

---

---

# APPENDIX A - IMPLEMENTATION GUIDE

---

---

This implementation guide is copied from the online implementation guide at <http://mise.mda.gov>. Refer to the web site for the latest version.

The Implementation Guide contains the following Sections:

1. [Introduction](#)
2. [Process Flows for Security, Publish/Update, Delete, Search, and Retrieve](#)
3. [Data Mapping](#)
4. [Code Overview](#)
5. [User Stories for Search](#)
6. [Interfacing with the Security Services](#)
7. [Interfacing with the Publication Service](#)
8. [Interfacing with the Delete Service](#)
9. [Interfacing with the Search Service](#)
10. [Interfacing with the Retrieve Service](#)
11. [Testing on the Test Service Platform](#)
12. [Going Live on the Integration Platform](#)

## 1. Introduction

Maritime security is a national priority that depends on the ability to efficiently, effectively, and appropriately share and safeguard information among trusted maritime partners within the Global Maritime Community of Interest (GMCOI). The Maritime Information Sharing Environment (MISE) as defined in the National Maritime Architecture Plan enables secure, standardized sharing of unclassified maritime information among a wide variety of federal, state and local agencies as well as international participants. MISE employs NIEM-M exchange models, representational state transfer (REST) services for publishing/consuming, and attribute-based access control to facilitate information sharing and safeguarding with non-provisioned users in a dynamic environment.

The purpose of this implementation guide is to provide practitioners with guidance and specific examples for interfacing with MISE. Specifically, it shows how to create messages that conform to the [National Information Exchange Model \(NIEM\)](#) Maritime IEPD formats, how to

implement security to successfully access the environment, and how to interface with the services to publish and consume messages from the environment.

For new practitioners it is recommended you start with [Process Flows for Security, Publish/Update, Delete, Search, and Retrieve](#) and proceed in order through the implementation guide.

## 1.1. NIEM-M EXCHANGE MODELS

To learn more about using the NIEM-M exchange models, where they can be downloaded, and how to produce messages to adhere these standards review the section on [Data Mapping](#).

## 1.2. SERVICE INTERFACES

To learn more about the service interfaces to share information via the MISE start with the section on [Process Flows for Security, Publish/Update, Delete, Search, and Retrieve](#), and then visit the sections on [Interfacing with the Publication Service](#), [Interfacing with the Delete Service](#), [Interfacing with the Search Service](#), and [Interfacing with the Retrieve Service](#).

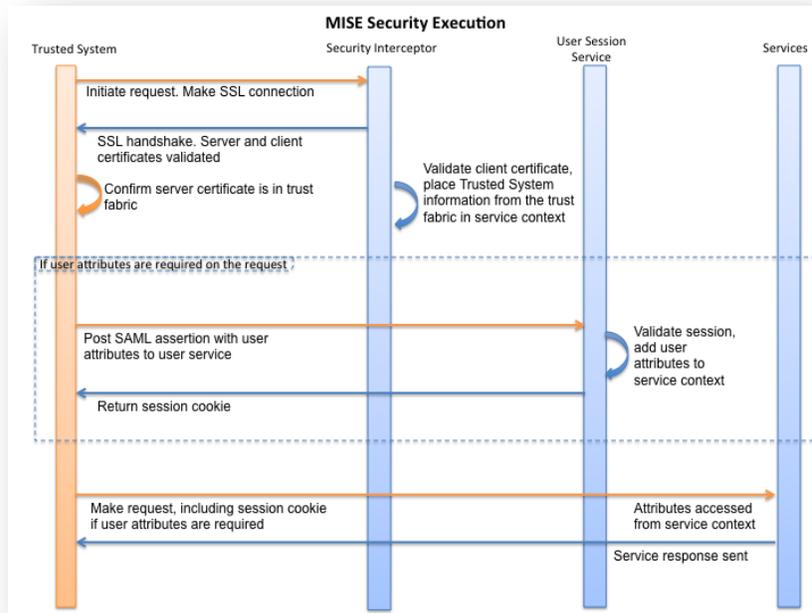
### 1.1. SECURITY SERVICES

To learn more about interfacing with the MISE security services see the sections on [Process Flows for Security, Publish/Update, Delete, Search, and Retrieve](#) and [Interfacing with the Security Services](#).

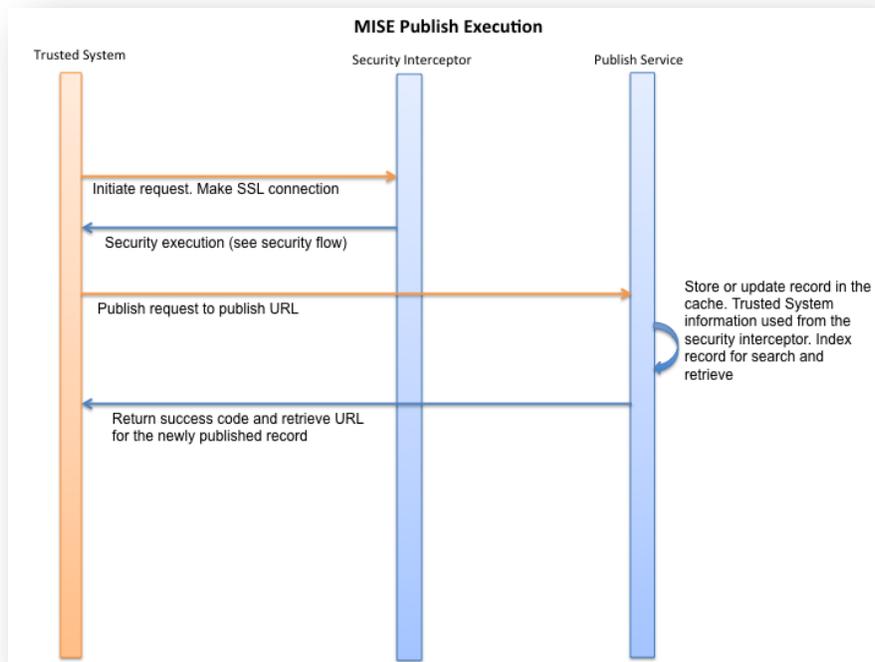
# 2. Process Flows for Security, Publish/Update, Delete, Search, and Retrieve

This section shows graphical representations of the major data provider and data consumer interactions with the MISE services.

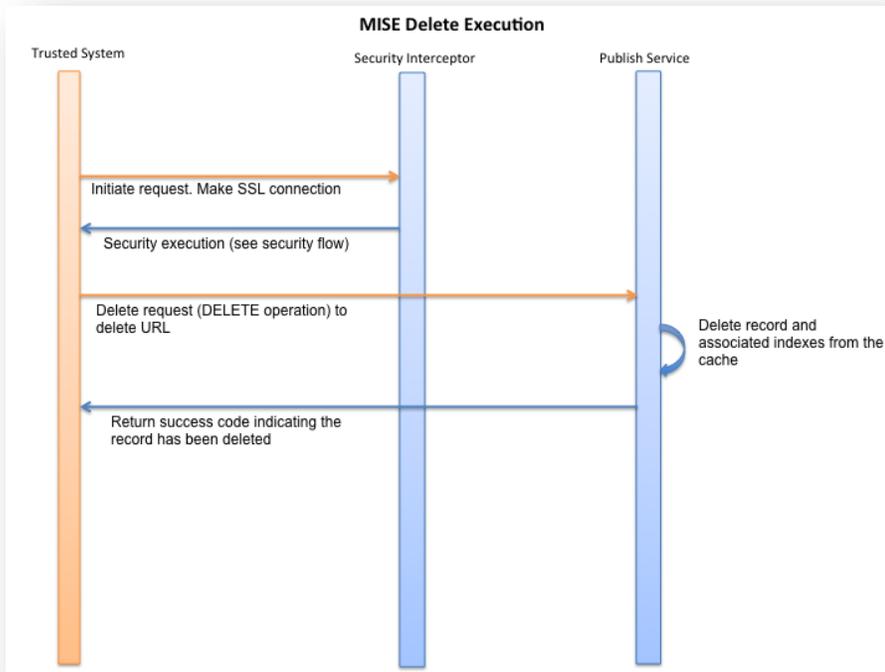
## 2.1. SECURITY



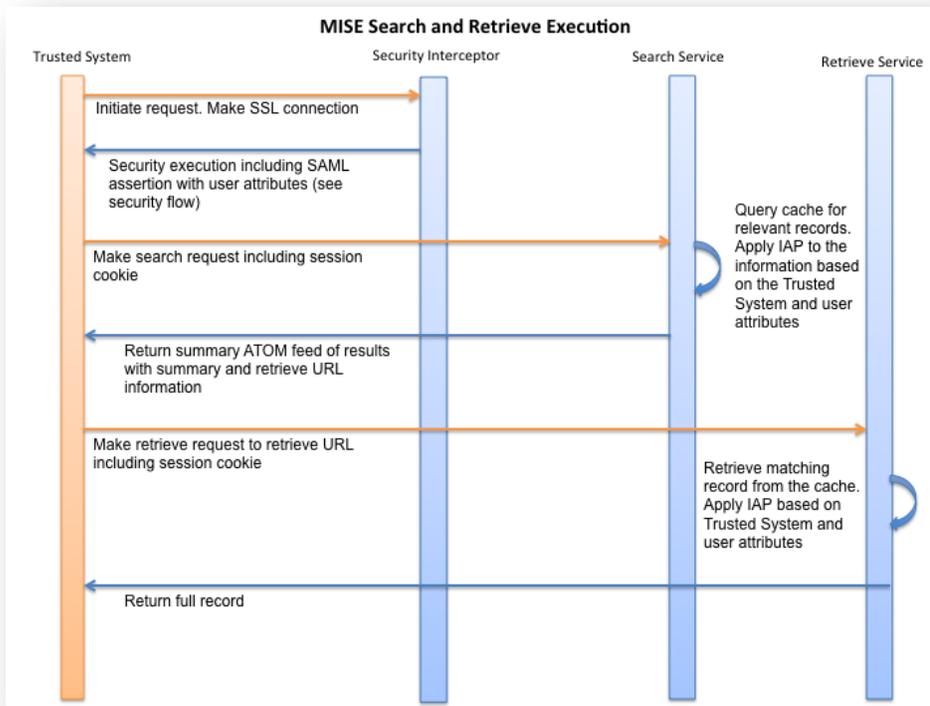
## 2.2. PUBLISH/UPDATE



## 2.3. DELETE



## 2.4. SEARCH AND RETRIEVE



## 3. Data Mapping

### 3.1. OBTAIN THE LATEST NIEM-M MODELS

The NIEM-M Maritime Domain Awareness (MDA) Enterprise Information Exchange Model (EIEM) and Information Exchange Package Documentation (IEPD) are registered and available for download from the [NIEM IEPD Clearinghouse](#) and the DoD Metadata Registry.

