Shib UI. If for some reason you need to build your own, you can do it with something like the following:

```
docker build --rm -t i2incommon/shib-idp-ui
```

- And then that image could be run with something like the following:

```
docker run -p 8080:8080 -v {yourlocalfile}:/opt/shibui/application.properties i2incommon/shib-idp-ui
```

Note: You'll almost certainly want to create a "local file" ('{yourlocalfile}' above) that contains the core application settings you want, overriding the defaults that in the Shibboleth IdP UI WAR file. Your file should be mounted at the location /opt/shibui/application.properties. The current set of supported properties is documented in the Internet2 Github Shib UI repository, but is also shown in this Readme.

Now that you have it running, you could access it at something like:

```
http://localhost:8080
```

If you did not set an explicit password in your local application.properties, you'll have to look at the startup "console messages" and find the one generated at startup, with a line that starts: **Using generated security password:**. The username is: **user**

Docker Testbed Environment

There are multiple "Testbed" environments that you can run, available in the repository (in the testbed folder of the repository project). The instances include various database setups as simple examples of how to quickly run the application configured for each database (Maria, Postgres, SQL Server, MySQL).

There is also a fully integrated example in the testbed folder in the **integration** folder. It includes the:

- Shibboleth UI
- a Shibboleth IdP
  - with a shared filesystem between the Shibboleth UI and Shibboleth IdP
- an LDAP server as the base credential/attribute store for the IdP
- a Postgres DB image for the UI's persistent database.

To setup the "Testbed", you will need to:

- Clone the repository:

```
$> git clone https://github.internet2.edu/TIER/shib-idp-ui.git
```

- **cd** into *shib-ui/testbed/integration*
- Run the following command:

```
docker-compose kill; docker-compose rm; docker-compose build && docker-compose up
```

Once Docker has completed the startup of all containers, you can access the ShibUI login screen with the following URL:

```
http://localhost:8080
```

- Default userid = root
- Default password = letmein7

Deployment via embedded Tomcat mode

This section describes how to run Shibboleth UI application as an executable WAR file with embedded Tomcat and external configuration sources which override default configuration setting embedded in Shibboleth UI war.

Shibboleth UI is a Spring Boot web application which supports all standard Spring Boot property sources and configuration options. So, let's assume that our external configuration directory is `/etc/shibboleth-ui`. By default, property sources named `application.properties` and `application.yml` will be recognized by Spring Boot and merged at runtime to form a finalized `Environment` object holding all the properties gathered from all the property sources locations and then available to configure Shibboleth UI web application. All the standard Spring Boot property sources precedence rules apply here, but for our purposes we need to know that Shibboleth UI war embeds the set of default configuration properties available on runtime classpath in `application. properties` file and then any standard property could be overridden by externalizing them in additional `application.properties` or `application.yml` files. So, back to our example, let's assume we use `/etc/shibboleth-ui/application.yml` file to run our Shibboleth UI application and connect to MariaDB RDBMS instead of a default embedded H2 database that is configured in `application.properties` embedded in shibui.war. And also let's assume that we have dowloaded the war file and placed it to `/usr/local/shibui/shibui-1.11.0.war`. The sample `/etc/shibboleth-ui/application.yml` containing properties to connect to MariaDB instance and default unencrypted password for the `root` user would look like this:

```
shibui:
  default-password: "{noop}pass"
spring:
 datasource:
   platform: mysql
   driver-class-name: org.mariadb.jdbc.Driver
   url: jdbc:mariadb://localhost:3306/shibui
   username: shibui
   password: shibui
 jpa:
   properties:
     hibernate:
       dialect: org.hibernate.dialect.MariaDB103Dialect
```

Note that you need to list an "encryption scheme" for the default-password, which is what the '{noop}' is preceding it. For more info on supported Spring Security's password storage formats, see: https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#pe-dpe-format

Then you would need to run the war and also tell Spring Boot where to find externalized **application.yml**. So inside **/usr/local/shibui** directory the command to run it would look like this:

java -Xmx1g -jar shibui-1.11.0.war --spring.config.additional-location=file:/etc/shibboleth-ui/
and then you could access application on http://localhost:8080 with root/pass username/password combination

So, now you could use externalized /etc/shibboleth-ui/application.yml file to override/configure any property available to Shibboleth UI web application independent of what is embedded in the shibui.war.

