ViewDS Application Integration Kit for .NET

Published: December 2020
Version: 7.5.1

ViewDS Application Integration Kit for .NET

December 2020

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

Overview

The Application Integration Kits (AIKs) for .NET and Java Version 7.5.1 abstract communication between a Policy Enforcement Point (PEP) and the Policy Decision Point (PDP) component of ViewDS Access Sentinel. It therefore helps streamline development of a PEP.

Attempting to communicate with the PDP without the library is complex. There are the intricacies of 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 .NET and Java AIKs 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 that 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.

Simple authorization requests

The .NET AIK is a class library developed in C# and compiled into files:

- `PdpLiaison.dll` (binaries)
- `PdpLiaison.xml` (documentation)

After adding the .NET AIK to your project library, you must choose which one of the three available connector methods you will use to send requests to the PDP. This choice is important as it determines how authentication between the AIK and the PDP will occur. The following options are available:

- Unsigned requests with AnonymousConnector
- Signed requests with XmlSigningConnector
- Requests over a secure connection with ClientSslConnector

Each of these methods is described in more detail in the following sections.

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(Uri PdpUrl, CommunicationType ct,
   X509Store trustStore, boolean verifySignature,
   X509ChainPolicy chainPolicy)
   ```

   The method has the following five arguments.

   - `PdpUrl` – the SOAP address (including the port number) and protocol used. For example, `new Uri("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 .NET AIK uses the windows certificate store 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: The communication type JSON_REST cannot be used if verifySignature is set to `true`.

- `trustStore` – the certificate store that is searched by the AIK to find the certificate that the server used to sign the responses, when only the subject name of the signing certificate is included in the signature.
- `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`.
- `chainPolicy` – by default the AIK builds a simple chain for the certificates and applies the base policy to that chain. However, if this optional argument is specified, then the value given will be applied to the chain.

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

   ```
   CreateRequest()
   ```

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

   ```
   addElement(string category, string attribute,
   AttributeDataType attributeDataType, 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 xpathExpression.

   - `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(AuthorizationRequestreq)
   ```

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 with the following details: a user (the subject in XACML terminology) with the username smith is attempting to modify (the action) a document called reports summary (the resource).

```
using System;
using PdpLiaison;
```

```
using PdpLiaison.Exceptions;

namespace SampleCodes
{
       class SimpleAuthzTest
       {
               static void Main(string[] args)
               {
                       PdpConnector connector;
                       AuthorizationRequest req;
                       AuthorizationResponse res;
                       Uri pdpUrl = new Uri("http://localhost:6009");

                       // Create AnonymousConnector object
                       try {
                               connector = new AnonymousConnector(pdpUrl,
                                       CommunicationType.XML_SOAP,
                                       null,
                                       false);
                       }
                       catch (AikException ex) {
                               Console.WriteLine(ex.Message);
                               return;
                       }

                       // Creating the request object and adding attributes to the request
                       req = connector.createRequest();

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

                       // Current time
                       req.addElement(
                               AttributeCategory.environment,
                               EnvironmentAttributes.current_time,
                               AttributeDataType.time,
                               PdpConnector.formatLocalTimeForXml(DateTime.Now));

                       // Send the request to the PDP
                       try {
                               res = connector.evaluate(req);

                       }
                       catch (AikException ex) {
                               Console.WriteLine(ex.Message);
                               return;
```

```
                        }

                        if (res.result == Result.permit) {
                                Console.WriteLine("Permit");
                        }
                        else {
                                Console.WriteLine("Not Permit.\r\n" + res.ToString());
                        }
                }
        }
}
```

This example is the same as the above except the signature verification is enabled.

```
using System;
using PdpLiaison;
using PdpLiaison.Exceptions;
using System.Security.Cryptography.X509Certificates;

namespace SampleCodes
{
        class SimpleAuthzTest
        {
                static void Main(string[] args)
                {
                        PdpConnector connector;
                        AuthorizationRequest req;
                        AuthorizationResponse res;
                        X509Store trustStore;
                        X509ChainPolicy chainPolicy;
                        Uri pdpUrl = new Uri("http://localhost:6009");

                        // Select the trusted certificate store.
                        // This certificate store will be searched to find the certificate
                        // that the server has used to sign the responses when only the that
                        // name of the signing certificate is included in the signature.
                        trustStore = new X509Store(StoreName.AddressBook,
                        StoreLocation.CurrentUser);