Quick links to download the artifacts:

- [Download MDA Enterprise Information Exchange Model \(EIEM\)](#)
- [Download Notices of Arrival IEPD](#)
- [Download Indicators and Notifications IEPD](#)
- [Download Vessel Positions IEPD](#)

### 3.2. HOW TO MAP DATA TO NIEM MARITIME

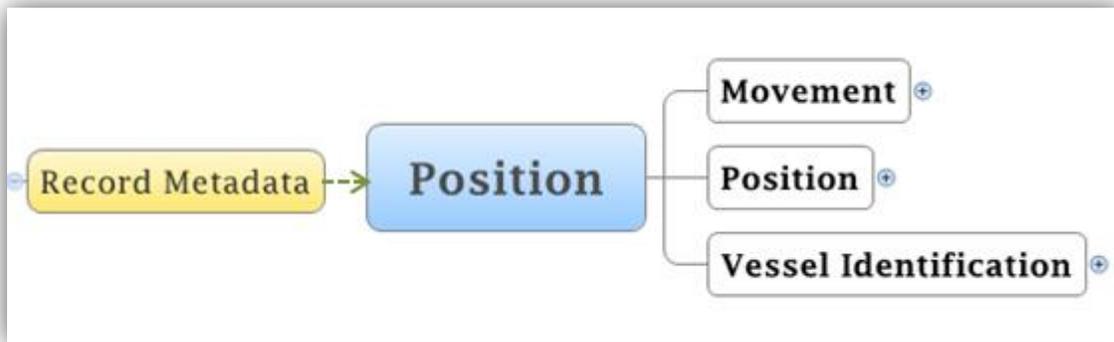
Suppose the following is a native vessel position report format. Data elements include ship's identification, GPS position, course, speed, navigational status, and time-stamp. Sample XML syntax is included below.

```

1  <Track>
2    <DateTime>2011-01-13T13:17:02.325Z</DateTime>
3    <CountryMID>303</CountryMID>
4    <RegionID>27</RegionID>
5    <TrackInfo>
6      <TrackInfoVessel>
7        <VesselId>
8          <MMSI>367354000</MMSI>
9        </VesselId>
10       <VesselAIS>
11         <VesselDataDynamic>
12           <Coordinate>
13             <LAT>53.8782833</LAT>
14             <LON>-166.538633</LON>
15           </Coordinate>
16           <COG>178</COG>
17           <SOG>0.1</SOG>
18           <HDT>228</HDT>
19           <ROT>0</ROT>
20           <NavStatus>Engine</NavStatus>
21           <PosAcc>Low</PosAcc>
22         </VesselDataDynamic>
23       </VesselAIS>
24     </TrackInfoVessel>
25   </TrackInfo>
26 </Track>

```

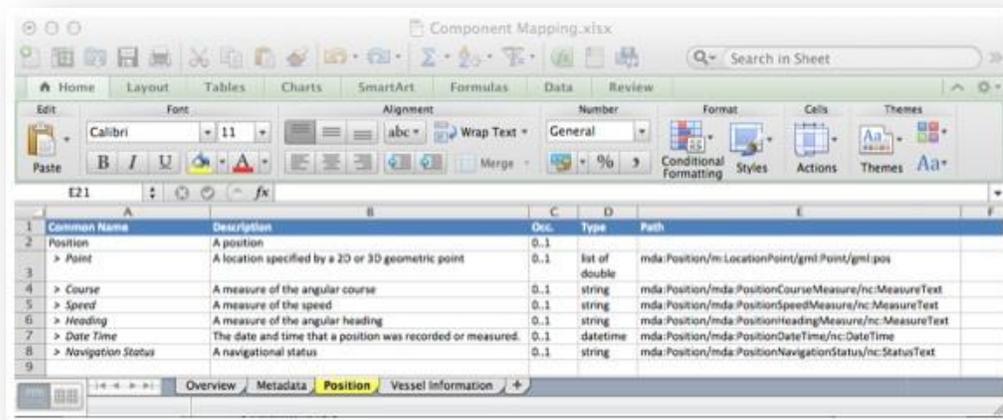
As depicted in the logical diagram for the Position IEPD, Movement, Position, and Vessel Identification are the three primary logical blocks included in a position report. Record Metadata is included in every message type.



The first step is to use the Mapping Spreadsheet from the IEPD. In this example, we are translating a track message into the NIEM format so we will be using the [Position IEPD](#). Find and open the component mapping spreadsheet.

The mapping spreadsheet provides the path for each element in the XML. Each tab corresponds to a "block" in the logical diagram.

An example of the Mapping Spreadsheet is captured below:



Use the Position tab of the Mapping spreadsheet to determine the xml elements to represent the position data in the position message type. For the example message the position is represented as follows:

```

1 <mda:Position>
2   <m:LocationPoint>
3     <gml:Point gml:id="tp1">
4       <gml:pos>53.8782833 -166.538633</gml:pos>
5     </gml:Point>
6   </m:LocationPoint>
7   <mda:PositionSpeedMeasure>
8     <nc:MeasureText>0.1</nc:MeasureText>
9     <nc:SpeedUnitCode>kt</nc:SpeedUnitCode>
10  </mda:PositionSpeedMeasure>
11  <mda:PositionCourseMeasure>
12    <nc:MeasureText>178</nc:MeasureText>

```

```

13         <m:AngleUnitText>deg</m:AngleUnitText>
14     </mda:PositionCourseMeasure>
15     <mda:PositionHeadingMeasure>
16         <nc:MeasureText>228</nc:MeasureText>
17         <m:AngleUnitText>deg</m:AngleUnitText>
18     </mda:PositionHeadingMeasure>
19     <mda:PositionNavigationStatus>
20         <nc:StatusText>Engine</nc:StatusText>
21     </mda:PositionNavigationStatus>
22     <mda:PositionDateTime>
23         <nc:DateTime>2011-01-13T13:17:02.325Z</nc:DateTime>
24     </mda:PositionDateTime>
25 </mda:Position>

```

If an element does not map to the NIEM format, it can be included in the expansion text if desired.

To complete the message, we used the Vessel Information tab of the Mapping Spreadsheet to complete translation. Vessel is represented as follows:

```

1 <mda:Vessel>
2     <m:VesselAugmentation>
3         <m:VesselMMSIText>367354000</m:VesselMMSIText>
4     </m:VesselAugmentation>
5 </mda:Vessel>

```

Below is the full NIEM conformant Position Message.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--Sample Position instance corresponding to the Position version 3.2 IEPD -->
3 <posex:Message
4     xsi:schemaLocation="<a
5 href="http://niem.gov/niem/domains/maritime/2.1/position/exchange/3.2">http://niem
6 .gov/niem/domains/maritime/2.1/position/exchange/3.2</a>
7 ../XMLSchemas/exchange/3.2/position-exchange.xsd"
8     xmlns:m="<a
9 href="http://niem.gov/niem/domains/maritime/2.1">http://niem.gov/niem/domains/mari
10 time/2.1</a>" xmlns:mda="<a
11 href="http://niem.gov/niem/domains/maritime/2.1/mda/3.2">http://niem.gov/niem/doma
12 ins/maritime/2.1/mda/3.2</a>"
13     xmlns:posex="<a
14 href="http://niem.gov/niem/domains/maritime/2.1/position/exchange/3.2">http://niem
15 .gov/niem/domains/maritime/2.1/position/exchange/3.2</a>"
16     xmlns:nc="<a href="http://niem.gov/niem/niem-
17 core/2.0">http://niem.gov/niem/niem-core/2.0</a>" xmlns:gml="<a
18 href="http://www.opengis.net/gml/3.2">http://www.opengis.net/gml/3.2</a>"
19     xmlns:ism="urn:us:gov:ic:ism" xmlns:xsi="<a
20 href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-
21 instance</a>"
22     mda:securityIndicatorText="LEI" mda:releasableNationsCode="USA"
23     mda:releasableIndicator="true">
24     <nc:DocumentCreationDate>
25         <nc:Date>2011-12-01</nc:Date>
26     </nc:DocumentCreationDate>
27     <nc:DocumentExpirationDate>
28         <nc:Date>2012-01-01</nc:Date>
29     </nc:DocumentExpirationDate>
30     <nc:DocumentCreator>
31         <nc:EntityOrganization>
32             <nc:OrganizationName>Example Organization</nc:OrganizationName>
33         </nc:EntityOrganization>
34     </nc:DocumentCreator>

```