Deployment via external servlet container (Tomcat et al)

This section describes how to deploy Shibboleth UI application as web archive with external configuration sources which override default configuration setting embedded in Shibboleth UI war to external Tomcat servlet container.

Shibboleth UI is a Spring Boot web application which supports all standard Spring Boot property sources and configuration options. So, let's assume that our external configuration directory is `/etc/shibboleth-ui`. By default, property sources named `application.properties` and `application.yml` will be recognized by Spring Boot and merged at runtime to form a finalized `Environment` object holding all the properties gathered from all the property sources locations and then available to configure Shibboleth UI web application. All the standard Spring Boot property sources precedence rules apply here, but for our purposes we need to know that Shibboleth UI war deployed to external servlet container, embeds the set of default configuration properties on runtime classpath in `application.properties` file and then any standard property could be overridden by externalizing them in additional `application.properties` or `application.yml` files. So, back to our example, let's assume we use `/etc/shibboleth-ui/application.yml` file to run our Shibboleth UI application and connect to MariaDB RDBMS instead of a default embedded H2 database that is configured in `application.properties` embedded in shibui.war which would be deployed to external servlet container. The sample `/etc/shibboleth-ui/application.yml` containing properties to connect to MariaDB instance would look like this:

```
shibui:
  default-password: "{noop}pass"

spring:
  datasource:
    platform: mysql
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://localhost:3306/shibui
    username: shibui
    password: shibui
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MariaDBDialect
```

Then you would tell Spring Boot where to find externalized `application.yml`. That would be accomplished by passing `spring.config.additional-location` property. For Tomcat it could be accomplished in `$CATALINA_HOME/bin/setenv.sh` file like so:

`export JAVA_OPTS="$JAVA_OPTS -Dspring.config.additional-location=file:/etc/shibboleth-ui/"`

then deploy `shibui.war` to external Tomcat and then you could access application on `http://localhost:8080` with `root/pass` username/password combination

So, now you could use externalized `/etc/shibboleth-ui/application.yml` file to override/configure any property available to Shibboleth UI web aplication independent of what is embedded in shibui.war deployed to external Tomcat container.

Configure HTTPS

To deploy under HTTPS, if the external Tomcat is used, the standard configuration of Tomcat HTTP connector applies here. When deploying in the embedded Tomcat mode, in order to enable HTTPS, the following configuration properties (sample) should be used:

```
server:
  ssl:
    key-store: /etc/shibui/keystore.p12
    key-store-password: password
    key-store-type: pkcs12
    key-alias: tomcat
    key-password: password
  port: 8443

Note that `keystore.p12` would contain a valid SSL certificate
```

Authentication Options

One key decision that you will need to make is how to control authentication of users of the ShibUI. If you use the default user or a users file, note the following on defining passwords

Currently, the supported values for ENCRYPT_SCHEME are either:

- noop -- clear text password follows
- bcrypt -- the following value has been encrypted with the $2a$ Bcrypt algorithm (limitation of the underlying Spring library currently incorporated is that only the $2a$ Bcrypt algorithm is supported.)

Default - Single "root" user

Simple, recommended only for secure environments (private networks available accessible via VPN or local access) with very limited user base (1-2 admin)

If you "do nothing", the ShibUI will set up a single "**root**" (that's the username) user, with a password it will generate the first time you run it. That password should be displayed in the console log as the UI starts up – but you probably really don't want to rely on that. If you are going to go with this option, you should set the password for that single "root" user in the configuration file.

application.properties: **shibui.default-password** = {ENCRYPT_SCHEME}password

Users Defined via File

Simple, recommended only for secure environments (private networks available accessible via VPN or local access) with limited user base (where users will not share a single login)

Obviously, you could start playing with the ShibUI, and even have multiple people use that same single root account. But likely just about any deployer is going to want to instead supply a "users file", and/or integrate your Shibboleth IdP as the authentication source.

Even if you are going to do the latter, you really need to "bootstrap" the Shibboleth IdP integration with at least a single user in that "users file". The reason for that is you need to establish at least one user who has the **ROLE_ADMIN** role.