                        // Create and configure an X509ChainPolicy object to be used by
                        // the connector as the certificate chain policy.
                        chainPolicy = new X509ChainPolicy();
                        chainPolicy.VerificationFlags = X509VerificationFlags.
                        IgnoreEndRevocationUnknown;

                        // Create AnonymousConnector object
                        try {
                                connector = new AnonymousConnector(pdpUrl,
                                        CommunicationType.XML_SOAP,
                                        trustStore,
                                        true,
                                        chainPolicy);
                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        // Creating the request object and adding attributes to the request
                        req = connector.createRequest();

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

                        // Send the request to the PDP
                        try {
                                res = connector.evaluate(req);

                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        if (res.result == Result.permit) {
                                Console.WriteLine("Permit");
                        }
                        else {
                                Console.WriteLine("Not Permit.\r\n" + res.ToString());
                        }
                }
        }
}
```

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 (Uri PdpUrl, CommunicationType ct, X509Certificate2 signingCert, CertificateInclusion certInclusion, X509Store trustStore, boolean verifySignature, X509ChainPolicy chainPolicy)`

   The method has seven arguments:

   - `PdpUrl` – the SOAP address (including the port number) and protocol used. For example, `new Uri(“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 .NET AIK uses the windows certificate store 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.

- `signingCert` – the certificate used to sign 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`.
- `trustStore` – the certificate store that is searched by the AIK to find the certificate that the server used to sign the responses when only the subject name of the signing certificate is included in the signature.
- `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`.
- `chainPolicy` – by default the AIK builds a simple chain for the certificates and applies the base policy to that chain. However, if this optional argument is specified, then the value given will be applied to the chain.

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

   `CreateRequest()`

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

   ```
   addElement(string category, string attribute,
   AttributeDataType attributeDataType, 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 xpathExpression.

   - `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

```
using System;
using PdpLiaison;
using PdpLiaison.Exceptions;
using System.Security.Cryptography.X509Certificates;

namespace SampleCodes
{
        class SimpleAuthzTest
        {
                static void Main(string[] args)
                {
                        PdpConnector connector;
                        AuthorizationRequest req;
                        AuthorizationResponse res;
                        Uri pdpUrl = new Uri("http://localhost:6009");
                        X509Certificate2 signingCert;
                        X509Store trustStore;
                        X509ChainPolicy chainPolicy;

                        // Select the trusted certificate store.
                        // This certificate store will be searched to find the certificate
                        // that the server has used to sign the responses when only the
                        // subject name of the signing certificate is included in the signature.
                        trustStore = new X509Store(StoreName.AddressBook,
                        StoreLocation.CurrentUser);

                        // Create and configure an X509ChainPolicy object to be used by
                        // the connector as the certificate chain policy.
                        chainPolicy = new X509ChainPolicy();
                        chainPolicy.VerificationFlags = X509VerificationFlags.
                        IgnoreEndRevocationUnknown;

                        // Load the signing certificate
                        signingCert = new X509Certificate2(@"C:\test\certificate.p12");

                        // Create AnonymousConnector object
                        try {
                                connector = new XmlSigningConnector(pdpUrl,
                                        CommunicationType.XML_SOAP,
                                        signingCert,
                                        CertificateInclusion.certificateChain,
                                        trustStore,
                                        true,
                                        chainPolicy);
                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        // Creating the request object and adding attributes to the request
                        req = connector.createRequest();

                        // User Id: smith
                        req.addElement(
                                AttributeCategory.access_subject,
                                SubjectAttributes.subject_id,
                                AttributeDataType._string,
                                "smith");
                        // Send the request to the PDP
                        try {
                                res = connector.evaluate(req);

                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        if (res.result == Result.permit) {
                                Console.WriteLine("Permit");
                        }
                        else {
                                Console.WriteLine("Not Permit.\r\n" + res.ToString());
                        }
                }
        }
}
```

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`:

   ```
   ClentSslConnector(Uri PdpUrl, X509Certificate2 clientCert,
   X509Store trustStore, boolean verifySignature,
   CommunicationType ct, X509ChainPolicy chainPolicy)
   ```

   The method has six arguments:

   - `PdpUrl` – the SOAP address (including the port number) and protocol used. For example, `new Uri("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 .NET AIK uses the windows certificate store as specified below. The HTTPS Address of the ViewDS server is configured using the ViewDS Management Agent.