```
22     <mda:RecordIDURI>00000001</mda:RecordIDURI>
23     <mda:MessageStatusCode>Initial</mda:MessageStatusCode>
24     <mda:MessageSourceSystemName>Track Source</mda:MessageSourceSystemName>
25     <mda:ICISMMarkings ism:classification="U"
26         ism:ownerProducer="USA" />
27     <mda:Vessel>
28         <m:VesselAugmentation>
29             <m:VesselMMSIText>367354000</m:VesselMMSIText>
30         </m:VesselAugmentation>
31     </mda:Vessel>
32     <mda:Position>
33         <m:LocationPoint>
34             <gml:Point gml:id="tp1">
35                 <gml:pos>53.8782833 -166.538633</gml:pos>
36             </gml:Point>
37         </m:LocationPoint>
38         <mda:PositionSpeedMeasure>
39             <nc:MeasureText>0.1</nc:MeasureText>
40             <nc:SpeedUnitCode>kt</nc:SpeedUnitCode>
41         </mda:PositionSpeedMeasure>
42         <mda:PositionCourseMeasure>
43             <nc:MeasureText>178</nc:MeasureText>
44             <m:AngleUnitText>deg</m:AngleUnitText>
45         </mda:PositionCourseMeasure>
46         <mda:PositionHeadingMeasure>
47             <nc:MeasureText>228</nc:MeasureText>
48             <m:AngleUnitText>deg</m:AngleUnitText>
49         </mda:PositionHeadingMeasure>
50         <mda:PositionNavigationStatus>
51             <nc:StatusText>Engine</nc:StatusText>
52         </mda:PositionNavigationStatus>
53         <mda:PositionDateTime>
54             <nc:DateTime>2011-01-13T13:17:02.325Z</nc:DateTime>
55         </mda:PositionDateTime>
56     </mda:Position>
57 </posex:Message>
```

## 4. Code Overview

The MISE implementation team provides a client toolkit for interface to the MISE REST services for Publish, Update, Delete, Search, and Retrieve. All operations are simply REST operations to the correct endpoint.

The client toolkit is primarily designed to make interfacing with the security services easier. The client toolkit is implemented in accordance with the following specifications. More information about each of the specifications can be found in the links below.

- [National MDA Architecture Attribute Specification](#)
- [National MDA Architecture Security Specification](#)
- [National MDA Architecture Publish Specification](#)
- [National MDA Architecture Search/Retrieve Specification](#)

The client toolkit can be downloaded from the MDA Architecture [tools page](#). The tools contain a simple java project that demonstrates how to connect and make a GET request against the ISI services. Two JAR files are included. The first is the MDAUtils JAR, which contains the MDA

Architecture security implementation, the client REST toolkit, and all the necessary dependencies. Additionally, the project also requires the included commons-io JAR, which provides file handling utilities for reading and writing files.

The client toolkit is used in all of the following examples:

- [Interfacing with the Security Services](#)
- [Interfacing with the Publication Service](#)
- [Interfacing with the Delete Service](#)
- [Interfacing with the Search Service](#)
- [Interfacing with the Retrieve Service](#)

Please note that the current base URL for the MISE is /services/MDAService/, followed by publish, search, or retrieve, as described in this guide.

## 5. User Stories for Search

Prior to reading this section, read [Process Flows for Security, Update, Delete, Search, and Retrieve](#) for a basic understanding of how information flows between the provider and consumer.

This set of user stories calls out specific examples for search and retrieve for the information products provided by the MISE. Examples for publish and delete are contained in the sections that discuss those operations, as they are simple, one-time operations.

As noted in the [National MDA Search/Retrieve Specification](#), these operations return the Atom summary feeds of each of the information products. The full message for any of the summaries would be accessed via a retrieve operation.

In each of the examples below, the URL is relative to the mise.mda.gov base path for MISE service access. The placeholder \$value is used in the place of query values.

### 5.1. POSITION

Return vessel position summary messages based on a geospatial area and time window, to see last known positions of all vessels within that geospatial area.	/search/pos?ulat=\$value&ulng=\$value&llat=\$value&llng=\$value&start=\$value&end=\$value
Retrieve a full vessel position message from the URL in a position summary for full details on a specific vessel.	/retrieve/pos?entityid=\$eid&recordid=\$posid
Retrieve the most recent vessel position summary data for each vessel that has updated in the last 30 seconds in the geographic area of interest.	/search/pos?ulat=\$value&ulng=\$value&llat=\$value&llng=\$value&start=\$value&end=\$value start, end should be the last 30 seconds

### 5.2. INDICATORS AND NOTIFICATIONS

Retrieve a summary of IANs about all vessels in a specific	/search/ian?ulat=\$value&ulng=\$value&llat=\$value&llng=
--	--

geospatial area.	<code>\$value &amp;start=\$value&amp;end=\$value</code>
Retrieve a summary message for vessels with a specific threat level within a geospatial area. Note that \$threat in the following example must align with one of the threat values available in the IAN schema.	<code>/search/ian?ulat=\$value&amp;ulng=\$value&amp;llat=\$value&amp;llng=\$value&amp;\$threat=\$value</code>

### 5.3. NOTICE OF ARRIVAL

Retrieve a summary message of all vessels inbound to a specified port.	<code>/search/noa?PortCodeText=\$value&amp;start=\$value&amp;end=\$value</code>
Retrieve a summary message of all pending notices of arrival for a specified vessel.	<code>/search/noa?VesselNameText=\$value&amp;VesselMMSIText=\$value &amp;VesselIMONumberText=\$value</code>
Retrieve a summary of IANs of all vessels in a specific geospatial area that are carrying certain dangerous cargo (CDC).	<code>/search/ian?ulat=\$value&amp;ulng=\$value&amp;llat=\$value&amp;llng=\$value &amp;start=\$value&amp;end=\$value&amp;VesselCDCCargoOnboardIndicator=true</code>
Retrieve a summary message based on a specified vessel and port for which any data element has been updated in the last 10 minutes.	<code>/search/noa?start=\$value&amp;end=\$value&amp;PortCodeText=\$value&amp;VesselNameText=\$value &amp;VesselMMSIText=\$value&amp;VesselIMONumberText=\$value start, end should be the last 10 minutes</code>

\*Note that any combination of MMSI, IMO, and Name can be used, all three are not required.

### 5.4. LEVELS OF AWARENESS

Retrieve a summary message for all vessels in a specified geospatial area.	<code>/search/loa?ulat=\$value&amp;ulng=\$value&amp;llat=\$value&amp;llng=\$value&amp;start=\$value&amp;end=\$value</code>
Retrieve a summary message for vessels with a specific threat level within a geospatial area. Note that \$threat in the following example must align with one of the threat values available in the LOA schema.	<code>/search/loa?ulat=\$value&amp;ulng=\$value&amp;llat=\$value&amp;llng=\$value&amp;\$threat=\$value</code>
Retrieve a summary of a specific vessel in a geospatial area.	<code>/search/loa?ulat=\$value&amp;ulng=\$value&amp;llat=\$value&amp;llng=\$value &amp;VesselNameText=\$value&amp;VesselMMSIText=\$value&amp;VesselIMONumberText=\$value</code>

\*Note that any combination of MMSI, IMO, and Name can be used, all three are not required.

## 6. Interfacing with the Security Services

All interactions to publish and consume data within the MISE are secured interactions over SSL between trusted systems. As a prerequisite to understanding the security implementation examples in this section, it is highly recommend you first read the following documents:

- [National MDA Architecture Plan](#) for an overview of the MISE security approach.
- [National MDA Architecture Security Specification](#) for the details of how trusted systems securely connect to the ISI.
- [National MDA Architecture Attribute Specification](#) for an explanation of the common attributes used for entitlement management.

## 6.1. OBTAINING X.509 CERTIFICATES

Numerous tools and processes are available for creating key pairs and X.509 certificates. The exact process chosen by a trusted system will vary depending on the platform the trusted system implementation is based upon, agency procedures, and the chosen root CA.

In some cases a trusted system may need to generate a keypair and a certificate signing request (CSR) internally using a tool such as OpenSSL or Java's keytool, and submit the CSR to a root CA for signing. The following sections provide steps for generating the private key and public Certificate Signing Request (CSR).

## 6.2. USING OPENSSL TO GENERATE A PRIVATE KEY AND PUBLIC CERTIFICATE SIGNING REQUEST (CSR)

Issue the following command to create private key and CSR

```
openssl req -new -nodes -keyout myserver.key -out server.csr -newkey rsa:2048
```

This creates two files. The file myserver.key contains a private key; do not disclose this file to anyone. Carefully protect the private key. In particular, be sure to back up the private key, as there is no means to recover it should it be lost. The private key is used as input in the command to generate a Certificate Signing Request (CSR).

1. You will now be asked to enter details to be entered into your CSR. What you are about to enter is what is called a Distinguished Name or a DN. Use the FQDN as Common Name (CN). The fields email address, optional company name and challenge password can be left blank for your SSL certificate.