The ShibUI application does support accepting the UI role for a user as an attribute from the Shibboleth IdP. If the IDP user information is not able to provide roles as listed above, you will need to configure a users file with at least one username (that will be passed from the IdP) listed with the ROLE_ADMIN role. You can configure the users file to have as many users as you want. The format of the users file is the following set of fields, separated by commas:

username,{ENCRYPT_SCHEME}password,firstname,lastname,ROLE_VALUE,email

Example Users File

```
dummy,{noop}password,first,last,ROLE_DUMMY,dummy@bill.com
admin,{noop}password,admin,admin,ROLE_ADMIN,admin@foo.com
user,{noop}password,some,user,ROLE_USER,user@foo.com
```

The property name that is used to indicate users file is:

- shibui.user-bootstrap-resource = file:/full/path/to/users/file.txt

**(the name of the file does not matter)

Every time you restart the ShibUI, it will read in that file, and update the internal user database entries with any changes. Password is still a required field, but won't be used if your Shibboleth IdP is being used to authenticate your ShibUI users.

The property name that is used to indicate users file is:

- shibui.user-bootstrap-resource = file:/full/path/to/users/file.txt

**(the name of the file does not matter)

Every time you restart the ShibUI, it will read in that file, and update the internal user database entries with any changes. Password is still a required field, but won't be used if your Shibboleth IdP is being used to authenticate your ShibUI users.

Users Authenticated via Shibboleth IDP

Recommended when a large volume of users needs to access the ShibUI or the application will be publicly accessible

Many deployers will presumably want to use their Shibboleth IdP for authentication. The ShibUI includes a Java-based SAML SP based on the Pac4j (https://www.pac4j.org) library (SAML support built on top of the Shibboleth Consortium's OpenSAML software). This is pretty easy to configure, but it requires doing the following steps, before you start up the ShibUI.

1. Configure, in a users file, at least one user that will match up with a Shibboleth IdP-supplied user identifier with ROLE_ADMIN (password is a required field in the file but is irrelevant/ignored in the user file)
2. Create a copy of the Shibboleth IdP's metadata, and place within the file system where the ShibUI can access
3. Configure the following settings in *application.properties or application.yml*.

Note: the settings for where Pac4j will store its SAML-related certificates (**shibui.pac4j.keystorePath** and related passwords) and SP metadata (**shibui.pac4j.serviceProviderMetadataPath**) will be the locations where you want Pac4j to store (it will self-generate files on first attempt to access. **These don't need to be existing files, it is easiest to let Pac4j do the generation for you.**)

application.properties config example

```
# Enable Pac4j, should generate its own certs and metadata on first attempt to use
shibui.pac4j-enabled = true
shibui.pac4j.keystorePath = /full/path/to/ShibUI/samlKeystore.jks
shibui.pac4j.keystorePassword = whatever_you_want
shibui.pac4j.privateKeyPassword = whatever_you_want
shibui.pac4j.serviceProviderMetadataPath = /full/path/to/ShibUI/sp-metadata.xml
shibui.pac4j.serviceProviderEntityId = http(s)://entityID/url/for/ShibUI

# Path to file containing Shibboleth IdP metadata
shibui.pac4j.identityProviderMetadataPath = /full/path/to/ShibIdP/idp-metadata.xml
shibui.pac4j.forceServiceProviderMetadataGeneration = false
shibui.pac4j.callbackUrl = https://localhost:8443/callback          Note this depends on the URL at which the
ShibUI will be available

# Following is the max allowed age of AuthnInstant allowed
# in SAML response sent to Pac4j SP
shibui.pac4j.maximumAuthenticationLifetime = 36000

# SAML attribute mapping. Name of the attribute the IdP will
# supply that the UI should use to populate its internal user store.
# As long as it least one Shibboleth IdP username matches up with at least one
# in the supplied users file that has the Admin (ROLE_ADMIN) roel, that person
# can manage the role assignment  of all other users thru the UI directly.shibui.pac4j.saml2ProfileMapping.
username = urn:oid:0.9.2342.19200300.100.1.1
shibui.pac4j.saml2ProfileMapping.firstname = urn:oid:2.5.4.42
shibui.pac4j.saml2ProfileMapping.lastname = urn:oid:2.5.4.4
shibui.pac4j.saml2ProfileMapping.email = urn:oid:0.9.2342.19200300.100.1.3
```

