ViewDS Access Sentinel:
Application Integration Kit for Java

Published: March 2024
Version: 7.5.2

ViewDS Access Sentinel: Application Integration Kit for Java

March 2024

Document Lifecycle

ViewDS may occasionally update documentation between software releases. Therefore, please visit www.viewds.com to ensure you have the PDF with most recent publication date. The site also hosts the most recent version of this document in HTML format.

Contents

Introduction 1
Related documentation 1
Simple authorization requests 2
Unsigned requests 2
Signed requests 6
Requests over a secure connection 9
Obligations and advice 14
Example code 14
Multiple requests 18
Example code 18
Tracing 20
Example code 20
Appendix A: AIK structure 22
AnonymousConnector 22
XmlSigningConnector 22
ClientSslConnector 23
AuthorizationRequest 23
MultiRequest 24
AuthorizationResponse 24
MultiResponse 24
Obligation 25
Advice 25
AttributeAssignment 25
AttributeCategory 26
SubjectAttributes 26
ResourceAttributes 27
ActionAttributes 27
EnvironmentAttributes 27
DelegationInfoAttributes 27

AttributeDataType 28
Result 28
XacmlStatus 29
AIK Class Diagram 30
Appendix B: AIK exceptions 32
Class: AnonymousConnector 32
Class: XmlSigningConnector 34
Class: ClientSslConnector 34

Introduction

The ViewDS Application Integration Kit (AIK) for Java version 7.5.2 helps streamline development of a Policy Enforcement Point (PEP). It abstracts communication between a new PEP and the Policy Decision Point (PDP) component of ViewDS Access Sentinel.

Attempting to communicate with the PDP without a library is complex. The intricacies include:

- building the XACML authorization decision request;
- wrapping and sending it in a SOAP envelope; and
- intercepting the consequent response from the PDP.

In contrast, the AIK libraries simply require a PEP to make calls that supply the attributes needed to make an authorization decision.

The design of the AIK is aligned with the concept of a deny-biased PEP. This means that if the decision is permit, then the PEP will permit access. If obligations accompany the decision, then the PEP will permit access only if it understands and is able to discharge the associated obligations. All other decisions result in the denial of access.

NOTE: The AIK is not thread safe. Even though simple multi-threaded code has been implemented successfully, the kit does not guard against issues arising from multi-threading.

Related documentation

Related documents include:

- ViewDS Access Sentinel: Application Integration Kit for .NET
- ViewDS Access Sentinel: Installation and Reference Guide
- ViewDS Directory: Installation and Operation Guide

Simple authorization requests

The Java AIK class library is distributed in the following file:

- `PdpLiaison.jar`

After adding the Java AIK to your project library, you must choose which of the three connector methods will be used to send requests to the PDP. This choice is important as it determines how authentication will occur between the AIK and PDP.

Each connector method is described in the following subsections:

- Unsigned requests
- Signed requests
- Requests over a secure connection

Unsigned requests

Use the `AnonymousConnector` object to perform simple unsigned authorisation requests by following these steps:

1. Instantiate an object of the class `AnonymousConnector`:

   ```
   AnonymousConnector(URL pdpUrl, CommunicationType ct,
   PKIXParameters parameters, boolean verifySignature)
   ```

   The method has four arguments:

   - `pdpUrl` – the SOAP address (including the port number) and protocol used. For example, `new URL("http://localhost:3009").`

   NOTE: A secure SSL connection from the PDP sever can be used by specifying the HTTPS Address (including the port number) of the ViewDS server in `PdpURL`. If you use this approach, then the server certificate must be trusted by the client AIK. To verify the server certificate the Java AIK uses the `PKIXParameters` as specified below. The HTTPS Address of the ViewDS server is configured using the ViewDS Management Agent.

   - `ct` – the method used to communicate with the PDP. Available methods are XML_SOAP, XML_REST and JSON_REST. The default value is `CommunicationType.XML_SOAP`.

   NOTE: JSON_REST requires an additional third-party library, *JSR353: Java API for JSON Processing* (click here to download). For more information, visit https://jsonp.java.net/download.html. JSON_REST cannot be used if verifySignature is set to `true`.

- `parameters` – the trust anchor and/or target certificate constraints. The AIK searches the trust anchor specified to establish an SSL connection to the server and verify the digital signature in the signed response (if signature verification is enabled). For example, `new PKIXParameters(trustAnchor)`
- `verifySignature` – a flag to indicate if signatures should be verified. The signatures on PDP responses will be checked if this is set to `true`. The default value is `false`.

2. Using `AnonymousConnector`, instantiate an object of the class `AuthorizationRequest`:

   `CreateRequest()`

3. Add attributes to the request object by calling the `addElement` method:

   ```
   addElement(java.lang.String category, java.lang.String
   attribute, AttributeDataType
   attributeDataType,java.lang.String value)
   ```

   The method must be called for each attribute, and has four arguments:

   - `category` – the XACML attribute category. The list of XACML standard categories is defined in the static class `AttributeCategory`.
   - `attribute` – the XACML attribute identifier. The lists of XACML standard attributes are defined in the static classes `SubjectAttributes`, `ResourceAttributes`, `ActionAttributes`, `EnvironmentAttributes`.
   - `attributeDataType` – the attribute data type.

   NOTE: All attribute data types described in the XACML 3.0 standard are supported with the exception of XPATH expressions.

   - `value` – the attribute value.

4. Call the evaluate method of the `AnonymousConnector` to evaluate the request. The method takes the request as the argument, and returns an `AuthorizationResponse` object:

   `AuthorizationResponse evaluate(AuthorizationRequest req)`

5. Process the response. The field result from the response should be checked to establish the authorization decision.

Example code

This simple example sends an unsigned XACML authorization decision request. The request relates to a user (the subject in XACML terminology) with the username 'smith' who is attempting to 'modify' (the action) a file called 'reports summary' (the resource).

```
public static void main(String[] args) throws AikConnectionException,
                                              AikSecurityException
{
  PdpConnector connector;
  AuthorizationRequest req;
  AuthorizationResponse res;
  URL pdpUrl;
  KeyStore trustAnchor;

  // Form the URL of the PDP
  try {
    pdpUrl = new URL("http://localhost:6009");
  }
  catch (MalformedURLException ex) {
    Logger.getLogger(Example1.class.getName()).log(Level.SEVERE, null, ex);
    return;
  }

  // Retrieve trusted KeyStore from file
  // File name: `truststore_test_path'
  // KeyStore type: `Java KeyStore'
  // KeyStore password: `testpass'
  trustAnchor = readKeyStore("truststore_test_path", "JKS", "testpass". toCharArray());

  // Create AnonymousConnector object
  try {
    connector = new AnonymousConnector(pdpUrl,
                CommunicationType.XML_SOAP,
                new PKIXParameters(trustAnchor),
                true);
  }
  catch (KeyStoreException | InvalidAlgorithmParameterException ex) {
    Logger.getLogger(Example1.class.getName()).log(Level.SEVERE, null, ex);
    return;
  }

  try {
    req = connector.createRequest();
    //username: smith
    req.addElement(
      AttributeCategory.access_subject,
      SubjectAttributes.subject_id,
      AttributeDataType._string,
      "smith");
    //resource: reports summary
    req.addElement(
      AttributeCategory.resource,
      ResourceAttributes.resource_id,
      AttributeDataType._string,
      "reports summary");
    //action: modify
    req.addElement(
      AttributeCategory.action,
      ActionAttributes.action_id,
      AttributeDataType.anyURI,
      "foo:bar:modify");
    //current time
    req.addElement(
      AttributeCategory.environment,
      EnvironmentAttributes.current_time,
      AttributeDataType.time,
      PdpConnector.formatLocalTimeForXml(new Date()));

    res = connector.evaluate(req);
  }
  catch (AikException | SOAPException ex) {
    Logger.getLogger(Example1.class.getName()).log(Level.SEVERE, null, ex);
    return;
  }

  if (res.getResult().equals(Result.permit)) {
    System.out.println("Permit");
  }
  else {
    System.out.println("Not permit");
  }
}
private static KeyStore readKeyStore(String keyStorePath, String storeType, char[]
storePass)
{
  KeyStore local = null;
  try {
    local = KeyStore.getInstance(storeType);
    local.load(new FileInputStream(keyStorePath), storePass);
  }
  catch (Exception ex) {
    Logger.getLogger(Example1.class.getName()).log(Level.SEVERE, null, ex);
  }
  return local;
}
```

NOTE: There is no need to use the static classes: AttributeCategory, SubjectAttributes, ResourceAttributes, ActionAttributes and EnvironmentAttribute. You can use your own identifiers instead of the XACML standard identifiers. This is illustrated below.

```
req.addElement(
       foo:bar:myCategory,
       foo:bar:myAttribute,
       AttributeDataType._integer,
       "150");
```

Signed requests

Alternatively, use the `XmlSigningConnector` object to sign simple authorisation requests using XML digital signatures before sending them to the server:

1. Instantiate an object of the class `XmlSigningConnector`:

   ```
   XmlSigningConnector(URL pdpUrl, CommunicationType ct,
   PrivateKeyEntry signingKeyPair, CertificateInclusion
   certInclusion, PKIXParameters parameters, boolean
   verifySignature)
   ```

   The method has six arguments:

   - `pdpUrl` - the SOAP address (including the port number) and protocol used. For example, `new URL("http://localhost:3009").`

   NOTE: A secure SSL connection from the PDP sever can be used by specifying the HTTPS Address (including the port number) of the ViewDS server in `PdpURL`. If you use this approach, then the server certificate must be trusted by the client AIK. To verify the server certificate the Java AIK uses the `PKIXParameters` as specified below. The HTTPS Address of the ViewDS server is configured using the ViewDS Management Agent.

   - `ct` - the method used to communicate with the PDP. Available methods are XML_SOAP and XML_REST. The default value is `CommunicationType.XML_SOAP`.

   NOTE: The XmlSigningConnector method does not support the communication type JSON_REST.

   - `signingKeyPair` – the signing keypair entry which contains a private key and the corresponding certificate chain, used for signing the requests.
   - `certInclusion` – determines whether the signing certificate only or all of the certificates in the certificate chain should be included in the signature. For example, `CertificateInclusion.certificateChain`.
   - `parameters` – the trust anchor and/or target certificate constraints. The AIK searches the trust anchor specified to establish an SSL connection to the server and verify the digital signature in the signed response (if signature verification is enabled). For example, `new PKIXParameters(trustAnchor).`
   - `verifySignature` – a flag to indicate if signatures should be verified. The signatures on PDP responses will be checked if this is set to `true`. The default value is `false`.

2. Using `XmlSigningConnector`, instantiate an object of the class `AuthorizationRequest`:
   `CreateRequest()`

3. Add attributes to the request object by calling the `addElement` method:

   `addElement(java.lang.String category, java.lang.String attribute, AttributeDataType attributeDataType,java.lang.String value)`

   The method must be called for each attribute, and has four arguments:

   - `category` – the XACML attribute category. The list of XACML standard categories is defined in the static class `AttributeCategory`.
   - `attribute` – the XACML attribute identifier. The lists of XACML standard attributes are defined in the static classes `SubjectAttributes`, `ResourceAttributes`, `ActionAttributes`, `EnvironmentAttributes`.
   - `attributeDataType` – the attribute data type.

   NOTE: All attribute data types described in the XACML 3.0 standard are supported with the exception of XPATH expressions.

   - `value` – the attribute value.

4. Call the evaluate method of the `XmlSigningConnector` to evaluate the request. The method takes the request as the argument, and returns an `AuthorizationResponse` object:

   `AuthorizationResponse evaluate(AuthorizationRequestreq)`

5. Process the response. The field result from the response should be checked to establish the authorization decision.

Example code

This example is the same as the one given for unsigned requests but in this case a signed XACML authorization decision request is sent.

```
public static void main(String[] args) throws AikConnectionException,
                                              AikSecurityException
{
  PdpConnector connector;
  AuthorizationRequest req;
  AuthorizationResponse res;
  URL pdpUrl;
  KeyStore trustAnchor;
  KeyStore keyStore;
  PrivateKeyEntry signingKeyPair;
  ProtectionParameter password;

  // Form the URL of PDP
  try {
    pdpUrl = new URL("http://localhost:6009");
  }
  catch (MalformedURLException ex) {
         Logger.getLogger(Example2.class.getName()).log(Level.SEVERE, null, ex);
         return;
  }
```

```
// Retrieve trusted KeyStore from file
// File name: `truststore_test_path'
// KeyStore type: `Java KeyStore'
// KeyStore password: `testpass'
trustAnchor = readKeyStore("truststore_test_path", "JKS", "testpass".toCharArray());

// Retrieve KeyStore which contains the signing KeyPair
// File name: `mykeystore.jks'
// KeyStore type: `Java KeyStore'
// KeyStore password: `testpass'
keyStore = readKeyStore("mykeystore.jks", "JKS", "testpass".toCharArray());
try {
  // Retrieving KeyPair from keyStore
  // Signing key Alias: `mhunter'
  // Signing key Alias password: empty
  password = new KeyStore.PasswordProtection("".toCharArray());
  signingKeyPair = (KeyStore.PrivateKeyEntry)keyStore.getEntry("mhunter", password);
}
catch (Exception ex) {
  Logger.getLogger(Example2.class.getName()).log(Level.SEVERE, null, ex);
  return;
}

// Create XmlSigningConnector object
try {
  connector = new XmlSigningConnector(pdpUrl,
              CommunicationType.XML_SOAP
              signingKeyPair,
              CertificateInclusion.certificateChain,
              new PKIXParameters(trustAnchor),
              true);
}
catch (KeyStoreException | InvalidAlgorithmParameterException ex) {
  Logger.getLogger(Example2.class.getName()).log(Level.SEVERE, null, ex);
  return;
}

try {
  req = connector.createRequest();

  //username: smith
  req.addElement(
    AttributeCategory.access_subject,
    SubjectAttributes.subject_id,
    AttributeDataType._string,
    "smith");

  //resource: reports summary
  req.addElement(
    AttributeCategory.resource,
    ResourceAttributes.resource_id,
    AttributeDataType._string,
    "reports summary");

  //action: modify
  req.addElement(
    AttributeCategory.action,
    ActionAttributes.action_id,
    AttributeDataType.anyURI,
    "foo:bar:modify");
```

```
      //current time
      req.addElement(
        AttributeCategory.environment,
        EnvironmentAttributes.current_time,
        AttributeDataType.time,
        PdpConnector.formatLocalTimeForXml(new Date()));

      res = connector.evaluate(req);
    }
    catch (AikException | SOAPException ex) {
      Logger.getLogger(Example2.class.getName()).log(Level.SEVERE, null, ex);
      return;
    }

    if (res.getResult().equals(Result.permit)) {
      System.out.println("Permit");
    }
    else {
      System.out.println("Not permit");
    }
  }
  private static KeyStore readKeyStore(String keyStorePath, String storeType, char[]
  storePass)
  {
    KeyStore local = null;
    try {
      local = KeyStore.getInstance(storeType);
      local.load(new FileInputStream(keyStorePath), storePass);
    }
    catch (Exception ex) {
      Logger.getLogger(Example2.class.getName()).log(Level.SEVERE, null, ex);
    }
    return local;
  }
```

Requests over a secure connection

Or use the `ClientSslConnector` object to send requests over a secure HTTPS connection by following these steps:

NOTE: When `ClientSslConnector` is used both the AIK client and the PDP server have to present their certificates to each other for authentication to occur.

1. Instantiate an object of the class `ClientSslConnector`:

   `ClientSslConnector(URL pdpUrl, CommunicationType ct, PrivateKeyEntry clientKeyPair, PKIXParameters parameters, boolean verifySignature)`

   The method has five arguments:

   - `pdpUrl` - the SOAP address (including the port number) and protocol used. For example, `new URL("http://localhost:3009")`.

- `ct` - the method used to communicate with the PDP. Available methods are XML_SOAP, XML_REST and JSON_REST. The default value is `CommunicationType.XML_SOAP`.

  NOTE: To use the communication type JSON_REST an additional third party library - JSR353: Java API for JSON Processing - is required. You can download the library here. To find out more about this library visit https://jsonp.java.net/download.html.

  NOTE: The communication type JSON_REST cannot be used if verifySignature is set to `true`.

- `clientKeyPair` – the client keypair entry which contains a private key and the corresponding certificate chain, used by the client to authenticate itself against the server.
- `parameters` – the trust anchor and/or target certificate constraints. The AIK searches the trust anchor specified to establish an SSL connection to the server and verify the digital signature in the signed response (if signature verification is enabled). For example, `new PKIXParameters(trustAnchor).`
- `verifySignature` – a flag to indicate if signatures should be verified. The signatures on PDP responses will be checked if this is set to `true`. The default value is `false`.

2. Using `ClientSslConnector`, instantiate an object of the class `AuthorizationRequest`:

   ```
   CreateRequest()
   ```

3. Add attributes to the request object by calling the `addElement` method:

   ```
   addElement(java.lang.String category, java.lang.String
   attribute, AttributeDataType
   attributeDataType,java.lang.String value)
   ```

   The method must be called for each attribute, and has four arguments:

   - `category` – the XACML attribute category. The list of XACML standard categories is defined in the static class `AttributeCategory`.
   - `attribute` – the XACML attribute identifier. The lists of XACML standard attributes are defined in the static classes `SubjectAttributes`, `ResourceAttributes`, `ActionAttributes`, `EnvironmentAttributes`.
   - `attributeDataType` – the attribute data type.

   NOTE: All attribute data types described in the XACML 3.0 standard are supported with the exception of XPATH expressions.

- `value` – the attribute value.

4. Call the evaluate method of the `ClientSslConnector` to evaluate the request. The method takes the request as the argument, and returns an `AuthorizationResponse` object:

   `AuthorizationResponse evaluate(AuthorizationRequest req)`

5. Process the response. The field result from the response should be checked to establish the authorization decision.

Example code

This example is the same as the previous ones but in this case the XACML authorization decision request is sent over a secure HTTPS connection.

```
public static void main(String[] args) throws AikSecurityException,
                                              AikConnectionException
{
  PdpConnector connector;
  AuthorizationRequest req;
  AuthorizationResponse res;
  KeyStore keyStore;
  KeyStore trustAnchor;
  PrivateKeyEntry clientKeyPair;
  ProtectionParameter password;
  URL pdpUrl;

  // Form the URL of PDP
  try {
    pdpUrl = new URL("https://localhost:6010");
  }
  catch (MalformedURLException ex) {
    Logger.getLogger(Example3.class.getName()).log(Level.SEVERE, null, ex);
    return;
  }

  // Retrieve trusted KeyStore from file
  // File name: `truststore_test_path'
  // KeyStore type: `Java KeyStore'
  // KeyStore password: `testpass'
  trustAnchor = readKeyStore("truststore_test_path", "JKS", "testpass".toCharArray());

  // Retrieve KeyStore which contains the client KeyPair
  // File name: `mykeystore.jks'
  // KeyStore type: `Java KeyStore'
  // KeyStore password: `testpass'
  keyStore = readKeyStore("mykeystore.jks", "JKS", "testpass".toCharArray());
  try {
    // Retrieving KeyPair from keyStore
    // Signing key Alias: `mhunter'
    // Signing key Alias password: empty
    password = new KeyStore.PasswordProtection("".toCharArray());
    clientKeyPair = (KeyStore.PrivateKeyEntry)keyStore.getEntry("asherma", password);
  }
  catch (Exception ex) {
    Logger.getLogger(Example3.class.getName()).log(Level.SEVERE, null, ex);
    return;
  }
  // Create ClientSslConnector object
  try {
    connector = new ClientSslConnector(pdpUrl,
                CommunicationType.XML_REST,
                clientKeyPair,
                new PKIXParameters(trustAnchor),
                false);
  }
  catch (KeyStoreException | InvalidAlgorithmParameterException ex) {
    Logger.getLogger(Example3.class.getName()).log(Level.SEVERE, null, ex);
    return;
  }

  try {
    req = connector.createRequest();
    //username: smith
    req.addElement(
      AttributeCategory.access_subject,
      SubjectAttributes.subject_id,
      AttributeDataType._string,
      "smith");

    //resource: reports summary
    req.addElement(
      AttributeCategory.resource,
      ResourceAttributes.resource_id,
      AttributeDataType._string,
      "reports summary");

    //action: modify
    req.addElement(
       AttributeCategory.action,
       ActionAttributes.action_id,
       AttributeDataType.anyURI,
       "foo:bar:modify");

    //current time
    req.addElement(
      AttributeCategory.environment,
      EnvironmentAttributes.current_time,
      AttributeDataType.time,
      PdpConnector.formatLocalTimeForXml(new Date()));

    res = connector.evaluate(req);
  }
  catch (AikException | SOAPException ex) {
    Logger.getLogger(Example3.class.getName()).log(Level.SEVERE, null, ex);
    return;
  }

  if (res.getResult().equals(Result.permit)) {
    System.out.println("Permit");
  }
  else {
    System.out.println("Not permit");
  }
}

private static KeyStore readKeyStore(String keyStorePath, String storeType,
                       char[] storePass)
{
       KeyStore local = null;
```

```
        try {
                local = KeyStore.getInstance(storeType);
                local.load(new FileInputStream(keyStorePath), storePass);
        }
        catch (Exception ex) {
                Logger.getLogger(Example3.class.getName()).log(Level.SEVERE, null, ex);
        }
        return local;
}
```

Obligations and advice

Obligations and advice are features of XACML 3.0 that can be used to convey directives to applications that define them within an XACML response. An obligation is a mandatory directive whereas advice is optional.

To illustrate, an obligation to add a log entry might be associated with permitting access to a highly restricted resource. In this case, when the application is told that access is permitted it is also told that it is obliged to log the access for auditing purposes. If the application cannot perform the logging operation, it will refuse access to the resource.

The application using the AIK is required to register known obligations. This is intended to ensure that all obligations are identified and supported by the application, and that any unsupported obligations result in the application returning `denyDueToUnrecognizedObligations`. To register obligations use:

`registerObligation(java.lang.String obligationId)`

The `Obligation` object is used to return obligations in the authorization response.

Advice is similar to an obligation, except execution of advice by the application is optional.

For example an XACML response might deny access to a document on the weekend and come with the advice to show a message to the user that access is only available on week days.

The `Advice` object is used to return advice in the authorization response.

NOTE: The specific obligations and advice implemented by a given application are defined by that application. The Java AIK merely provides a mechanism for handling authorization responses that include obligations and advice.

Example code

This example shows how to register and fulfil an obligation. For the sake of brevity simple strings are used as identifiers for attribute assignments (e.g. email and recipientaddress) in place of URIs (e.g. foo:bar:recipientaddress).

```
public class Example5
{
  public static void main(String[] args) throws AikSecurityException,
                                               AikConnectionException
  {
    PdpConnector connector;
    AuthorizationRequest req;
    AuthorizationResponse response;
    URL pdpUrl;

    // Form the URL of PDP
    try {
      pdpUrl = new URL("http://localhost:6009");
    }
    catch (MalformedURLException ex) {
      Logger.getLogger(Example5.class.getName()).log(Level.SEVERE, null, ex);
      return;
    }

    // Create AnonymousConnector object
    connector = new AnonymousConnector(pdpUrl,
                CommunicationType.XML_SOAP,
                null,
                false);
    connector.registerObligation("foo:bar:email");

    try {
      req = connector.createRequest();
      req.addElement(
        AttributeCategory.access_subject,
        SubjectAttributes.subject_id,
        AttributeDataType._string,
        "asherma");

      req.addElement(
        AttributeCategory.action,
        ActionAttributes.action_id,
        AttributeDataType._string,
        "foo:bar:modify");

      response = connector.evaluate(req);

      switch (response.getResult()) {
        case deny:
          //deny
          break;
        case denyWithObligations:
        //deny
          fulfilObligations(response.getObligations());
          break;
        case denyDueToUnrecognizedObligations:
          //deny
          break;
        case denyUnlessAllObligationsSatisfied:
          if (fulfilObligations(response.getObligations())) {
            //permit
          }
          else {
            //deny
          }
          break;
        case permit:
          //permit
          break;
      }
      System.out.println("Result: " + response.getResult().toString());
    }
    catch (AikException | SOAPException ex) {
      Logger.getLogger(Example5.class.getName()).log(Level.SEVERE, null, ex);
    }
  }

  private static boolean fulfilObligations(List<Obligation> obligations)
  {
    for (Obligation ob : obligations) {
      if (ob.getId().equals("foo:bar:email")) {
        sendEmail(ob);
      }
      else {
        return false;
      }
    }
    return true;
  }

  private static boolean sendEmail(Obligation ob)
  {
    // List of Recipient's email IDs needs to be mentioned.
    List<String> recipientAddresses = new ArrayList<String>();
    // Sender's email ID needs to be mentioned
    String from = "test@viewds.com";
    // Host name of the mail server
    String host = "viewds.com";
    String subject = "";
    String body = "";

    Properties properties;

    for (AttributeAssignment aa : ob.getAttributes()) {
      String attId;
      String attCat;
      String attVal;
      attId = aa.getAttributeId().toLowerCase();
      attCat = aa.getCategoryId().toLowerCase();
      attVal = aa.getAttributeValue();

      if (attCat.equals("email") && attId.equals("recipientaddress")) {
        recipientAddresses.add(attVal);
      }

      if (attCat.equals("email") && attId.equals("subject")) {
        subject = attVal;
      }

      if (attCat.equals("email") && attId.equals("body")) {
        body = attVal;
      }
    }

    if (recipientAddresses.isEmpty()) {
      return false;
    }

    // Get system properties
    properties = System.getProperties();
    // Setup mail server
    properties.setProperty("mail.smtp.host", host);
    // Get the default Session object.
    Session session = Session.getDefaultInstance(properties);

    try {
      // Create a default MimeMessage object.
      MimeMessage message = new MimeMessage(session);
      // Set From: header field of the header.
      message.setFrom(new InternetAddress(from));
      // Set To: header field of the header.
      for (String recipient : recipientAddresses) {
        message.addRecipient(Message.RecipientType.TO, new InternetAddress(recipient));
      }
      // Set Subject: header field
      message.setSubject(subject);
      // Now set the actual message
      message.setText(body);
      // Send message
      Transport.send(message);
    }
    catch (MessagingException ex) {
      Logger.getLogger(Example5.class.getName()).log(Level.SEVERE, null, ex);
      return false;
    }
    return true;
  }
}
```

Multiple requests

In addition to sending individual authorisation requests, the . NET and Java AIKs also allow you to create multiple authorization requests and add them to one `MultiRequest` object. The `MultiRequest` object is then sent that to the PDP, which returns and `MultiResponse` object. Each request added to the multi-request is assigned a UID which is used to identify the corresponding result element in the multi-response.