```
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:San Diego
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Agency One
Organizational Unit Name (eg, section) []:IT
Common Name (eg, YOUR name) []:agencyone.gov
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
-----
```

2. Your CSR has been created. You can open the server.csr in a text editor to view it.
3. Follow the CA-specific instructions for submitting the CSR to the CA to generate your SSL certificate.

## 6.3. USE JAVA'S KEYTOOL TO GENERATE A PRIVATE KEY AND PUBLIC CERTIFICATE SIGNING REQUEST (CSR)

1. Using the java keytool command line utility, the first thing you need to do is create a key store and generate the key pair. Do this with the following command:

```
keytool -genkey -keysize 2048 -keyalg RSA -alias agencyone -keystore
mykeystore
```

1. Tip: The 2048 in the command above is the key bit length. MISE requires a key bit length of 2048.
2. You will be prompted for a password for the key store. The key password must be at least 6 characters long.
3. Tip: Make a note of the password. If lost it cannot be retrieved.
4. You will be asked for several pieces of info which will be used by the MISE Test Certificate Authority to create your new SSL certificate. When it asks for your first and last name, make sure you enter the FQDN of your server that will make the connection to the MISE. Here is an example:

```
What is your first and last name?
[Unknown]: http://agencyone.mda.gov
What is the name of your organizational unit?
[Unknown]: IT
What is the name of your organization?
[Unknown]: Agency One
What is the name of your City or Locality?
[Unknown]: San Diego
What is the name of your State or Province?
[Unknown]: California
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=http://agencyone.mda.gov, OU=IT, O=Agency One, L=San Diego,
ST=California, C=US correct?
[no]: yes
```

5. You will be prompted for a password for the private key within the key store. If you press enter at the prompt, the key password is set to the same password as that used for the key store from the previous step. The key password must be at least 6 characters long.
6. Tip: Make a note of the passwords. If lost they cannot be retrieved.
7. Now generate the Certificate Signing Request (CSR) from the private key generated above using the following command:

```
keytool -certreq -alias agencyone -file agencyone.gov.csr -keystore mykeystore
This creates a CSR and stores it in a file named agencyone.gov.csr.
```

8. Follow the CA-specific instructions for submitting the CSR to the CA to generate your SSL certificate.

Below is an example of what your CSR will look like. This is an example only and cannot be used to generate your SSL certificate.

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIICtTCCAZOCAQAwcDELMAkGA1UEBhMCMVVMxEZARBgNVBAgTCmNhbGlmb3JuaWEeEjAQBgNVBAcT
CXNhbWVnbzETMBEGA1UEChMKYVdlbWlbnN5IG9uZTELMaKGA1UECzMCSVQxRjAUBgNVBAMTDWFn
ZW5jeW9uZS5nb3YwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCwjMqyx0R2Oqh1tjXJ
4tH7275YW01+6K6kLOc/yFmfDLK8oSynDoTq4PzO2z1BAHq/8CgkPTs/tHgNphWTlw0c5WpaR487
bVh0dyJwvz3EI6hLmPAYqTvAB2C2aW0zcLzGTSxR8rAhHoX7oOgA3E9xKmoYVMVIZLFFN63Tn/F6M
T5NdFdTbkoRzcxpkkVmH6o60Vv6jGTI+zUpdyC7W8QRm/kshQgtjXLeYLACXuLvaKzn69p1TLCss
knRCsOsLjHhJrBmPK3upD9HRZ2bbe+Pp1/QUUVztiHDhnwPyEV6Iq2ZF0AuIF4otQU05DLQMsI28
EPu52GmfDMkPuXZFmYYDAgMBAAGgADANBgkqhkiG9w0BAQUFAAOCAQEAJTWTApJiggCgSxE48+Wi
ATNHe3rHJYPLzFmTRupM0tReLQWA+246g+ZGFH0wRv2VO90mMW/MivoxAnoyyP5J708MNsHo1LmN
```

```
bW9kyUuZK22T0lpO3t6BDDix8NWLg5cEGm08sI20iptesemlKq4E/2TxFdEmLpiARD9768WGVtbX
8EB08V2U78A8s5hTMly7hnaywfOm4ezpW1lktU1EuzVxGLHkBj7H5CEKpjH02/AZRNyJRYWrzcdO
YS/gUqs/cvqL77QrwoXWjrCEjSKYtibaXNlSbjEnDKbkoKJl0UsKRLAhMs8NI/HvalV1o8J8/ftc
1J1xTgHFYyxRJluV6w==
-----END NEW CERTIFICATE REQUEST-----
```

## 6.4. REGISTRATION OF TRUSTED SYSTEM IN TRUST FABRIC

Once the necessary X.509 certificate is obtained, your trusted system must be registered in the trust fabric document by the MISE Management team. You will have an entry in the trust fabric for each role for which your system is authorized, i.e. data provider and/or data consumer.

Following is an example entry for a provider trusted system:

```
1 <md:EntityDescriptor entityID="<a
  href="https://mise.agencythree.gov/">https://mise.agencythree.gov/</a>">
2   <md:RoleDescriptor
  protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol"
3     xsi:type="mise:MISEProviderDescriptorType">
4     <md:KeyDescriptor use="signing">
5       <ds:KeyInfo xmlns:ds="<a
  href="http://www.w3.org/2000/09/xmldsig#">http://www.w3.org/2000/09/xmldsig#</a>">
6         <ds:X509Data>
7           <ds:X509Certificate>
8             <!-- Base 64 encoded certificate embedded here
9              This is the client certificate which the trusted
10             system will present during SSL connection handshake.
11             The private key matching this certificate will also
12             be used by this trusted system for signing SAML
13             assertions.
14             -->
15           </ds:X509Certificate>
16         </ds:X509Data>
17       </ds:KeyInfo>
18     </md:KeyDescriptor>
19   </md:RoleDescriptor>
20   <md:ContactPerson contactType="technical">
21     <md:Company>Trusted Federal Systems, Inc.</md:Company>
22     <md:GivenName>Eric</md:GivenName>
23     <md:SurName>Jakstadt</md:SurName>
24     <md:EmailAddress><a
  href="mailto:eric.jakstadt@trustedfederal.com">eric.jakstadt@trustedfederal.com</a
  ></md:EmailAddress>
25     <md:TelephoneNumber>404-806-8143</md:TelephoneNumber>
26   </md:ContactPerson>
27 </md:EntityDescriptor>
```

Now an example entry for a consumer trusted system. Notice in the consuming system entry in the trust fabric, the trusted system is assigned the appropriate indicator attributes used to make authorization decisions on queries.

```
1 <md:EntityDescriptor entityID="<a
  href="https://mise.agencyone.gov/">https://mise.agencyone.gov/</a>">
2   <md:Extensions>
3     <gfipm:EntityAttribute FriendlyName="COIIndicator"
4       Name="mise:1.2:entity:COIIndicator"
5       NameFormat="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
6       <gfipm:EntityAttributeValue
7         xsi:type="xs:string">True</gfipm:EntityAttributeValue>
8     </gfipm:EntityAttribute>
9     <gfipm:EntityAttribute FriendlyName="LawEnforcementIndicator"
```

```

8         Name="mise:1.2:entity:LawEnforcementIndicator"
NameFormat="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
9         <gfipm:EntityAttributeValue
xsi:type="xs:string">True</gfipm:EntityAttributeValue>
10        </gfipm:EntityAttribute>
11        <gfipm:EntityAttribute FriendlyName="PrivacyProtectedIndicator"
12        Name="mise:1.2:entity:PrivacyProtectedIndicator"
NameFormat="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
13        <gfipm:EntityAttributeValue
xsi:type="xs:string">True</gfipm:EntityAttributeValue>
14        </gfipm:EntityAttribute>
15        <gfipm:EntityAttribute FriendlyName="OwnerAgencyCountryCode"
16        Name="mise:1.2:entity:OwnerAgencyCountryCode"
NameFormat="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
17        <gfipm:EntityAttributeValue
xsi:type="xs:string">USA</gfipm:EntityAttributeValue>
18        </gfipm:EntityAttribute>
19    </md:Extensions>
20    <md:RoleDescriptor
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol"
21        xsi:type="mise:MISEConsumerDescriptorType">
22        <md:KeyDescriptor use="signing">
23        <ds:KeyInfo xmlns:ds="a
href="http://www.w3.org/2000/09/xmldsig#"></a>">
24            <ds:X509Data>
25                <ds:X509Certificate>
26                    <!-- Base 64 encoded certificate embedded here
27                    This is the client certificate which the trusted
28                    system will present during SSL connection
handshake.
29                    The private key matching this certificate will
also
30                    be used by this trusted system for signing SAML
31                    assertions.
32                    -->
33                </ds:X509Certificate>
34            </ds:X509Data>
35        </ds:KeyInfo>
36    </md:KeyDescriptor>
37 </md:RoleDescriptor>
38 <md:Organization>
39     <md:OrganizationName xml:lang="en">Agency One</md:OrganizationName>
40     <md:OrganizationDisplayName xml:lang="en">Agency
41     One</md:OrganizationDisplayName>
42     <md:OrganizationURL xml:lang="en"><a
href="http://www.agencyone.gov"</md:OrganizationURL"><a href="http://www.agencyone.gov"></md
:OrganizationURL</a>>
43     </md:Organization>
44     <md:ContactPerson contactType="technical">
45         <md:Company>Trusted Federal Systems, Inc.</md:Company>
46         <md:GivenName>Eric</md:GivenName>
47         <md:SurName>Jakstadt</md:SurName>
48         <md:EmailAddress><a
href="mailto:eric.jakstadt@trustedfederal.com">eric.jakstadt@trustedfederal.com</a
></md:EmailAddress>
49         <md:TelephoneNumber>404-806-8143</md:TelephoneNumber>
50     </md:ContactPerson>
51 </md:EntityDescriptor>

```

## 6.5. DOWNLOAD THE TRUST FABRIC DOCUMENT

As discussed in the [Security Specification](#), the trust fabric contains public keys for all trusted systems that interact with the MISE. The trust fabric endpoint requires HTTPS but does not require a client certificate or any other method of authentication.