application.yml config example

```
shibui:
  pac4j-enabled: true # Enable Pac4j, should generate its own certs and metadata on first attempt to use
  pac4j:
    keystorePath: /full/path/to/ShibUI/samlKeystore.jks
    keystorePassword: whatever_you_want
    privateKeyPassword: whatever_you_want
    serviceProviderMetadataPath: /full/path/to/ShibUI/sp-metadata.xml
    serviceProviderEntityId: http(s)://entityID/url/for/ShibUI
    identityProviderMetadataPath: /full/path/to/ShibIdP/idp-metadata.xml
    forceServiceProviderMetadataGeneration: false
    callbackUrl: https://localhost:8443/callback
    maximumAuthenticationLifetime: 3600000
    saml2ProfileMapping:
      username: urn:oid:0.9.2342.19200300.100.1.1
      firstname: urn:oid:2.5.4.42
      lastname: urn:oid:2.5.4.4
      email: urn:oid:0.9.2342.19200300.100.1.3
```

Once you have those settings in place, then start up the ShibUI, and try to access the dashboard, you should be directed to the configured IdP for authentication. The IdP should present an error, because you have not yet configured it with metadata and attribute release for the ShibUI.

As the files you provided for SP keystore and metadata were "non-existent", Pac4j will generate those. **So now you will have a ShibUI SP metadata file (for ShibUI) that you can add to the IdP, and configure attribute release to the ShibUI entityID, matching up with the attribute mapping you configured above.**

Re-try to access the ShibUI dashboard again, and you should be "good to go".

The above process should work no matter what deployment model you choose. What will be different between the models is how the ShibUI interacts with the file system, and its expectations as to where various files will be found.

Configuration Properties

Database Configuration via application.yaml

This set of examples shows the basic configuration for each of the database types - please adjust server-names/addresses/ports/db-name/dbusers accordingly with your database. Defaults shown below

Support for MySQL, Postgres, MariaDB, and SQL Server are available

database configuration

```
spring:
  profiles:
    include:
  datasource:
    platform: mysql
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://db:3306/shibui
    username: shibui
    password: shibui
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MariaDB103Dialect

----------------------------------------------------------------

spring:
  profiles:
    include:
  datasource:
    platform: mysql
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://db:3306/shibui
    username: shibui
    password: shibui
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL8Dialect

----------------------------------------------------------------

spring:
  profiles:
    include:
  datasource:
    platform: postgres
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://db:5432/shibui
    username: shibui
    password: shibui
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQL95Dialect

----------------------------------------------------------------

spring:
  profiles:
    include:
  datasource:
    platform: sqlserver
    driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
    url: jdbc:sqlserver://db:1433
    username: shibui
    password: shibui
  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.SQLServerDialect
```