This feature is particularly useful in circumstances where one access control action by the application requires more than one authorization decision to be made.

For example, if a user (subject) is trying to view (action) a list of documents (resources), then an authorization decision is required for each item on the list. In such a scenario, sending all the requests in a single message, rather than sending one message for each request, reduces the messaging overhead considerably.

Example code

```
public class Example4
{
  public static void main(String[] args) throws AikSecurityException,
                                                AikConnectionException
  {
    PdpConnector connector;
    AuthorizationRequest req1, req2;
    MultiRequest mulReq;
    MultiResponse mulRes;
    URL pdpUrl;

    // Form the URL of PDP
    try {
      pdpUrl = new URL("http://localhost:6009");
    }
    catch (MalformedURLException ex) {
      Logger.getLogger(Example4.class.getName()).log(Level.SEVERE, null, ex);
      return;
    }

    // Create AnonymousConnector object
    connector = new AnonymousConnector(pdpUrl,
                CommunicationType.XML_SOAP,
                null,
                false);

    mulReq = new MultiRequest(false);

    try {
      req1 = connector.createRequest();

      req1.addElement(
        AttributeCategory.access_subject,
        SubjectAttributes.role,
        AttributeDataType._string,
        "MANAGER");
      req1.addElement(
        AttributeCategory.resource,
        ResourceAttributes.resource_id,
        AttributeDataType._string,
        "REPORT A");

      req2 = connector.createRequest();

      req2.addElement(
        AttributeCategory.access_subject,
        SubjectAttributes.role,
        AttributeDataType._string,
        "MANAGER");

      req2.addElement(
        AttributeCategory.resource,
        ResourceAttributes.resource_id,
        AttributeDataType._string,
        "REPORT B");

      mulReq.addRequest(req1);
      mulReq.addRequest(req2);

      mulRes = connector.evaluate(mulReq);

      if (mulRes.getResultForRequest(req1).getDenyBiasedResult().equals(Result.permit)) {
        System.out.println("Permit");
      }
      else {
        System.out.println("Not permit");
      }
      if (mulRes.getResultForRequest(req2).getDenyBiasedResult().equals(Result.permit)) {
        System.out.println("Permit");
      }
      else {
        System.out.println("Not permit");
      }
    }
    catch (SOAPException | AikException ex) {
      Logger.getLogger(Example4.class.getName()).log(Level.SEVERE, null, ex);
    }
  }
}
```

Tracing

The .NET and Java AIKs provide a tracing feature to allow you to investigate the cause of any unexpected responses you obtain from the PDP. If trace information is requested, the response from the PDP will include information about the policy evaluation process that took place on the server.

NOTE: Tracing is not supported for the communication type JSON_REST.

To request trace information you must set the `traceSwitch`:

```
AuthorizationRequest req = new AuthorizationRequest(true);
```

NOTE: In order to get trace information, tracing must also be enabled on the ViewDS Directory. See *Enable tracing* in the *ViewDS Access Sentinel: Installation and Reference Guide* for details.

Trace information will then be available in the `traceInfo` property included in the response.

Example code

This example shows how to switch on tracing.

```
public class Example6
{
  public static void main(String[] args) throws AikSecurityException,
                                                AikConnectionException
  {
    PdpConnector connector;
    AuthorizationRequest req;
    AuthorizationResponse res;
    URL pdpUrl;

    // Form the URL of PDP
    try {
      pdpUrl = new URL("http://localhost:6009");
    }
    catch (MalformedURLException ex) {
      Logger.getLogger(Example6.class.getName()).log(Level.SEVERE, null, ex);
      return;
    }

    // Create AnonymousConnector object
    connector = new AnonymousConnector(pdpUrl, CommunicationType.XML_SOAP, null, false);
    try {
      req = connector.createRequest();
      req.addElement(
        AttributeCategory.resource,
        ResourceAttributes.resource_id,
        AttributeDataType._string,
        "REPORT A");
      res = connector.evaluate(req);

      //print the tracing information from the response
      System.out.println(res.getTraceInfo());
    }
    catch (AikException | SOAPException ex) {
      Logger.getLogger(Example6.class.getName()).log(Level.SEVERE, null, ex);
    }
  }
}
```

Appendix A: AIK structure

The Java AIK is a class library developed in Java and distributed in following file:

- `PdpLiaison.jar`

The library exposes the following members.

AnonymousConnector

This class is used to configure an anonymous connection to a PDP. An object of this type should be instantiated at the beginning to be used for sending anonymous authorization requests to the PDP.

```
AnonymousConnector :: PdpConnector
  AnonymousConnector()
  AnonymousConnector(URL pdpUrl, CommunicationType ct, PKIXParameters param,
  boolean verifySignature)
  getCommunicationType(): CommunicationType
  evaluate(AuthorizationRequest request): AuthorizationResponse
  evaluate(MultiRequest multiReq): MultiResponse
```

XmlSigningConnector

This class is used to configure a connection to a PDP through which signed authorisation request can be sent. An object of this type should be instantiated at the beginning to be used to sign authorisation requests using XML digital signatures before sending them to the PDP.

```
XmlSigningConnector :: PdpConnector
  XmlSigningConnector()
  XmlSigningConnector(URL pdpUrl, CommunicationType ct, PrivateKeyEntry
  signingKeyPair, CertificateInclusion certInclusion, PKIXParmaeters
  parameters, boolean verifySignature)
  getCommunicationType(): CommunicationType
  getCertificateInclusion(): CertificateInclusion
  getSigningKeyPair(): PrivateKeyEntry
  evaluate(AuthorizationRequest request): AuthorizationResponse
  evaluate(MultiRequest multiReq): MultiResponse
```

ClientSslConnector

This class is used to configure a secure HTTPS connection to a PDP. An object of this type should be instantiated at the beginning to be used to send authorisation requests to the PDP via a secure HTTPS connection.