Retrieve the trust fabric document by any standard means, including viewing in any browser, at the MISE server at `/miseresources/TrustFabric.xml`

The trust fabric document for the MISE test environment is available at:  
<https://107.23.66.168:9443/miseresources/TrustFabric.xml>

## 6.6. VALIDATE THE TRUST FABRIC SIGNATURE PROGRAMMATICALLY

The following sample code (written in Java) taken from [client toolkit](#) demonstrates validating of the signed trust fabric document. Since the trust fabric document is a SAML metadata file with a few simple extensions, this sample code is able to leverage the open source OpenSAML project to simplify implementation. Trusted system implementations not written in Java, or which already include other SAML implementations, may also be able to simplify implementation by relying on existing SAML metadata implementations.

The following code snippet shows how the trust fabric document may be loaded into a DOM object so the signing certificate can be parsed and the signature on the document validated.

```

1 // read in the trust fabric from a local file location
2   FileInputStream fis = new FileInputStream("/local/path/TrustFabric.xml");
3   m_domFactory = DocumentBuilderFactory.newInstance();
4   m_domFactory.setNamespaceAware(true);
5   Element domElement =
6     m_domFactory.newDocumentBuilder().parse(fis).getDocumentElement();
7 // cryptographic validation of signature
8 X509Certificate signedByCert = verifyXMLSignature(domElement);
9 System.out.println(String.format("Signature validation %s", signedByCert == null ?
10 "FAILED" : "SUCCEEDED"));

```

The following snippet takes the trust fabric as a DOM object and returns the signing certificate if it is valid.

```

1 public static X509Certificate verifyXMLSignature(Element target) throws Exception {
2   // Validate the signature -- i.e. SAML object is pristine:
3   NodeList nl = target.getElementsByTagNameNS(XMLSignature.XMLNS, "Signature");
4   if (nl.getLength() == 0)
5     return null;
6
7   KeyValueKeySelector kvs = new KeyValueKeySelector();
8   DOMValidateContext context = new DOMValidateContext(kvs, nl.item(0));
9
10  // Create a DOM XMLSignatureFactory that will be used to unmarshal the
11  // document containing the XMLSignature
12  String providerName = System.getProperty("jsr105Provider",
13  "org.jcp.xml.dsig.internal.dom.XMLDSigRI");
14  XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM", (Provider)
15  Class.forName(providerName).newInstance());
16
17  DOMXMLSignature signature = (DOMXMLSignature)
18  fac.unmarshalXMLSignature(context);
19  if (!signature.validate(context))

```

```

17         return null;
18
19         return kvs.getUsedCertificate();
20     }
    
```

## 6.7. IMPLEMENTING MISE SECURITY ATTRIBUTES

As detailed in the [National MDA Attribute Specification](#), entitlement management within the MISE relies on the use of a common set of entity, user, and data attributes to make run-time authorization decisions as to whether a trusted system and requesting user are authorized to access a requested information resource.

There are three categories for attributes defined for the National MDA Architecture:

1. Entity Attributes: Attributes that pertain to a trusted system within the MISE.
2. User Attributes: Attributes that pertain to a human user.
3. Data Attributes: Attributes that pertain to data.

Currently data is grouped by LE sensitive (LEI), privacy protected (PPI) and the rest of the community (COI). The security indicators defined in the attribute specification map to these groups, i.e. Law Enforcement Indicator, Privacy Protected Indicator, and COI Indicator. Additionally, there is a one-to-one relationship between the security indicators assigned to data (data attributes) by information providers to convey sharing restrictions and the indicators assigned to information consumer trusted systems (entity attributes) and users (user attributes) to convey their respective privileges.

## 6.8. APPLYING DATA ATTRIBUTES ON PUBLISH

As a Data Provider Trusted System you must tag messages before publishing to the ISI with metadata to convey any restrictions on the data.

Attribute Name	Possible Values	Description
SecurityIndicatorText	“LEI”   “PCII”   “PPI”   “SBU”   “FSLT”   “PSO”   “COI”	Indicates the level of access required to access the data. LEI for Law Enforcement Sensitive Information, PPI for Privacy Protected Information, Protected Critical Infrastructure Information, Sensitive but Unclassified, Federal State Local and Tribal, Private Sector Only or COI for the rest of the community.
ReleasableIndicator	“true”   “false”	Marks data as releasable to the public domain under the restrictions of the associated security indicator.
ReleasableNationsCode	Space-delimited list of 3-letter country codes, ex. “CAN USA FRA”	Indicates data can only be released to those nations identified by the country codes. Default value is “USA”.

For example to publish a vessel position messages that is law enforcement sensitive, not publically releasable, and shareable with only US, these fields will be added to the exchange element of the publish message as depicted in the example below.

```

1 <message
  xmlns:posex="http://niem.gov/niem/domains/maritime/2.1/position/exchange/3.2"
  mda:securityindicatortext="LEI" mda:releasablenationscode="USA"
  mda:releasableindicator="false"></message>
    
```

## 6.9. SUPPLYING USER/ENTITY ATTRIBUTES FOR SEARCH/RETRIEVE

### 6.9.1. MAP LOCAL USER PRIVILEGES TO MISE SECURITY ATTRIBUTES

Use the MISE security attributes as defined in the [Attribute Specification](#) to assert citizenship and the access level for the user associated with a query. Citizenship is conveyed using the CitizenshipCode attribute, `mise:1.4:user:CitizenshipCode`, with a value equal to the ISO 3-letter country code.

The access level is conveyed using one or more attributes defined as indicators.

Indicator	Attribute Name	Description
Law Enforcement Indicator	<code>mise:1.4:user:LawEnforcementIndicator</code>	User requires and qualifies for access to law enforcement information in accordance with all appropriate statutes and legislation.
Privacy Protected Indicator	<code>mise:1.4:user:PrivacyProtectedIndicator</code>	User requires and qualifies for access to privacy protected information in accordance with all appropriate statutes and legislation.
Community of Interest Indicator	<code>mise:1.4:user:COIIndicator</code>	Minimum access level assigned to user that requires access to information shared by the MISE community.

### 6.9.2. FORMING SAML USER ASSERTION

The following code snippet provides an example of building the user assertions and adding them to the context of the request. The full example is shown in the section on [Interfacing with the Search Services](#).

```

1 //Form the user assertion
2     String assertingPartyID = "test.client";
3     AssertionBuilder builder = new AssertionBuilder(assertingPartyID);
4     builder.addStandardConditions(Constants.MISE_AUDIENCE_RESTRICTION,
5     10*60); // valid for 10 minutes
6     builder.addAttribute("ElectronicIdentityId",
7     "gfipm:2.0:user:ElectronicIdentityId", "<a
8     href='mailto:testuser@testsystem.gov'>testuser@testsystem.gov</a>");
9     builder.addAttribute("FullName", "gfipm:2.0:user:FullName", "Test T.
10    User");
11
12     Attribute attr = builder.addAttribute("CitizenshipCode",
13     "mise:1.4:user:CitizenshipCode", "USA");
14     builder.addAttribute("LawEnforcementIndicator",
15     "mise:1.4:user:LawEnforcementIndicator", "true");
16     builder.addAttribute("PrivacyProtectedIndicator",
17     "mise:1.4:user:PrivacyProtectedIndicator", "true");
18
19     builder.signUsingPkcs12(assertingPartyID,
20     FilenameUtils.separatorsToSystem(keystorePath), keystorePass);

```

```

13         Assertion assertion = builder.getAssertion();
14
15         //Important to not use SAMLUtils.asPrettyXMLString(object) as it will
cause the signature validation to fail
16         HttpResponse response = m_client.post("/MDAUserSessionService/login",
null, SAMLUtils.asXMLString(assertion),

```

## 7. Interfacing with the Publication Service

The publication service provides the interface to publish information products to the MISE. The complete interface description for publish is in the [Code Overview](#) page for the code download and library details.

This example walks through publishing a single Position instance to the position interface, assuming that the publishing system can already interface with the [security services](#), including registering the publishing system as a trusted system in the Trust Fabric.

The instance file that might be published via this operation can be downloaded [here](#). This file contains a NIEM-M Position instance containing three position reports for the MV Example.

The actual XML for a publish operation would normally be assembled from a database or some other storage location. The example below just reads the XML from a file.

```

1 package test;
2
3 import gov.mda.trustfabric.TrustFabric;
4 import gov.mda.util.RestServiceClient;
5
6 import java.io.File;
7 import java.io.FileInputStream;
8 import java.security.cert.CertificateFactory;
9 import java.security.cert.X509Certificate;
10
11 import javax.xml.parsers.DocumentBuilderFactory;
12 import javax.xml.xpath.XPathFactory;
13
14 import org.apache.commons.io.FileUtils;
15 import org.apache.commons.io.FilenameUtils;
16 import org.apache.http.HttpResponse;
17 import org.apache.http.entity.ContentType;
18
19 public class TestPublishClient {
20
21     /**
22      * @param args
23      */
24     public static void main(String[] args) {
25         RestServiceClient m_client;
26
27
28         /* Strongly recommend that these be loaded from a configuration file
dynamically in production code */
29
30         String miseCert = "C:\\Users\\user\\Documents\\ca\\ca.crt"; //public
certificate for the MISE
31         String trustFabricUrl = "<a
href="https://mise.mda.gov/miseresources/TrustFabric.xml">https://mise.mda.gov/mise
resources/TrustFabric.xml</a>"; //trust fabric URL on the MISE server
32         String trustFabricBackupPath =

```

```

"C:\\Users\\user\\Documents\\TrustFabricBackup.xml"; //backup local file location
for a cached version of the trust fabric
33     String serverScheme = "https";
34     String serverHost = "mise.mda.gov";
35     String serverPort = "9443";
36     String serverBasePath = "/services";
37     String keystorePath = "C:\\Users\\user\\Documents\\server.p12"; //keystore
which contains the certificate and private key for this trusted system
38     String keystorePass = "password";
39
40
41
42     try {
43         FileInputStream isCert = new
FileInputStream(FilenameUtils.separatorToSystem(miseCert));
44         CertificateFactory certFactory =
CertificateFactory.getInstance("X.509");
45         X509Certificate cert = (X509Certificate)
certFactory.generateCertificate(isCert);
46
47         TrustFabric.initializeFromURL(trustFabricUrl,
trustFabricBackupPath, cert);
48
49         m_client = new RestServiceClient(serverScheme, serverHost,
Integer.valueOf(serverPort), serverBasePath);
50
51         m_client.setClientCert(FilenameUtils.separatorToSystem(keystorePath),
keystorePass);
52
53         String body = null;
54         FileUtils.readFileToString(new File("SamplePosition.xml"), body);
55
56         HttpResponse response = m_client.put("/publish/pos/id", "", body,
ContentType.APPLICATION_XML); //perform the request
57     } catch(Exception e) {
58         e.printStackTrace();
59     }
60
61
62 }
63
64 }

```

Note again that all the configuration parameters should not be hardcoded as strings in production code, but should be loaded dynamically from a configuration file or configuration database.