application.properties settings

This is a reflection of the default `application.properties` file included in the distribution. Note that lines beginning with # are commented out.

```
# Server Configuration
#server.port=8080

# Logging Configuration
#logging.config=classpath:log4j2.xml

#logging.level.org.springframework.security=INFO
logging.level.org.springframework=INFO
logging.level.edu.internet2.tier.shibboleth.admin.ui=INFO

spring.main.allow-bean-definition-overriding=true

# Database Credentials
spring.datasource.username=shibui
spring.datasource.password=shibui

# Database Configuration H2
spring.datasource.url=jdbc:h2:mem:shibui;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.platform=h2
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.h2.console.settings.web-allow-others=true

# spring.jackson.default-property-inclusion=non_absent
spring.jackson.default-property-inclusion=NON_NULL
spring.jackson.mapper.accept-case-insensitive-enums=true

# Database Configuration PostgreSQL
#spring.datasource.url=jdbc:postgresql://localhost:5432/shibui
#spring.datasource.driverClassName=org.postgresql.Driver
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

#Maria/MySQL DB
#spring.datasource.url=jdbc:mariadb://localhost:3306/shibui
#spring.datasource.driverClassName=org.mariadb.jdbc.Driver
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect

# Liquibase properties
spring.liquibase.enabled=false

# Hibernate properties
# for production never ever use create, create-drop. It's BEST to use validate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.
ImplicitNamingStrategyJpaCompliantImpl
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=false
spring.jpa.properties.hibernate.check_nullability=true
spring.jpa.hibernate.use-new-id-generator-mappings=true

#Envers versioning
spring.jpa.properties.org.hibernate.envers.store_data_at_delete=true

#Needed in the latest versions of Spring Boot when doing manual transaction management like we do in envers
versioning code
spring.jpa.properties.hibernate.current_session_context_class=org.springframework.orm.hibernate5.
SpringSessionContext

# Set the following property to periodically write out the generated metadata files. There is no default value;
the following is just an example
# shibui.metadata-dir=/opt/shibboleth-idp/metadata/generated
shibui.logout-url=/dashboard

# spring.profiles.active=default

## Default root user can be set in application.yml or here - setting in both places can be undeterministic
## Default password must be set for the default user to be configured and setup
```

```
#shibui.default-password={noop}somepassword
shibui.default-rootuser=root

shibui.metadata-sources-ui-schema-location=classpath:metadata-sources-ui-schema.json
shibui.entity-attributes-filters-ui-schema-location=classpath:entity-attributes-filters-ui-schema.json
shibui.nameid-filter-ui-schema-location=classpath:nameid-filter.schema.json

#Actuator endpoints (info)
# Un-comment to get full git details exposed like author, abbreviated SHA-1, commit message
#management.info.git.mode=full

###
# metadata-providers.xml write configuration

# Set the following property to periodically write out metadata providers configuration. There is no default
value; the following is just an example
# The run rate is defined in milliseconds. You will need to configure your Shibboleth IDP to read the produced
file
# shibui.metadataProviders.target=file:/opt/shibboleth-idp/conf/shibui-metadata-providers.xml
# shibui.metadataProviders.taskRunRate=30000

# Email configuration (local mailhog)
# spring.mail.host=mailhog
# spring.mail.port=1025
# spring.mail.username=username
# spring.mail.password=password
# spring.mail.properties.mail.smtp.auth=false
# spring.mail.properties.mail.smtp.starttls.enable=false

shibui.mail.text-email-template-path-prefix=/mail/text/
shibui.mail.html.email-template-path-prefix=/mail/html/
shibui.mail.system-email-address=doNotReply@shibui.org


#ShibUIConfiguration slurps in these values and they are bootstrapped in on startup
shibui.roles=ROLE_ADMIN,ROLE_ENABLE,ROLE_USER,ROLE_NONE
#Authenticated access roles - used by Spring Security to allow access when authenticated
shibui.roles.authenticated=ADMIN,ENABLE,USER

#In order to enable authentication via configured pac4j library (with external SAMl Idp, for example)
#This property must be set to true and pac4j properties configured. For sample pac4j properties, see
application.yml
#for an example pac4j configuration
#shibui.pac4j-enabled=true

#This property must be set to true in order to enable posting stats to beacon endpoint. Furthermore, appropriate
#environment variables must be set for beacon publisher to be used (the ones that are set when running shib-ui
in
#docker container
shibui.beacon-enabled=true
```

Additional Configuration via YAML Properties

The following properties may be customized through an `application.yml` file.

Attributes (for Attribute Release)

Example:

Attribute Release

```
custom:
  attributes:
    - name: eduPersonPrincipalName
      displayName: label.attribute-eduPersonPrincipalName
    - name: uid
      displayName: label.attribute-uid
```

name: The name of the entry. used to uniquely identify this entry.

displayName: This will normally be the label used when displaying this override in the UI. (set in messages.properties)

Defaults attributes

- **eduPersonPrincipalName**: label.attribute-eduPersonPrincipalName
- **uid**: label.attribute-uid
- **mail**: label.attribute-mail
- **surname**: label.attribute-surname
- **givenName**: label.attribute-givenName
- **eduPersonAffiliation**: label.attribute-eduPersonAffiliation
- **eduPersonScopedAffiliation**: label.attribute-eduPersonScopedAffiliation
- **eduPersonPrimaryAffiliation**: label.attribute-eduPersonPrimaryAffiliation
- **eduPersonEntitlement**: label.attribute-eduPersonEntitlement
- **eduPersonAssurance**: label.attribute-eduPersonAssurance
- **eduPersonUniqueId**: label.attribute-eduPersonUniqueId
- **employeeNumber**: label.attribute-employeeNumber