   - `clientCert` – the certificate that the AIK uses to establish an SSL connection to the PDP.
   - `trustStore` – the certificate store that is searched by the AIK to find the certificate that the server used to sign the responses when only the subject name of the signing certificate is included in the signature.

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

- `chainPolicy` – by default the AIK builds a simple chain for the certificates and applies the base policy to that chain. However, if this optional argument is specified, then the value given will be applied to the chain.

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

   ```
   CreateRequest()
   ```

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

   ```
   addElement(string category, string attribute,
   AttributeDataType attributeDataType, 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 xpathExpression.

   - `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(AuthorizationRequestreq)
   ```

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

Example code

```
using System;
using PdpLiaison;
using PdpLiaison.Exceptions;
using System.Security.Cryptography.X509Certificates;

namespace SampleCodes
{
       class SimpleAuthzTest
       {
               static void Main(string[] args)
               {
                       PdpConnector connector;
                       AuthorizationRequest req;
                       AuthorizationResponse res;
                       Uri pdpUrl = new Uri("https://localhost:6010");
                       X509Certificate2 clientCert;

                       // Load the signing certificate
                       clientCert = new X509Certificate2(@"C:\test\certificate.p12");

                       // Create AnonymousConnector object
                       try {
                               connector = new ClientSslConnector(pdpUrl,
                                       clientCert,
                                       null,
                                       false,
                                       CommunicationType.XML_SOAP);
                       }
                       catch (AikException ex) {
                               Console.WriteLine(ex.Message);
                               return;
                       }

                       // Creating the request object and adding attributes to the request
                       req = connector.createRequest();

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

                       // Send the request to the PDP
                       try {
                               res = connector.evaluate(req);
                       }
                       catch (AikException ex) {
                               Console.WriteLine(ex.Message);
                               return;
                       }

                       if (res.result == Result.permit) {
                               Console.WriteLine("Permit");
                       }
                       else {
                               Console.WriteLine("Not Permit.\r\n" + res.ToString());
                       }
               }
       }
}
```

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(string obligationIdentifier)`

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 .NET 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).

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Mail;
using PdpLiaison;
using PdpLiaison.Exceptions;
using System.Security.Cryptography.X509Certificates;

namespace SampleCodes
{
       class SimpleAuthzTest
       {
               static void Main(string[] args)
               {
                       PdpConnector connector;
                       AuthorizationRequest req;
```

```
                        AuthorizationResponse res;
                        Uri pdpUrl = new Uri("http://localhost:6009");

                        // Create AnonymousConnector object
                        try {
                                connector = new AnonymousConnector(pdpUrl,
                                CommunicationType.XML_SOAP,
                                null,
                                false);
                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        // Register obligations
                        connector.registerObligation("foo:bar:email");

                        // Creating the request object and adding attributes to the request
                        req = connector.createRequest();

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

                        // Send the request to the PDP
                        try {
                                res = connector.evaluate(req);

                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        switch (res.result) {
                                case Result.deny:
                                        Console.WriteLine("Deny");
                                        break;
                                case Result.denyWithObligations:
                                        Console.WriteLine("Deny");
                                        fullfillObligations(res.obligations);
                                        break;
                                case Result.denyDueToUnrecognizedObligations:
                                        Console.WriteLine("Deny");
                                        break;
                                case Result.denyUnlessAllObligationsSatisfied:
                                        if (fullfillObligations(res.obligations)) {
                                                Console.WriteLine("Permit");
                                        }
                                        else {
                                                Console.WriteLine("Deny");
                                        }
                                        break;
                                case Result.permit:
                                        Console.WriteLine("Permit");
                                        break;
                        }
                }

                private static bool fullfillObligations
                (System.Collections.Generic.List<Obligation> obligations)
                {
                        foreach (Obligation ob in obligations) {
                                if (ob.id == "foo:bar:email") {
                                        if (!sendEmail(ob)) {
                                                return false;
                                        }
                                }
                                else {
                                        return false;
                                }
                        }

                        return true;
                }

                private static bool sendEmail(Obligation ob)
                {
                        MailMessage message = new MailMessage();
                        List<string> recipientAddresses = new List<string>();

                        message.From = new MailAddress("testpep@somedomain.com", "TEST-PEP");

                        foreach (AttributeAssignment aa in ob.attributes) {
                                string attId = "";
                                string attCat = "";
                                string attVal = "";

                                try { attId = aa.attributeId.ToLower(); }
                                catch { }
                                try { attCat = aa.categoryId.ToLower(); }
                                catch { }
                                try { attVal = aa.attributeValue; }
                                catch { }