Public methods:

```
ClientSslConnector :: PdpConnector
  ClientSslConnector()
  ClientSslConnector(URL pdpUrl, PrivateKeyEntry clientKeyPair, PKIXParameters
  param, boolean verifySignature, CommunicationType ct)
  getCommunicationType(): CommunicationType
  getCertificateInclusion(): CertificateInclusion
  evaluate(AuthorizationRequest request): AuthorizationResponse
  evaluate(MultiRequest multiReq): MultiResponse
```

AuthorizationRequest

Objects of this type should be instantiated for each XACML authorization decision request to be sent to the PDP.

```
AuthorizationRequest :: RequestBase
  AuthorizationRequest()
  AuthorizationRequest(boolean traceSwitch)
  AuthorizationRequest(boolean traceSwitch, boolean returnPolicyIdSwitch)
  AuthorizationRequest(boolean traceSwitch, boolean returnPolicyIdSwitch,
  boolean combinePoliciesSwitch)
  setConent(String category, Element content)
  addElement(String category, String attribute, AttributeDataType
  attributeDataType, String value)
  addElement(String category, String attribute, String dataType, String value)
  addElement(String category, String attribute, String dataType, String value,
  boolean includeInResult)
  getUID() : UID
```

MultiRequest

Multiple AuthorizationRequest objects can be added to an object of this type and sent together to the PDP which then provides a MultiReponse. Each request added to the MultiRequest object is assigned a UID which is used to identify the corresponding result element in the MultiResponse.

```
MultiRequest :: RequestBase
  MultiRequest(boolean returnPolicyIdList)
  addRequest(AuthorizationRequest request)
  getRequest() : List<AuthorizationRequest>
```

AuthorizationResponse

The results of an authorization decision request are returned as objects of this type.

```
AuthorizationResponse
  AuthorizationResponse(Element serverResponse)
  AuthorizationResponse(String serverResponse)
  getAssociatedAdvice() : List<Advice>
  getObligations() : List<Obligation>
  getPdpDecision() : XacmlResult
  getPolicyIdReferences(): List<PolicyIdReference>
  getResult() : Result
  getTraceInfo() : String
  getXacmlStatus() : XacmlStatus
  getXacmlStatusMessage() : String
  toString() : String
```

MultiResponse

The results of a MultiRequest authorization decision request are returned as objects of this type.

```
MultiResponse
  MultiResponse(Element serverResponse)
  MultiResponse(String serverResponse)
  getResults() : List<XacmlResultElement>
  getResultForRequest(String requestUID) : LXacmlResultElement
  getResultForRequest(AuthorizationRequest request) : LXacmlResultElement
  getResultForRequest(UID requestUID) : LXacmlResultElement
  toString() : String
```

Obligation

The obligations to be fulfilled are returned as objects of this type which are included in the authorization response.

```
Obligation :: ObligationAdvice
  Obligation(SOAPElement node)
  equals(Object obj) : boolean
  getAttributes() : List<AttributeAssignment>
  getClass() : Class<?>
  getId() : String
```

Advice

The advice to be fulfilled is returned as objects of this type which are included in the authorization response.

```
Advice :: ObligationAdvice
  Advice(SOAPElement node)
  equals(Object obj) : boolean
  getAttributes() : List<AttributeAssignment>
  getClass() : Class<?>
  getId() : String
```

AttributeAssignment

The attributes of an obligation or an advice are objects of this type.

```
AttributeAssignment
  AttributeAssignment(SOAPElement node)
  AttributeAssigment()
  getAttributeId() : String
  getAttributeValue() : String
  getCategoryId() : String
  getDataType() : String
  toString() : String
```

AttributeCategory

A list of constant strings that represent identifiers of the standard attribute categories specified in the XACML core specification.

```
AttributeCategory
  access_subject : String
  action : String
  codebase : String
  delegation_category : String
  delegation_info : String
  environment : String
  intermediary_subject : String
  recipient_subject : String
  requesting_machine : String
  resource : String
  role : String
```

SubjectAttributes

A list of constant strings that represent identifiers of the standard subject attributes specified in the XACML core specification.

```
SubjectAttributes
  authentication_method : String
  authentication_time : String
  dns_name: String
  ip_address : String
  key_info : String
  request_time : String
  role : String
  session_start_time : String
  subject_id : String
  subject_id_qualifier : String
```

ResourceAttributes

A list of constant strings that represent identifiers of the standard resource attributes specified in the XACML core specification.

```
ResourceAttributes
  resource_id : String
  target_namespace : String
  xpath : String
```

ActionAttributes

A list of constant strings that represent identifiers of the standard action attributes specified in the XACML core specification.

```
ActionAttributes
  action_id : String
  action_namespace : String
  implied_action: String
```

EnvironmentAttributes

A list of constant strings that represent identifiers of the standard environment attributes specified in the XACML core specification.

```
EnvironmentAttributes
  current_date : String
  current_dateTime : String
  current_time : String
  request_id : String
  trace : String
```

DelegationInfoAttributes

A list of constant strings that represent identifiers of the standard delegationInfo attributes specified in the XACML core specification.

```
DelegationInfoAttributes
  decision : String
```

AttributeDataType

An enumerative list of standard data types specified in the XACML core specification.

```
AttributeDataType
  values() : AttributeDataType[]
  valueOf(String name) : AttributeDataType
  _boolean
  _double
  _string
  anyType
  anyURI
  base64Binary
  date
  dateTime
  dayTimeDuration
  dnsName
  hexBinary
  integer
  ipAddress
  rfc822Name
  time
  x500Name
  yearMonthDuration
  getValue() : String
  getValue(AttributeDataType dataType) : String
```

Result

An enumerative list of authorization results specified in the AIK.