Relying Party Overrides

It is imperative when defining these that the "displayType" and "persistType" are known types. Typos or unsupported values here will result in that override being skipped! Supported types are as follows: boolean, integer, string, set, list. Note that "persistType" doesn't have to match "displayType". However, the only unmatching combination currently supported is a "displayType" of "boolean" and "persistType" of "string".

**Note: Relying Party Overrides can be managed through the ShibUI (as Custom Attributes) and need not be configured in the application

Example:
Relying Party Overrides

```
custom:
  overrides:
    - name: signAssertion
      displayName: label.sign-the-assertion
      displayType: boolean
      defaultValue: false
      helpText: tooltip.sign-assertion
      attributeName: http://shibboleth.net/ns/profiles/saml2/sso/browser/signAssertions
      attributeFriendlyName: signAssertions
    - name: dontSignResponse
      displayName: label.dont-sign-the-response
      displayType: boolean
      defaultValue: false
      helpText: tooltip.dont-sign-response
      attributeName: http://shibboleth.net/ns/profiles/saml2/sso/browser/signResponses
      attributeFriendlyName: signResponses
      invert: true
```

name: The name of the entry. used to uniquely identify this entry.

displayName: This will normally be the label used when displaying this override in the UI. (set in messages.properties)

displayType: The type to use when displaying this option

defaultValue(s): One or more values to be displayed as default options in the UI

helpText: This is the help-icon hover-over text

attributeName: This is the name of the attribute to be used in the xml. This is assumed to be a URI.

attributeFriendlyName: This is the friendly name associated with the above attributeName.

invert:

persistType: Optional. If it is necessary to persist something different than the override's display type, set that type here. For example, display a boolean, but persist a string.

persistValue: Required only when persistType is used. Defines the value to be persisted.

Default properties

- **signAssertion**
  displayName: label.sign-the-assertion
  displayType: boolean
  defaultValue: false
  helpText: tooltip.sign-assertion
  attributeName: http://shibboleth.net/ns/profiles/saml2/sso/browser/signAssertions
  attributeFriendlyName: signAssertions
- **dontSignResponse**
  displayName: label.dont-sign-the-response
  displayType: boolean
  defaultValue: false
  helpText: tooltip.dont-sign-response
  attributeName: http://shibboleth.net/ns/profiles/saml2/sso/browser/signResponses
  attributeFriendlyName: signResponses
  invert: true
- **turnOffEncryption**
  displayName: label.turn-off-encryption-of-response
  displayType: boolean
  defaultValue: false
  helpText: tooltip.turn-off-encryption
  attributeName: http://shibboleth.net/ns/profiles/encryptAssertions
  attributeFriendlyName: encryptAssertions
  invert: true
- **useSha**
  displayName: label.use-sha1-signing-algorithm
  displayType: boolean
  defaultValue: false
  helpText: tooltip.usa-sha-algorithm
  persistType: string
  persistValue: shibboleth.SecurityConfiguration.SHA1
  attributeName: http://shibboleth.net/ns/profiles/securityConfiguration
  attributeFriendlyName: securityConfiguration
- **ignoreAuthenticationMethod**
  displayName: label.ignore-any-sp-requested-authentication-method
  displayType: boolean
  defaultValue: false
  helpText: tooltip.ignore-auth-method
  persistType: string
  persistValue: 0x1
  attributeName: http://shibboleth.net/ns/profiles/disallowedFeatures
  attributeFriendlyName: disallowedFeatures
- **omitNotBefore**
  displayName: label.omit-not-before-condition
  displayType: boolean
  defaultValue: false
  helpText: tooltip.omit-not-before-condition
  attributeName: http://shibboleth.net/ns/profiles/includeConditionsNotBefore
  attributeFriendlyName: includeConditionsNotBefore
  invert: true
- **responderId**
  displayName: label.responder-id
  displayType: string
  defaultValue: null
  helpText: tooltip.responder-id
  attributeName: http://shibboleth.net/ns/profiles/responderId
  attributeFriendlyName: responderId
- **nameIdFormats**
  displayName: label.nameid-format-to-send
  displayType: set
  helpText: tooltip.nameid-format
  defaultValues:
  - urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified
  - urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
  - urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
  - urn:oasis:names:tc:SAML:2.0:nameid-format:transient
  attributeName: http://shibboleth.net/ns/profiles/nameIDFormatPrecedence
  attributeFriendlyName: nameIDFormatPrecedence
- **authenticationMethods**
  displayName: label.authentication-methods-to-use
  displayType: set
  helpText: tooltip.authentication-methods-to-use
  defaultValues:
  - https://refeds.org/profile/mfa
  - urn:oasis:names:tc:SAML:2.0:ac:classes:TimeSyncToken
  - urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
  attributeName: http://shibboleth.net/ns/profiles/defaultAuthenticationMethods
  attributeFriendlyName: defaultAuthenticationMethods
- **forceAuthn**
  displayName: label.force-authn