                                if (attCat == "email" && attId == "recipientaddress") {
                                        recipientAddresses.Add(attVal);
                                }
                                if (attCat == "email" && attId == "subject") {
                                        message.Subject = attVal;
                                }
                                if (attCat == "email" && attId == "body") {
                                        message.Body = attVal;
                                }
                        }

                        if (recipientAddresses.Count < 1) {
                                return false;
                        }
                        foreach (string recipient in recipientAddresses) {
                                message.To.Add(recipient);
                        }

                        try {
                                SmtpClient smtp = new SmtpClient {
                                        Host = "smtp.somedomain.com",
                                        Port = 587,
                                        EnableSsl = true,
                                        DeliveryMethod = SmtpDeliveryMethod.Network,
                                        UseDefaultCredentials = false,
                                        Credentials = new NetworkCredential
                                        ("testpep@somedomain.com", "password")
                                };
                                smtp.Send(message);
                        }
                        catch {
                                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

This example shows how to use MultiRequest to send an authorisation request for the user with the role MANAGER who is trying to access two files REPORT A and REPORT B:

```
using System;
using PdpLiaison;
using PdpLiaison.Exceptions;

namespace SampleCodes
{
        class SimpleAuthzTest
        {
                static void Main(string[] args)
                {
                        PdpConnector connector;
                        AuthorizationRequest req1, req2;
                        MultiRequest mulReq;
                        MultiResponse mulRes;
                        Uri pdpUrl = new Uri("http://localhost:6009");

                        // Create AnonymousConnector object
                        try {
                                connector = new AnonymousConnector(pdpUrl,
                                CommunicationType.XML_SOAP,
                                null,
                                false);
                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        // Creating the request objects and adding attributes to them
                        req1 = connector.createRequest();
                        req2 = 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 = new MultiRequest(false);
                        mulReq.addRequest(req1);
                        mulReq.addRequest(req2);

                        // Send the request to the PDP
                        try {
                                mulRes = connector.evaluate(mulReq);
                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        if (mulRes.getResponseForRequest(req1).result == Result.permit &&
                                mulRes.getResponseForRequest(req2).result == Result.permit) {
                                System.Console.WriteLine("Permit");
                        }
                        else {
                                System.Console.WriteLine("Deny");
                        }
                }
        }
}
```

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 server. See the Enable tracing topic in the ViewDS Access Sentinel Installation and Reference Guide for full 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.

```
using System;
using PdpLiaison;
using PdpLiaison.Exceptions;

namespace SampleCodes
{
	class SimpleAuthzTest
	{
		static void Main(string[] args)
		{
			PdpConnector connector;
			AuthorizationRequest req;
			AuthorizationResponse res;
			Uri pdpUrl = new Uri("http://localhost:6009");

			// Create AnonymousConnector object
			try {
				connector = new AnonymousConnector(pdpUrl,
				CommunicationType.XML_SOAP,
				null,
				false);
			}
			catch (AikException ex) {
				Console.WriteLine(ex.Message);
				return;
			}

			// Creating the request object with the trace flag set to true.
			req = connector.createRequest(true);
```

```
                        req.addElement(
                                AttributeCategory.resource,
                                ResourceAttributes.resource_id,
                                AttributeDataType._string,
                                "REPORT A");

                        // Send the request to the PDP
                        try {
                                res = connector.evaluate(req);
                        }
                        catch (AikException ex) {
                                Console.WriteLine(ex.Message);
                                return;
                        }

                        if (res.result == Result.permit) {
                                System.Console.WriteLine("Permit");
                        }
                        else {
                                System.Console.WriteLine("Deny");
                        }

                        System.Console.WriteLine(res.traceInfo);
                }
        }
}
```

Appendix A: AIK structure

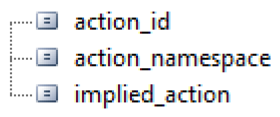
The .NET AIK is a class library developed in C# .NET and compiled into two files:

- `PdpLiaison.dll` (binaries)
- `PdpLiaison.xml` (documentation)

The library exposes the following members.