Examining the code in detail, the following 4 lines load the SSL certificate for the MISE from the file system, and initialize the Trust Fabric. The Trust Fabric code attempts to load the MISE-hosted Trust Fabric from the MISE endpoint first, and can fall back to a local copy if the MISE endpoint cannot be accessed.

```

1  FileInputStream isCert = new
2  FileInputStream(FilenameUtils.separatorToSystem(miseCert));
3  CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
4  X509Certificate cert = (X509Certificate) certFactory.generateCertificate(isCert);
5
6  TrustFabric.initializeFromURL(trustFabricUrl, trustFabricBackupPath, cert);

```

The next section of the code creates the MISE Rest Client and initializes it with the client key store. The client key store must contain the private key corresponding to the certificate registered for the publishing system in the Trust Fabric.

```
1 m_client = new RestServiceClient(serverScheme, serverHost,
  Integer.valueOf(serverPort), serverBasePath);
2
3 m_client.setClientCert(FilenameUtils.separatorsToSystem(keystorePath),
  keystorePass);
```

Finally, the last few lines read the XML from a file and publish it to the MISE. The /id on the publish URL must actually be a unique ID for this Position message in the publishing system.

```
1 String body = null;
2 FileUtils.readFileToString(new File("SamplePosition.xml"), body);
3
4 HttpResponse response = m_client.put("/publish/pos/id", "", body,
  ContentType.APPLICATION_XML); //perform the request
```

In production, the publishing, response-handling, and error-handling code must be more robust, to deal with the various error codes that might be returned by the MISE depending on the interaction with the security services and the outcome of the publish operation.

Please note that the current base URL for the MSIE is /services/MDAService/, followed by publish, search, or retrieve, as described in this guide.

## 8. Interfacing with the Delete Service

The delete interface on the MISE is an extension of the publish interface. To delete a previously published information product, the publishing system need only issue a DELETE HTTP request with the same parameters as the original publish message. The complete interface description for delete is in the Publish Specification. See the Code Overview<sup>1</sup> page for the code download and library details. The ClientTest project containing these code examples is also available on that page.

The following code example is structurally identical to the publish example, save for the actual HTTP operation on line 47, which is a DELETE, instead of a PUT with XML content.

```
1 package test;
2
3 import gov.mda.trustfabric.TrustFabric;
4 import gov.mda.util.RestServiceClient;
5
6 import java.io.FileInputStream;
7 import java.security.cert.CertificateFactory;
8 import java.security.cert.X509Certificate;
9
10 import org.apache.commons.io.FilenameUtils;
11 import org.apache.http.HttpResponse;
12
13 public class TestDeleteClient {
```

---

<sup>1</sup> <https://mise.mda.gov/drupal/node/25>

```

14
15     /**
16      * @param args
17      */
18     public static void main(String[] args) {
19         RestClient m_client;
20
21
22         /* Strongly recommend that these be loaded from a configuration file
23         dynamically in production code */
24         String miseCert = "C:\\Users\\user\\Documents\\ca\\ca.crt"; //public
25         String trustFabricUrl = "<a
26         href="https://mise.mda.gov/miseresources/TrustFabric.xml">https://mise.mda.gov/mise
27         resources/TrustFabric.xml</a>"; //trust fabric URL on the MISE server
28         String trustFabricBackupPath =
29         "C:\\Users\\user\\Documents\\TrustFabricBackup.xml"; //backup local file location
30         for a cached version of the trust fabric
31         String serverScheme = "https";
32         String serverHost = "mise.mda.gov";
33         String serverPort = "9443";
34         String serverBasePath = "/services";
35         String keystorePath = "C:\\Users\\user\\Documents\\server.p12"; //keystore
36         which contains the certificate and private key for this trusted system
37         String keystorePass = "password";
38
39
40
41         try {
42             FileInputStream isCert = new
43             FileInputStream(FilenameUtils.separatorToSystem(miseCert));
44             CertificateFactory certFactory =
45             CertificateFactory.getInstance("X.509");
46             X509Certificate cert = (X509Certificate)
47             certFactory.generateCertificate(isCert);
48
49             TrustFabric.initializeFromURL(trustFabricUrl, trustFabricBackupPath,
50             cert);
51
52             m_client = new RestClient(serverScheme,
53             serverHost, Integer.valueOf(serverPort), serverBasePath);
54
55             m_client.setClientCert(FilenameUtils.separatorToSystem(keystorePath),
56             keystorePass);
57
58             HttpResponse response = m_client.delete("/publish/pos/id", "", null,
59             null);
60         } catch(Exception e) {
61             e.printStackTrace();
62         }
63     }
64 }

```

For the DELETE operation, the /id on the publish URL must actually be the ID under which the information product was originally published.

```
1 HttpResponse response = m_client.delete("/publish/pos/id", "", null, null);
```

In production, the delete, response-handling, and error-handling code must be more robust, to deal with the various error codes that might be returned by the MISE depending on the interaction with the security services and the outcome of the delete operation.

Please note that the current base URL for the MSIE is `/services/MDAService/`, followed by `publish`, `search`, or `retrieve`, as described in this guide.

## 9. Interfacing with the Search Service

The search service provides the interface to search for information products on the MISE. The complete interface description for search is in the Search/Retrieve Specification. See the [Code Overview](#) page for the code download and library details. The ClientTest project containing these code examples is also available on that page.

This example walks through a query for Position reports based on the first Position [User Story](#), a search bounded by geospatial area and time. This example assumes the searching system can interface with the MISE [security services](#), including registering the system as a trusted system in the Trust Fabric.

Unlike the previous two examples which only require the SSL handshake with the MISE, the search and retrieve operations require that the consumer system pass the entitlement attributes of the user. The meanings of the attributes are discussed in detail in the [Attribute Specification](#). These entitlement attributes are provided to the MISE via a SAML assertion. When the SAML assertion is validated, the MISE returns a session cookie to the client system, which must be provided for all subsequent search and retrieve operations.

```
1 package test;
2
3 import gov.mda.Constants;
4 import gov.mda.saml.AssertionBuilder;
5 import gov.mda.saml.SAMLUtils;
6 import gov.mda.trustfabric.TrustFabric;
7 import gov.mda.util.RestServiceClient;
8
9 import java.io.FileInputStream;
10 import java.security.cert.CertificateFactory;
11 import java.security.cert.X509Certificate;
12
13 import org.apache.commons.io.FilenameUtils;
14 import org.apache.http.HttpResponse;
15 import org.apache.http.entity.ContentType;
16 import org.apache.http.util.EntityUtils;
17 import org.opensaml.saml2.core.Assertion;
18 import org.opensaml.saml2.core.Attribute;
19
20 public class TestSearchClient {
21     /**
22      * @param args
23      */
24     public static void main(String[] args) {
25         RestServiceClient m_client;
26
27         /* Strongly recommend that these be loaded from a configuration file
```

```

29 dynamically in production code */
30
31     String miseCert = "C:\\Users\\user\\Documents\\ca\\ca.crt"; //public
32 certificate for the MISE
33     String trustFabricUrl = "<a
34 href=https://mise.mda.gov/miseresources/TrustFabric.xml>https://mise.mda.gov/mise
35 resources/TrustFabric.xml</a>"; //trust fabric URL on the MISE server
36     String trustFabricBackupPath =
37 "C:\\Users\\user\\Documents\\TrustFabricBackup.xml"; //backup local file location
for a cached version of the trust fabric
38     String serverScheme = "https";
39     String serverHost = "mise.mda.gov";
40     String serverPort = "9443";
41     String serverBasePath = "/services";
42     String keystorePath = "C:\\Users\\user\\Documents\\server.p12"; //keystore
which contains the certificate and private key for this trusted system
43     String keystorePass = "password";
44
45     try {
46         FileInputStream isCert = new
47 FileInputStream(FilenameUtils.separatorsToSystem(miseCert));
48         CertificateFactory certFactory =
49 CertificateFactory.getInstance("X.509");
50         X509Certificate cert = (X509Certificate)
51 certFactory.generateCertificate(isCert);
52
53         TrustFabric.initializeFromURL(trustFabricUrl,
54 trustFabricBackupPath, cert);
55
56         m_client = new RestServiceClient(serverScheme,
57 serverHost, Integer.valueOf(serverPort), serverBasePath);
58         m_client.setClientCert(FilenameUtils.separatorsToSystem(keystorePath),
59 keystorePass);
60
61         //Form the user assertion
62         String assertingPartyID = "test.client";
63         AssertionBuilder builder = new AssertionBuilder(assertingPartyID);
64         builder.addStandardConditions(Constants.MISE_AUDIENCE_RESTRICTION,
65 10*60); // valid for 10 minutes
66         builder.addAttribute("ElectronicIdentityId",
67 "gfipm:2.0:user:ElectronicIdentityId", "<a
68 href=mailto:testuser@testsystem.gov>testuser@testsystem.gov</a>");
69         builder.addAttribute("FullName", "gfipm:2.0:user:FullName", "Test T.
70 User");
71
72         Attribute attr = builder.addAttribute("CitizenshipCode",
73 "mise:1.4:user:CitizenshipCode", "USA");
74         builder.addAttribute("LawEnforcementIndicator",
75 "mise:1.4:user:LawEnforcementIndicator", "true");
76         builder.addAttribute("PrivacyProtectedIndicator",
77 "mise:1.4:user:PrivacyProtectedIndicator", "true");
78
79         builder.signUsingPkcs12(assertingPartyID,
80 FilenameUtils.separatorsToSystem(keystorePath), keystorePass);
81         Assertion assertion = builder.getAssertion();
82
83         //Important to not use SAMLUtils.asPrettyXMLString(object) as it will
84 cause the signature validation to fail
85         HttpResponse response = m_client.post("/MDAUserService/login",
86 null, SAMLUtils.asXMLString(assertion), ContentType.APPLICATION_XML);
87         EntityUtils.consumeQuietly(response.getEntity());
88         response = m_client.get("/MDAService/search/pos?ulat=3.75&ulng=-
89 2.0&llat=-2.75&llng=3.0&start=2012-06-10T12:10:00&end=2013-01-25T12:30:00", null);