displayType: boolean
defaultValue: false
helpText: tooltip.force-authn
attributeName: http://shibboleth.net/ns/profiles/forceAuthn
attributeFriendlyName: forceAuthn

Steps to integrate a SAML2 based IdP with ShibUI

Note: be sure to read through the Authentication options above, they provide some details that apply regardless of the deployment option you have chosen.

Using Testbed Environment:

We will be using the Docker Testbed environment for this working demo. The Testbed is included in the source project. Please make sure you have already deployed the Testbed environment and
verified it is working:

- Verify can log into ShibUI via http://localhost:8080
- Verify Shibboleth is running by accessing metadata page for Shibboleth instance https://localhost:443/idp/shibboleth
  We will be needing this metadata info later, so feel free to save to a file now.

  ```
  curl -k https://localhost:443/idp/shibboleth --output <ShibUI Root>/test-compose/shibui/conf/idp_metadata.
  xml
  ```

You will need to stop the Docker containers before continuing in the next section:

```
docker-compose down
```

ShibUI Setup

We will be using the Pac4j's library for SAML2 support in ShibUI. This will be easy to implement since ShibUI uses a pluggable architecture.

To enable Pac4j, open and update the following files:

- *test-compose/shibui/conf/applications.properties:*

  ```
  shibui.pac4j-enabled=true
  ```

- *test-compose/shibui/conf/applications.yml:*
  * Under ShibUI section add:

  ```
    pac4j-enabled: true
    pac4j:
      keystorePath: "/etc/opt/samlKeystore.jks"
      keystorePassword: "changeit"
      privateKeyPassword: "changeit"
      serviceProviderEntityId: "https://idp.example.com/shibui"
      serviceProviderMetadataPath: "/etc/opt/sp-metadata.xml"
      identityProviderMetadataPath: "/etc/opt/idp-metadata.xml"
      forceServiceProviderMetadataGeneration: false
      callbackUrl: "https://localhost:8443/callback"
      maximumAuthenticationLifetime: 3600000
      saml2ProfileMapping:
        username: urn:oid:0.9.2342.19200300.100.1.1
        firstname: urn:oid:2.5.4.42
        lastname: urn:oid:2.5.4.4
        email: urn:oid:0.9.2342.19200300.100.1.3
  ```

    - Key fields:
        - **keystorePath**: URL to an existing or newly created keystore. Create or move keystore to *test-compose/shibui/conf/.*
          Create command:
          keytool -genkeypair -alias pac4j -keypass changeit -keystore samlKeystore.jks -storepass changeit -keyalg RSA -keysize 2048 -validity 3650
        - **serviceProviderEntityId**: Entity ID of ShibUI
        - **serviceProviderMetadataPath**: Location of where you want SP metadata file to be created
        - **identityProviderMetadataPath**: Location of saved Shibb IdP metadata file (saved earlier when verifying Testbed environment)
        - **saml2ProfileMapping**: Attributes needed by ShibUI to work with SAML2 IdP

Make sure the keystore file, idp metadata file, and both application files are moved to the ShibUI container when started:

- test-compose/docker-compose.yml:
  under shibui: volumes:

```
        - ./shibui/conf/application.yml:/opt/shibui/application.yml
        - ./shibui/conf/samlKeystore.jks:/opt/shibui/samlKeystore.jks
        - ./shibui/conf/application.properties:/opt/shibui/application.properties
        - ./shibui/conf/idp-metadata.xml:/opt/shibui/idp-metadata.xml
```

Now run Docker:

```
docker-compose up
```

When the Docker containers are running, you will need to log into the ShibUI container and copy the newly created sp-metadata.xml file to a new *test-compose/idp/container-files/services/sp-metadata.xml* file.