ActionAttributes

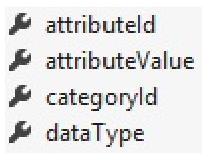
A list of constant strings that represent identifiers of the standard action attributes specified in the XACML core specification.



action_id
action_namespace
implied_action

AttributeAssignment

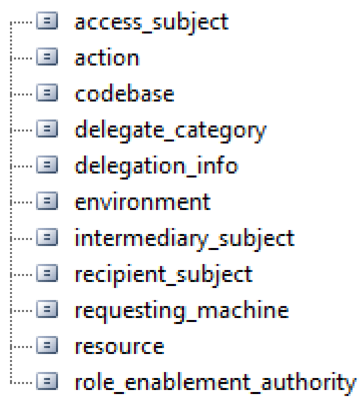
The attributes of an obligation or an advice are objects of this type. The public members of this class are shown below.



attributeId
attributeValue
categoryId
dataType

AttributeCategory

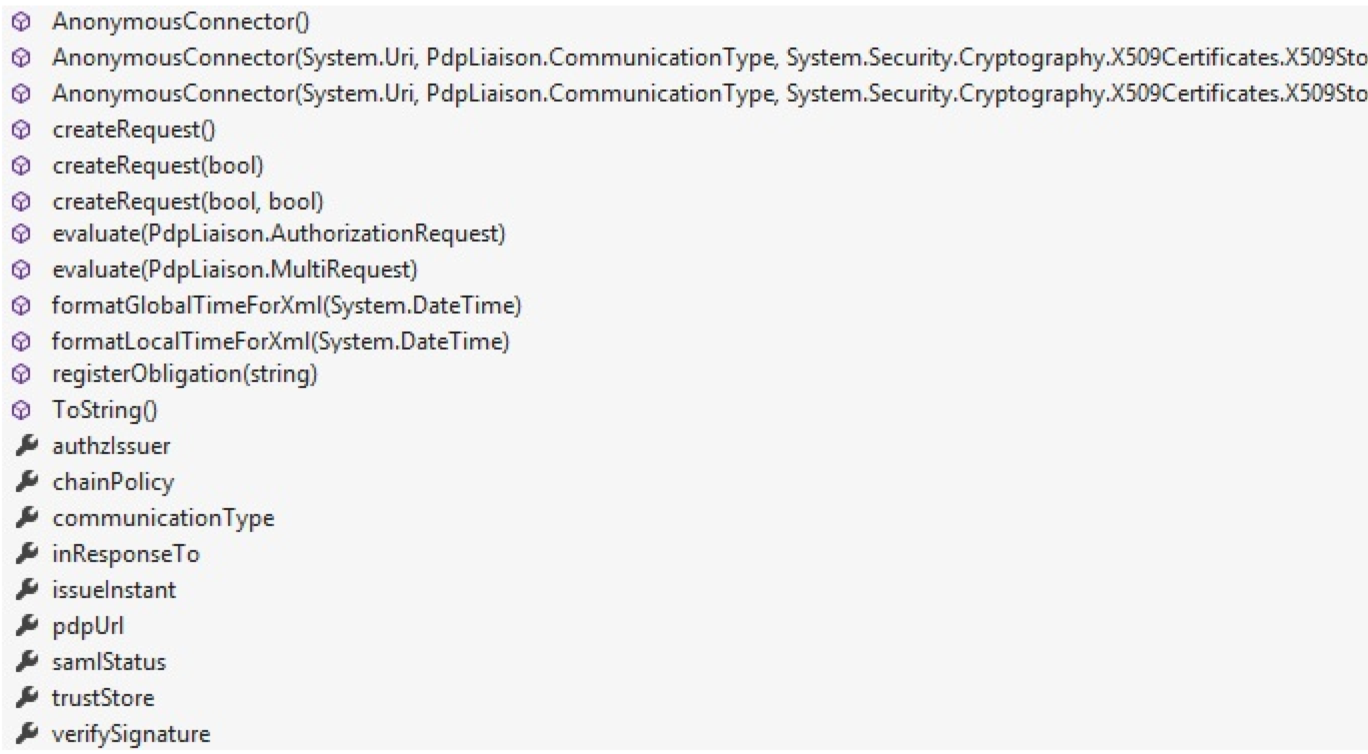
A list of constant strings that represent identifiers of the standard attribute categories specified in the XACML core specification.



access_subject
action
codebase
delegate_category
delegation_info
environment
intermediary_subject
recipient_subject
requesting_machine
resource
role_enablement_authority

AnonymousConnector

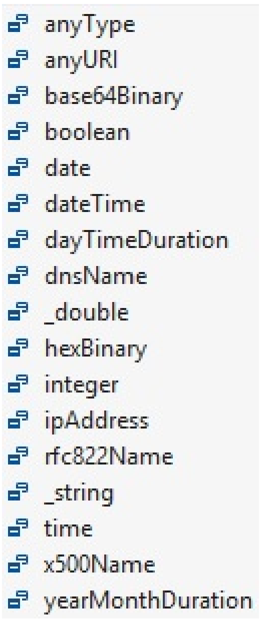
This class is used to configure an anonymous connection to a PDP. An object of this type should be instantiated at the beginning to be used for sending anonymous authorization requests to the PDP. The public members of this class are shown below.