```

```
70         //do something with the response
71     } catch(Exception e) {
72         e.printStackTrace();
73     }
74 }
75 }
```

Note again that all the configuration parameters should not be hardcoded as strings in production code, but should be loaded dynamically from a configuration file or configuration database.

Examining the code in detail, the following 4 lines load the SSL certificate for the MISE from the file system, and initialize the Trust Fabric. The Trust Fabric code attempts to load the MISE-hosted Trust Fabric from the MISE endpoint first, and can fall back to a local copy if the MISE endpoint cannot be accessed.

```
1 FileInputStream isCert = new
  FileInputStream(FilenameUtils.separatorsToSystem(miseCert));
2 CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
3 X509Certificate cert = (X509Certificate) certFactory.generateCertificate(isCert);
4
5 TrustFabric.initializeFromURL(trustFabricUrl, trustFabricBackupPath, cert);
```

The next section of the code creates the MISE Rest Client and initializes it with the client key store. The client key store must contain the private key corresponding to the certificate registered for the publishing system in the Trust Fabric.

```
1 m_client = new RestServiceClient(serverScheme, serverHost,
  Integer.valueOf(serverPort), serverBasePath);
2
3 m_client.setClientCert(FilenameUtils.separatorsToSystem(keystorePath),
  keystorePass);
```

The next section of the code deals with the creation of the SAML assertion with the user's attributes. The AssertionBuilder is a utility provided by the MDA toolkit to aid in creating the SAML. The required attributes are defined in the attribute specification. Every assertion must provide the ElectronicIdentityID, FullName, CitizenshipCode, and entitlement attributes. Note that the example below shows how to explicitly set the expiration time for the assertion. If not included, the sessions formed by each assertion are timed-out automatically.

```
1 String assertingPartyID = "test.client";
2 AssertionBuilder builder = new AssertionBuilder(assertingPartyID);
3 builder.addStandardConditions(Constants.MISE_AUDIENCE_RESTRICTION, 10*60); // valid
  for 10 minutes
4 builder.addAttribute("ElectronicIdentityId", "gfipm:2.0:user:ElectronicIdentityId",
  "<a href='mailto:testuser@testsystem.gov'>testuser@testsystem.gov</a>");
5 builder.addAttribute("FullName", "gfipm:2.0:user:FullName", "Test T. User");
```

The following code shows how to set the citizenship and entitlement information. These attributes are mapped from the consumer system's internal user database.

```
1 Attribute attr = builder.addAttribute("CitizenshipCode",
  "mise:1.4:user:CitizenshipCode", "USA");
```

```
2 builder.addAttribute("LawEnforcementIndicator",
  "mise:1.4:user:LawEnforcementIndicator", "true");
3 builder.addAttribute("PrivacyProtectedIndicator",
  "mise:1.4:user:PrivacyProtectedIndicator", "true");
```

As the final step in the process to create the SAML assertion, the assertion must be signed using the private key of the consumer system. This is the same private key/key store used to establish the SSL connection. Note that this signing operation explicitly includes the `assertingPartyID`, which should match the entity ID of the consumer system registered with the Trust Fabric.

```
1 builder.signUsingPkcs12(assertingPartyID,
  FilenameUtils.separatorsToSystem(keystorePath), keystorePass);
2 Assertion assertion = builder.getAssertion();
```

Once the assertion has been created, the HTTP request for the session and the search can be performed. Prior to the actual request, the consuming system must establish a session with the MISE with the entitlements for the requesting user. This code makes that request using the assertion that was just created. The `RestClient` internally stores the session cookie provided back by the MISE to use in future requests.

```
1 //Important to not use SAMLUtils.asPrettyXMLString(object) as it will cause the
  signature validation to fail
2 HttpResponse response = m_client.post("/MDAUserService/login", null,
  SAMLUtils.asXMLString(assertion), ContentType.APPLICATION_XML);
3 EntityUtils.consumeQuietly(response.getEntity());
```

Finally, once the assertion has been created and the session established, the search request can be made. The search request shown requests all `Position` instances in the specified bounding box, published in the specified time period. This will return an Atom feed containing summaries of all matching records in the MISE. The retrieve operation for each of the individual records in the Atom feed is discussed in the next section.

```
1 response = m_client.get("/MDAService/search/pos?ulat=3.75&ulng=-2.0&llat=-
  2.75&llng=3.0&start=2012-06-10T12:10:00&end=2013-012-25T12:30:00", null);
```

In production, the searching response handling, and error handling code must be more robust, to deal with the various error codes that might be returned by the MISE depending on the interaction with the security services and the outcome of the search operation.

Please note that the current base URL for the MISE is `/services/MDAService/`, followed by `publish`, `search`, or `retrieve`, as described in this guide.

## 10. Interfacing with the Retrieve Service

Once a search operation has been performed, the Retrieve interface on the MISE allows a consuming system to retrieve the complete NIEM-M XML instance of any record. Alternately, the retrieve URL for any record can be accessed directly if known, bypassing the search operation entirely.

The complete interface description for search is in the Search/Retrieve Specification. See the [Code Overview](#) page for the code download and library details. The ClientTest project containing these code examples is also available on that page.

Each record published to the MISE creates a unique retrieve URL for that record. As long as that record exists in the MISE cache, it can be accessed via that URL. The following example demonstrates the retrieve operation. As with the search, retrieve requires that the consumer system pass the entitlement attributes of the user. The meanings of the attributes are discussed in detail in the [Attribute Specification](#). These entitlement attributes are provided to the MISE via a SAML assertion. When the SAML assertion is validated, the MISE returns a session cookie to the client system, which must be provided for all subsequent retrieve operations.

```
1  package test;
2
3  import gov.mda.Constants;
4  import gov.mda.saml.AssertionBuilder;
5  import gov.mda.saml.SAMLUtils;
6  import gov.mda.trustfabric.TrustFabric;
7  import gov.mda.util.RestServiceClient;
8
9  import java.io.FileInputStream;
10 import java.security.cert.CertificateFactory;
11 import java.security.cert.X509Certificate;
12
13 import org.apache.commons.io.FilenameUtils;
14 import org.apache.http.HttpResponse;
15 import org.apache.http.entity.ContentType;
16 import org.opensaml.saml2.core.Assertion;
17 import org.opensaml.saml2.core.Attribute;
18
19 public class TestRetrieveClient {
20
21     /**
22      * @param args
23      */
24     public static void main(String[] args) {
25         RestServiceClient m_client;
26
27         /* Strongly recommend that these be loaded from a configuration file
28            dynamically in production code */
29
30         String miseCert = "C:\\Users\\user\\Documents\\ca\\ca.crt"; //public
31         certificate for the MISE
32         String trustFabricUrl = "<a
33 href=https://mise.mda.gov/miseresources/TrustFabric.xml>https://mise.mda.gov/mise
34 resources/TrustFabric.xml</a>"; //trust fabric URL on the MISE server
35
36         String trustFabricBackupPath =
37 "C:\\Users\\user\\Documents\\TrustFabricBackup.xml"; //backup local file location
38         for a cached version of the trust fabric
39         String serverScheme = "https";
40         String serverHost = "mise.mda.gov";
41         String serverPort = "9443";
42         String serverBasePath = "/services";
43         String keystorePath = "C:\\Users\\user\\Documents\\server.p12"; //keystore
44         which contains the certificate and private key for this trusted system
45         String keystorePass = "password";
46
47         try {
48             FileInputStream isCert = new
49             FileInputStream(FilenameUtils.separatorsToSystem(miseCert));
```

```

41         CertificateFactory certFactory =
CertificateFactory.getInstance("X.509");
42         X509Certificate cert = (X509Certificate)
certFactory.generateCertificate(isCert);
43
44         TrustFabric.initializeFromURL(trustFabricUrl, trustFabricBackupPath,
cert);
45
46         m_client = new RestServiceClient(serverScheme,
serverHost, Integer.valueOf(serverPort), serverBasePath);
47         m_client.setClientCert(FilenameUtils.separatorsToSystem(keystorePath),
keystorePass);
48
49         //Form the user assertion
50         String assertingPartyID = "test.client";
51         AssertionBuilder builder = new AssertionBuilder(assertingPartyID);
52         builder.addStandardConditions(Constants.MISE_AUDIENCE_RESTRICTION,
10*60); // valid for 10 minutes
53         builder.addAttribute("ElectronicIdentityId",
"gfipm:2.0:user:ElectronicIdentityId", "<a
href="mailto:testuser@testsystem.gov">testuser@testsystem.gov</a>");
54         builder.addAttribute("FullName", "gfipm:2.0:user:FullName", "Test T.
User");
55
56         Attribute attr = builder.addAttribute("CitizenshipCode",
"mise:1.4:user:CitizenshipCode", "USA");
57         builder.addAttribute("LawEnforcementIndicator",
"mise:1.4:user:LawEnforcementIndicator", "true");
58         builder.addAttribute("PrivacyProtectedIndicator",
"mise:1.4:user:PrivacyProtectedIndicator", "true");
59
60         builder.signUsingPkcs12(assertingPartyID,
FilenameUtils.separatorsToSystem(keystorePath), keystorePass);
61         Assertion assertion = builder.getAssertion();
62
63         //Important to not use SAMLUtils.asPrettyXMLString(object) as it will
cause the signature validation to fail
64         HttpResponse response = m_client.post("/MDAUserService/login",
null, SAMLUtils.asXMLString(assertion), ContentType.APPLICATION_XML);
65
66         String entityID = "https%3A%2F%2Fmise.agencyone.gov%2F";
67         String uuid = "79869883848892520";
68         response = m_client.get("/MDAService/retrieve/ian?entityid=" + entityID
+ "&recordid=" + uuid, null);
69
70         //do something with the response
71
72     } catch(Exception e) {
73         e.printStackTrace();
74     }
75 }
76 }