Once this is complete, add the following to the *test-compose/idp/Dockerfile*:

```
COPY container-files/services/sp-metadata.xml /opt/shibboleth-idp/metadata/sp-metadata.xml
```

Shibboleth IdP Setup

At this point you will need to add the ShibUI application as a new SP to Shibboleth IdP. The files you will need to update are located at *test-compose/idp /container-files/conf/.* It is assumed that you understand adding a new SP to your Shibboleth IDP

Suggested Setup Note

The following suggested setup will allow you to configure the generation of configuration from the ShibUI to be ingested for use in your Shibboleth IDP with the minimal amount of effort.

1. In your application properties, configure: **shibui.metadataProviders.target** - this should be the full path, including filename, of the XML output ShibUI will generate for metadata providers (the location must be writable by the ShibUi application and readable by the Shibboleth IdP , but not a temp directory that will get deleted/cleaned by the host system processes. eg - **file:/opt/shibboleth/config/dynamic_config/metadata-providers.xml**
2. In your application properties, configure: **shibui.metadata-dir** - this should be the full path of the directory to create metadata source files. Again, the location should be writeable by the ShibUI and readable by the IDP. eg - **/opt/shibboleth/config/dynamic_metadata**
3. In your Shibboleth configuration's services.xml file, update the block for shibboleth.MetadataResolverResources to include the shibui. metadataProviders.target location

example

```
<util:list id="shibboleth.MetadataResolverResources">
    <value>%{idp.home}/conf/metadata-providers.xml</value>
    <value>%{idp.home}/system/conf/metadata-providers-system.xml</value>
    <value>${idp.home}/conf/dynamic_config/metadata-providers.xml</value> <!-- match the shibui.
metadataProviders.target value -->
</util:list>
```

Shibboleth will require a restart to pick up the change.

Once ShibUI is up and running, login as an admin user and create a new Metadata Provider (type: **LocalDynamicMetadataResolver**). Use the shibui. metadata-dir location from step 2 above as the directory location.

The dynamic provider will provide Shibboleth with the location of any SP configured using the ShibUI.

*NOTE: you can use the testbed/integration setup in the project source code to test how this integration works and to see an example of the full end-to-end workings of the ShibUi and Shibboleth IDP

User Maintenance

TAP Beacon instrumentation

Shibboleth Idp UI software includes piece of instrumentation functionality which sends a small batch of statistical data about the environment in which application is deployed such as Docker image name, version, application name, etc. to a running "Beacon collector" facility which is exposed as a REST HTTP endpoint, as defined here: https://spaces.at.internet2.edu/display/TWGH/TIER+Instrumentation+-+The+TIER+Beacon In the specification page it is described to be implemented as a external cron job running inside Docker container, which is true for other TAP docker images instrumented with Beacon, but Shibboleth Idp UI application has this functionality implemented as an optional Java module. It is an opt-in type of functionality which is off by default but could be turned on with the following application property:

shibui.beacon-enabled=true

Once it is turned on it will assynchronoiusly send beacon data which it will gather from necessary environment variables (which will be set by TAP Docker image for shibboleth idp ui application), but if running outside of TAP Docker container and those environment variables are not set, even though the beacon module is enabled, the data will not be sent. Below is the example of necessary environment variables that need to be set in order for Beacon module to kick in if running outside of TAP Docker container:

LOGHOST="https://collector.testbed.tier.internet2.edu"
LOGPORT="5001"
IMAGE="shibui_local_no_image"
MAINTAINER="local_no_maintainer"
VERSION="1.11.0-SNAPSHOT"
TIERVERSION="191010"