AnonymousConnector()
AnonymousConnector(System.Uri, PdpLiaison.CommunicationType, System.Security.Cryptography.X509Certificates.X509Sto
AnonymousConnector(System.Uri, PdpLiaison.CommunicationType, System.Security.Cryptography.X509Certificates.X509Sto
createRequest()
createRequest(bool)
createRequest(bool, bool)
evaluate(PdpLiaison.AuthorizationRequest)
evaluate(PdpLiaison.MultiRequest)
formatGlobalTimeForXml(System.DateTime)
formatLocalTimeForXml(System.DateTime)
registerObligation(string)
ToString()
authzIssuer
chainPolicy
communicationType
inResponseTo
issueInstant
pdpUrl
samlStatus
trustStore
verifySignature

AttributeDataType

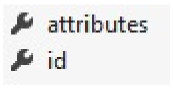
An enumerative list of standard data types specified in the XACML core specification.



anyType
anyURI
base64Binary
boolean
date
dateTime
dayTimeDuration
dnsName
_double
hexBinary
integer
ipAddress
rfc822Name
_string
time
x500Name
yearMonthDuration

Advice

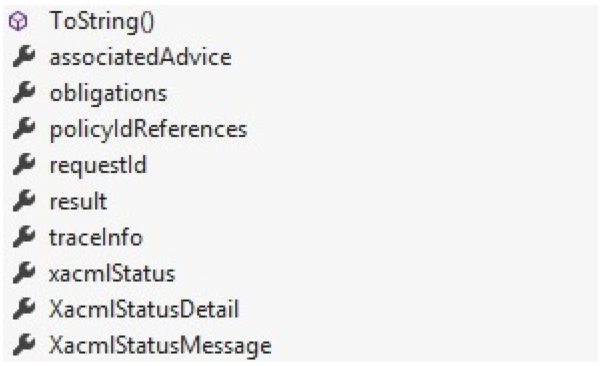
The advice to be fulfilled is returned as objects of this type which are included in the authorization response. The public members of this class are shown below.



attributes
id

AuthorizationResponse

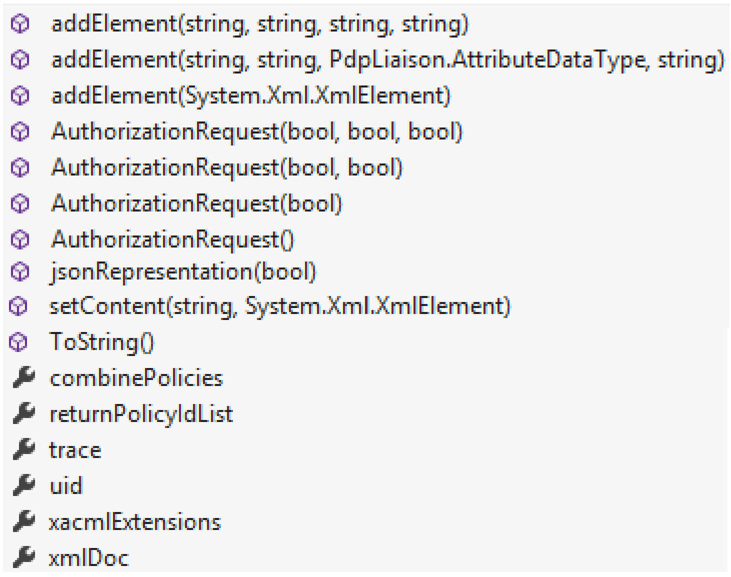
The results of an XACML authorization decision request are returned as objects of this type. The public members of this class are shown below.