```

Note that all the configuration parameters should not be hardcoded as strings in production code, but should be loaded dynamically from a configuration file or configuration database.

Examining the code in detail, the following 4 lines load the SSL certificate for the MISE from the file system, and initialize the Trust Fabric. The Trust Fabric code attempts to load the MISE-hosted Trust Fabric from the MISE endpoint first, and can fall back to a local copy if the MISE endpoint cannot be accessed.

```
1 FileInputStream isCert = new
  FileInputStream(FilenameUtils.separatorsToSystem(miseCert));
2 CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
3 X509Certificate cert = (X509Certificate) certFactory.generateCertificate(isCert);
4
5 TrustFabric.initializeFromURL(trustFabricUrl, trustFabricBackupPath, cert);
```

The next section of the code creates the MISE Rest Client and initializes it with the client key store. The client key store must contain the private key corresponding to the certificate registered for the publishing system in the Trust Fabric.

```
1 m_client = new RestServiceClient(serverScheme, serverHost,
  Integer.valueOf(serverPort), serverBasePath);
2
3 m_client.setClientCert(FilenameUtils.separatorsToSystem(keystorePath),
  keystorePass);
```

The next section of the code deals with the creation of the SAML assertion with the user's attributes. The AssertionBuilder is a utility provided by the MDA toolkit to aid in creating the SAML. The required attributes are defined in the attribute specification. Every assertion must provide the ElectronicIdentityID, FullName, CitizenshipCode, and entitlement attributes. Note that the example below shows how to explicitly set the expiration time for the assertion. If not included, the sessions formed by each assertion are timed-out automatically.

```
1 String assertingPartyID = "test.client";
2 AssertionBuilder builder = new AssertionBuilder(assertingPartyID);
3 builder.addStandardConditions(Constants.MISE_AUDIENCE_RESTRICTION, 10*60); // valid
  for 10 minutes
  builder.addAttribute("ElectronicIdentityId", "gfipm:2.0:user:ElectronicIdentityId",
4 "<a href='mailto:testuser@testsystem.gov'>testuser@testsystem.gov</a>");
5 builder.addAttribute("FullName", "gfipm:2.0:user:FullName", "Test T. User");
```

The following code shows how to set the citizenship and entitlement information. These attributes are mapped from the consumer system's internal user database.

```
1 Attribute attr = builder.addAttribute("CitizenshipCode",
  "mise:1.4:user:CitizenshipCode", "USA");
2 builder.addAttribute("LawEnforcementIndicator",
  "mise:1.4:user:LawEnforcementIndicator", "true");
3 builder.addAttribute("PrivacyProtectedIndicator",
  "mise:1.4:user:PrivacyProtectedIndicator", "true");
```

As the final step in the process to create the SAML assertion, the assertion must be signed using the private key of the consumer system. This is the same private key/key store used to establish the SSL connection. Note that this signing operation explicitly includes the assertingPartyID, which should match the entity ID of the consumer system registered with the Trust Fabric.

```
1 builder.signUsingPkcs12(assertingPartyID,
  FilenameUtils.separatorsToSystem(keystorePath), keystorePass);
2 Assertion assertion = builder.getAssertion();
```

Once the assertion has been created, the HTTP request for the session and the retrieve can be performed. Prior to the actual request, the consuming system must establish a session with the MISE with the entitlements for the requesting user. This code makes that request using the

assertion that was just created. The RestClient internally stores the session cookie provided back by the MISE to use in future requests.

```
1 //Important to not use SAMLUtils.asPrettyXMLString(object) as it will cause the
  signature validation to fail
2 HttpResponseMessage response = m_client.post("/MDAUserService/login", null,
  SAMLUtils.asXMLString(assertion), ContentType.APPLICATION_XML);
```

Finally, once the assertion has been created and the session established, the retrieve request can be made. Each record has an ID that is unique to the publishing system, so the entity ID/record ID pair provides a unique key for the record. The entity ID in each case is the same as the entity ID in the trust fabric.

```
1 String entityID = "https%3A%2F%2Fmise.agencyone.gov%2F";
2 String uuid = "79869883848892520";
3 response = m_client.get("/MDAService/retrieve/ian?entityid=" + entityID +
  "&recordid=" + uuid, null);
```

In production, the response-handling and error-handling code must be more robust, to deal with the various error codes that might be returned by the MISE depending on the interaction with the security services and the outcome of the retrieve operation.

Please note that the current base URL for the MISE is /services/MDAService/, followed by publish, search, or retrieve, as described in this guide.

## 11. Testing on the Test Service Platform

Once the initial development work has been completed, the operation of the code can be tested against the MISE Test Platform. The following steps detail the process to test.

All information exchanged on the Test Platform must be TEST ONLY. No operational data may be exchanged on the Test Platform

1. Contact the MISE Engineering Team to obtain a test key store. The test key store provides the private key and certificate for one of the identities in the test Trust Fabric. This will allow connections with the security services on the MISE Test Platform.
2. Test the SSL handshake process with the Test Platform. Using the test key store, it should be possible to make the initial SSL connection, enabling further testing. Test the
3. Once the SSL connection has been established, test information can be published. Test, publish, and update for success and error conditions.
4. With the SSL connection, the delete operation can also be tested. Test delete for both success and error conditions.
5. If the search and retrieve services are required, the SAML assertions for user entitlement exchange should be tested. Ensure that the trusted system code can create and sign the necessary assertions to pass user entitlement information. Once a SAML assertion is correctly passed to the Test Platform, it will return a session cookie representing that user entitlement session, enabling access to the search and retrieve services. This test

process should also test that the trusted system code correctly separates sessions, handles errors, logs out, and handles SAML re-assertion when a session expires.

6. Once user entitlement sessions can be established, the search service can be tested. Test that issuing queries for previously published information is successful. Test for error conditions and handling empty responses. Finally, if a query will return too much information, the MISE will issue a 413 error code. Make sure this case is handled, typically by refining and re-issuing the query.
7. The final operation to test is retrieve. Using the results of a search, check that the full NIEM-M instance document can be retrieved. Again, make sure that error conditions are correctly handled
8. For a system that does both publish and search/retrieve, a final end-to-end test can be performed by publishing and then searching for and retrieving an information product or a set of products.

Testing a publishing system requires steps 1-4, since user entitlements and session handling is not required for publish and delete. Testing a search/retrieve consumer system requires steps 1,2, and 5-8. The MISE Engineering Team is available to help with error logs and debugging code on the MISE side. The [Specifications](#) detail all of the error code and header information that might be returned by the MISE for success and failure states in the interaction.

## 12. Going Live on the Integration Platform

Once the Testing is complete, the trusted system development team should perform the following actions in conjunction with the MISE Engineering Team to go live on the MISE Integration Platform.

1. Provide the public certificate for the private key that will be used for all interactions with the MISE Integration Platform.
2. The MISE engineering team will place this certificate and the trusted system information in the Trust Fabric and sign the new Trust Fabric with the MISE private key as discussed in the [Security Specification](#).
3. At an agreed-upon time, the new Trust Fabric will be loaded on the MISE Integration Platform and made available to all trusted systems. This is a hot-reload operation, which does not require that the MISE Integration Platform be taken offline.
4. The trusted system can now begin publishing to and consuming information from the MISE. The first publish/search/retrieve operations should be monitored by both the trusted system development team and the MISE Engineering Team to ensure successful operation and if any troubleshooting is required.

**VERSION 3.0**  
**RELEASE 1**  
**FEBRUARY 2015**