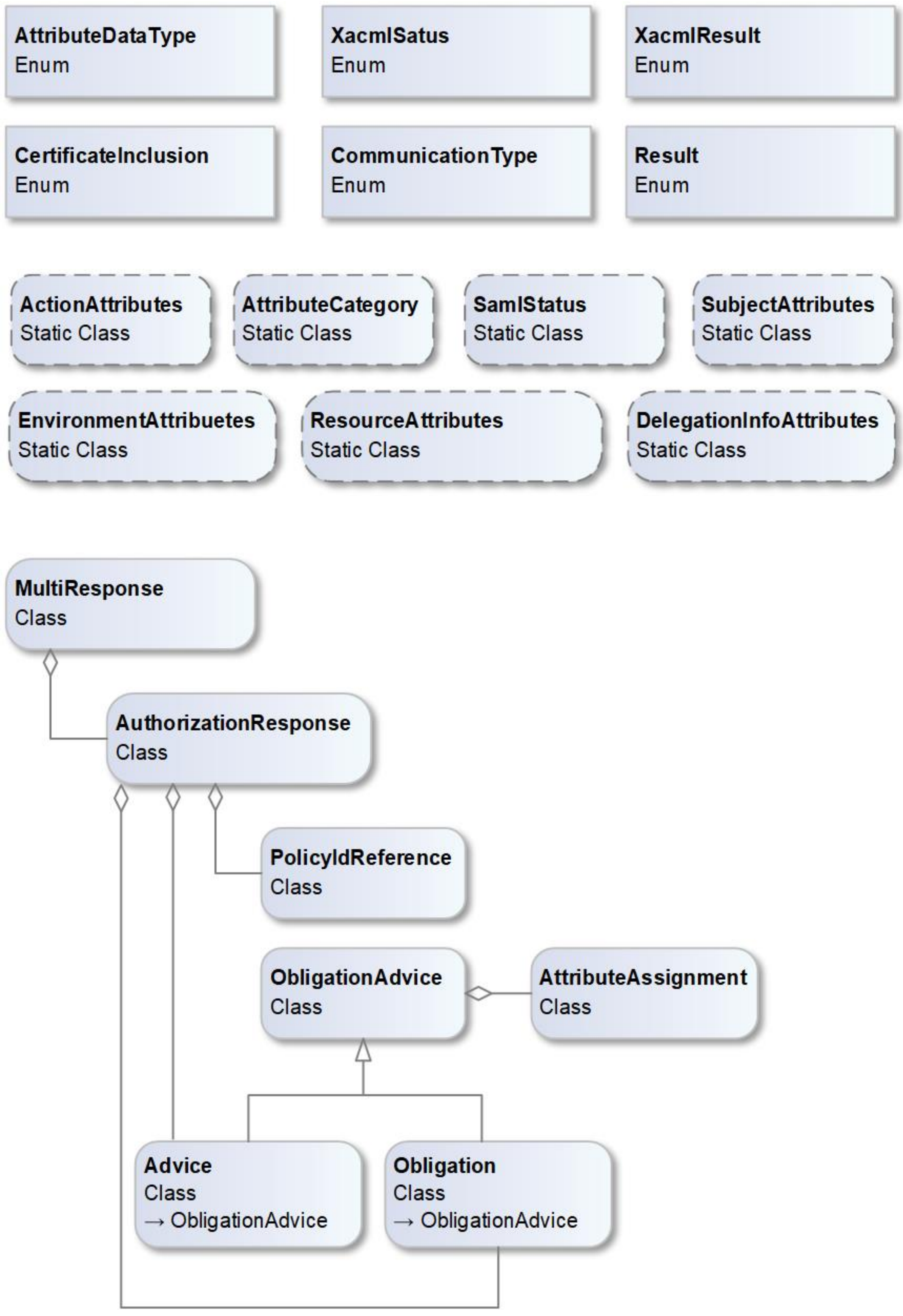
```
Result
  valueOf(String name): Result
  values() : Result[]
  deny
  denyDueToUnrecognizedObligations
  denyUnlessAllObligationsSatisfied
  denyWithObligations
  permit
```

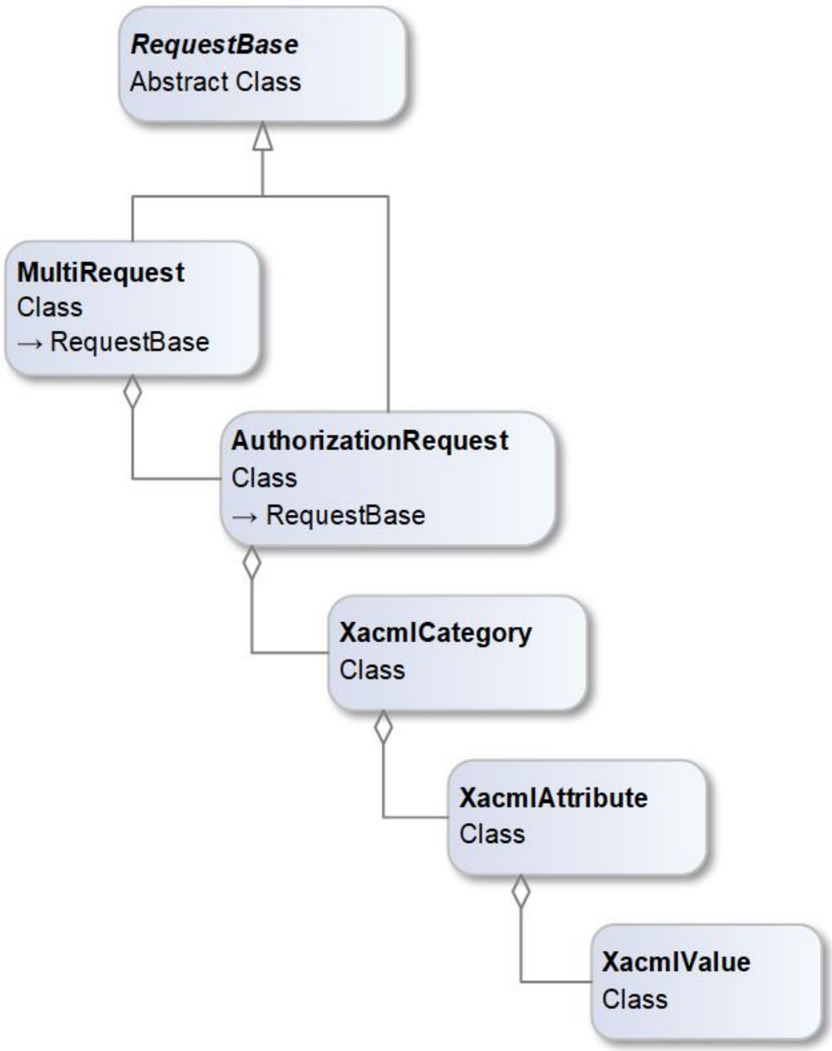
XacmlStatus

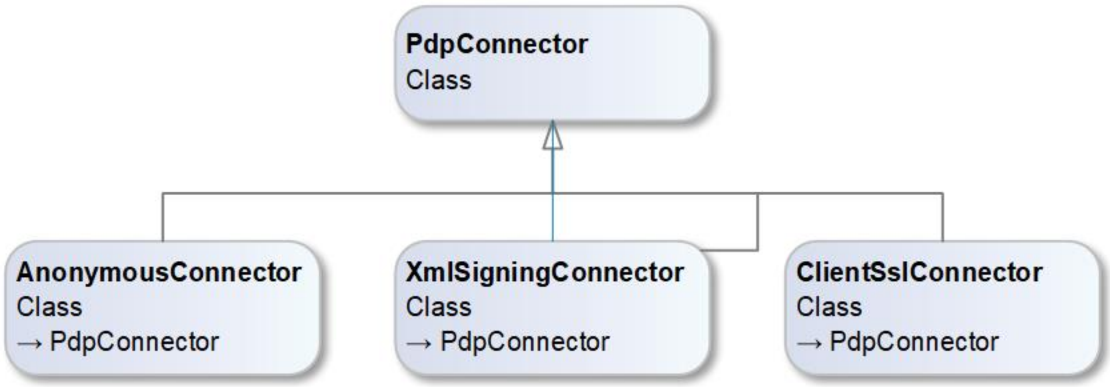
An enumerative list of standard XACML status types.

```
XacmlStatus
  valueOf(String name): XacmlStatus
  values() : XacmlStatus[]
  missingAttribute
  notDefined
  ok
  processingError
  syntaxError
```