ToString()
associatedAdvice
obligations
policyIdReferences
requestId
result
traceInfo
xacmlStatus
XacmlStatusDetail
XacmlStatusMessage

AuthorizationRequest

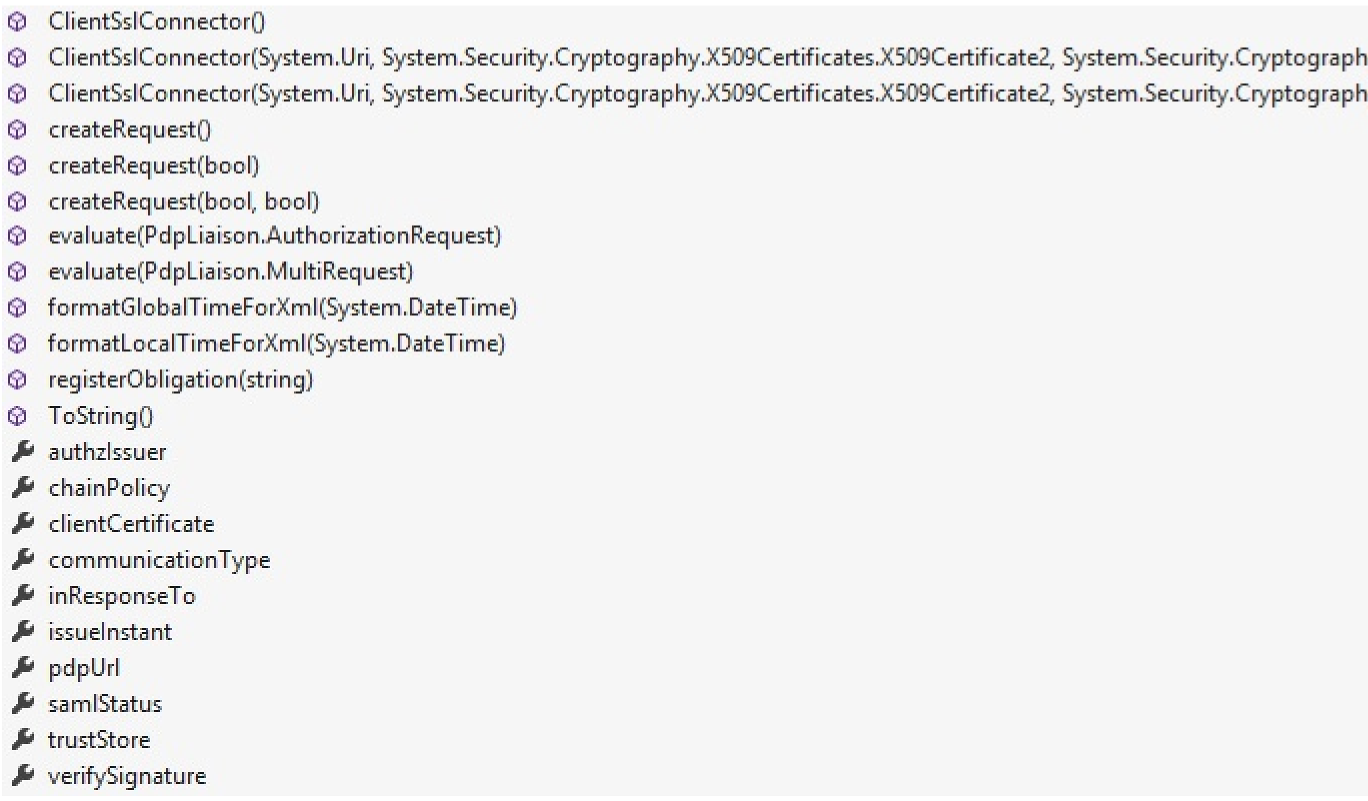
Objects of this type should be instantiated for each XACML authorization decision request to be sent to the PDP. The public members of this class are shown below.



addElement(string, string, string, string)
addElement(string, string, PdpLiaison.AttributeDataType, string)
addElement(System.Xml.XmlElement)
AuthorizationRequest(bool, bool, bool)
AuthorizationRequest(bool, bool)
AuthorizationRequest(bool)
AuthorizationRequest()
jsonRepresentation(bool)
setContent(string, System.Xml.XmlElement)
ToString()
combinePolicies
returnPolicyIdList
trace
uid
xacmlExtensions
xmlDoc

ClientSslConnector

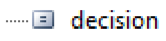
This class is used to configure a secure HTTPS connection to a PDP. An object of this type should be instantiated at the beginning to be used to send authorisation requests to the PDP via a secure HTTPS connection. The public members of this class are shown below.