AIK Class Diagram



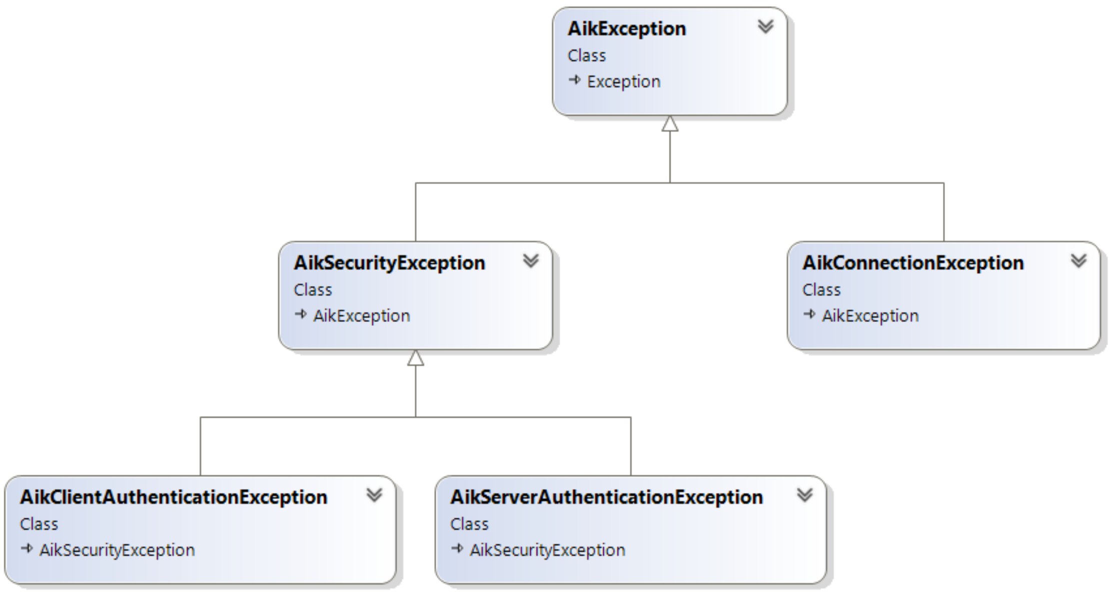

RequestBase
Abstract Class
MultiRequest
Class
→ RequestBase
AuthorizationRequest
Class
→ RequestBase
XacmlCategory
Class
XacmlAttribute
Class
XacmlValue
Class


PdpConnector
Class
AnonymousConnector
Class
→ PdpConnector
XmlSigningConnector
Class
→ PdpConnector
ClientSslConnector
Class
→ PdpConnector

Appendix B: AIK exceptions

The Java AIK throws exceptions when errors occur, for example, when the AIK fails to send a request to the PDP because the PDP is unreachable.

A series of exception classes have been added to Java AIK to handle these, as shown below:



This appendix lists all of the exceptions thrown for each supported connector class and their causes.

Class: AnonymousConnector

Method: evaluate

Cause: failure in sending the HTTP request to the PDP

- Exception type: AikConnectionException
- Exception message: "Error in sending the HTTP request to PDP: " + innerWebException.Message

Cause: saml request Id mismatch with InResponseTo Id

- Exception type: AikSecurityException
- Exception message: "Request ID does not match the response ID"

Cause: failure in finding a signature in the response

- Exception type: AikServerAuthenticationException
- Exception message: "Verification failed: No Signature was found in the message."

Cause: finding more than one signature on the response

- Exception type: AikServerAuthenticationException
- Exception message: "Verification failed: More that one signature was found for the message."

Cause: failure in signature validation

- Exception type: AikServerAuthenticationException
- Exception message: "Invalid signature."

Cause: failure in certificate validation

- Exception type: AikServerAuthenticationException
- Exception message: "Certificate not trusted."

Cause: failure in finding the X509SubjectName in the response in the absence of certificate in the response

- Exception type: AikServerAuthenticationException
- Exception message: "Subject name of the signing certificate not found."

Cause: failure in finding a certificate in the identified store with the identified X509SubjectName

- Exception type: AikServerAuthenticationException
- Exception message: "Certificate with the identified subject name does not exist in the certificate store.

Cause: finding more than one certificate in the identified store with the identified X509SubjectName

- Exception type: AikServerAuthenticationException
- Exception message: "More than one certificate with the identified subject name in the certificate store."

Cause: the inResponseTo field of the received response does not match the queryId of the sent request

- Exception type: AikException
- Exception message: Request ID does not match the response ID.

Constructor initialization

Cause: invalid constructor's parameter combination. XML signature in json rest is invalid.

- Exception type: AikSecurityException
- Exception message: "XML Signature is not supported in JSON_REST"

Cause: invalid constructor's parameter combination. AIK requires parameters attribute to be set in order to establish SSL connection.

- Exception type: AikSecurityException
- Exception message: "secure connection is set, parameters cannot be null."

Cause: invalid constructor's parameter combination. AIK requires parameters attribute to be set in order to verify signed responses.

- Exception type: AikSecurityException
- Exception message: "verify signature flag is set, parameters cannot be null."

Class: XmlSigningConnector

All of the exceptions for AnonymousConnector plus the following.

Method: evaluate

Cause: server does not accept the AIK's signature on the request and returns urn:oasis:names:tc:SAML:2.0:status:AuthnFailed as the SAML status.

- Exception type: AikClientAuthenticationException
- Exception message: "Authentication failed."

Class: ClientSslConnector

All of the exceptions for AnonymousConnector plus the following.

Method: evaluate

Cause: server does not allow the SSL connection from the AIK because of the client's certificate.

- Exception type: AikClientAuthenticationException
- Exception message: "Authentication failed."