ClientSslConnector()
ClientSslConnector(System.Uri, System.Security.Cryptography.X509Certificates.X509Certificate2, System.Security.Cryptograph
ClientSslConnector(System.Uri, System.Security.Cryptography.X509Certificates.X509Certificate2, System.Security.Cryptograph
createRequest()
createRequest(bool)
createRequest(bool, bool)
evaluate(PdpLiaison.AuthorizationRequest)
evaluate(PdpLiaison.MultiRequest)
formatGlobalTimeForXml(System.DateTime)
formatLocalTimeForXml(System.DateTime)
registerObligation(string)
ToString()
authzIssuer
chainPolicy
clientCertificate
communicationType
inResponseTo
issueInstant
pdpUrl
samlStatus
trustStore
verifySignature

DelegationInfoAttributes

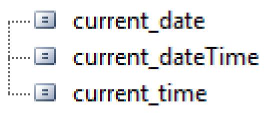
A list of constant strings that represent identifiers of the standard delegationInfo attributes specified in the XACML core specification.



decision

EnvironmentAttributes

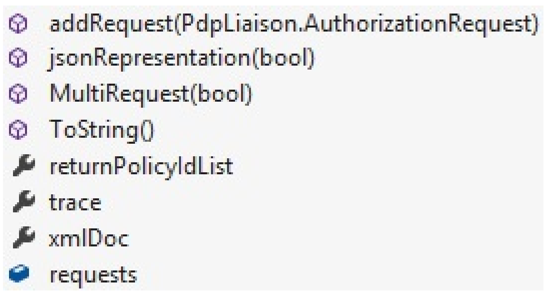
A list of constant strings that represent identifiers of the standard environment attributes specified in the XACML core specification.



current_date
current_dateTime
current_time

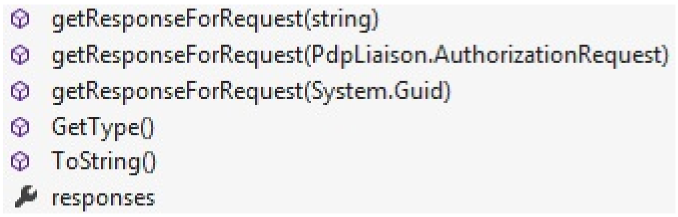
MultiRequest

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. The public members of this class are shown below:


addRequest(PdpLiaison.AuthorizationRequest)
jsonRepresentation(bool)
MultiRequest(bool)
ToString()
returnPolicyIdList
trace
xmlDoc
requests

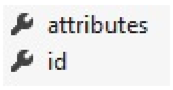
MultiResponse

The results of a MultiRequest authorization decision request are returned as objects of this type. The public members of this class are shown below:


getResponseForRequest(string)
getResponseForRequest(PdpLiaison.AuthorizationRequest)
getResponseForRequest(System.Guid)
GetType()
ToString()
responses

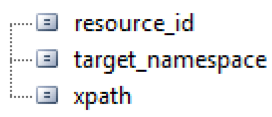
Obligation

The obligations to be fulfilled are returned as objects of this type which are included in the authorization response. The public members of this class are shown below.


attributes
id

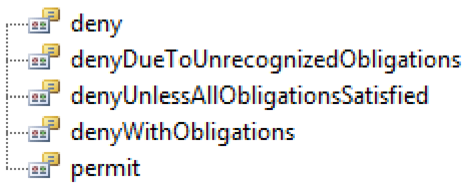
ResourceAttributes

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


resource_id
target_namespace
xpath

Result

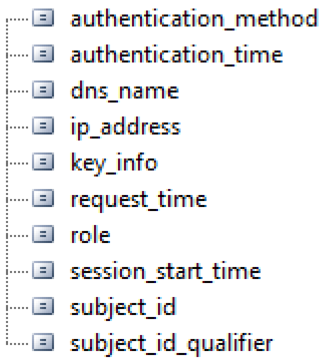
An enumerative list of authorization results specified in the AIK.



- deny
- denyDueToUnrecognizedObligations
- denyUnlessAllObligationsSatisfied
- denyWithObligations
- permit

SubjectAttributes

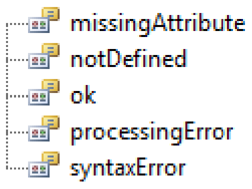
A list of constant strings that represent identifiers of the standard subject attributes specified in the XACML core specification.



- authentication_method
- authentication_time
- dns_name
- ip_address
- key_info
- request_time
- role
- session_start_time
- subject_id
- subject_id_qualifier

XacmlStatus

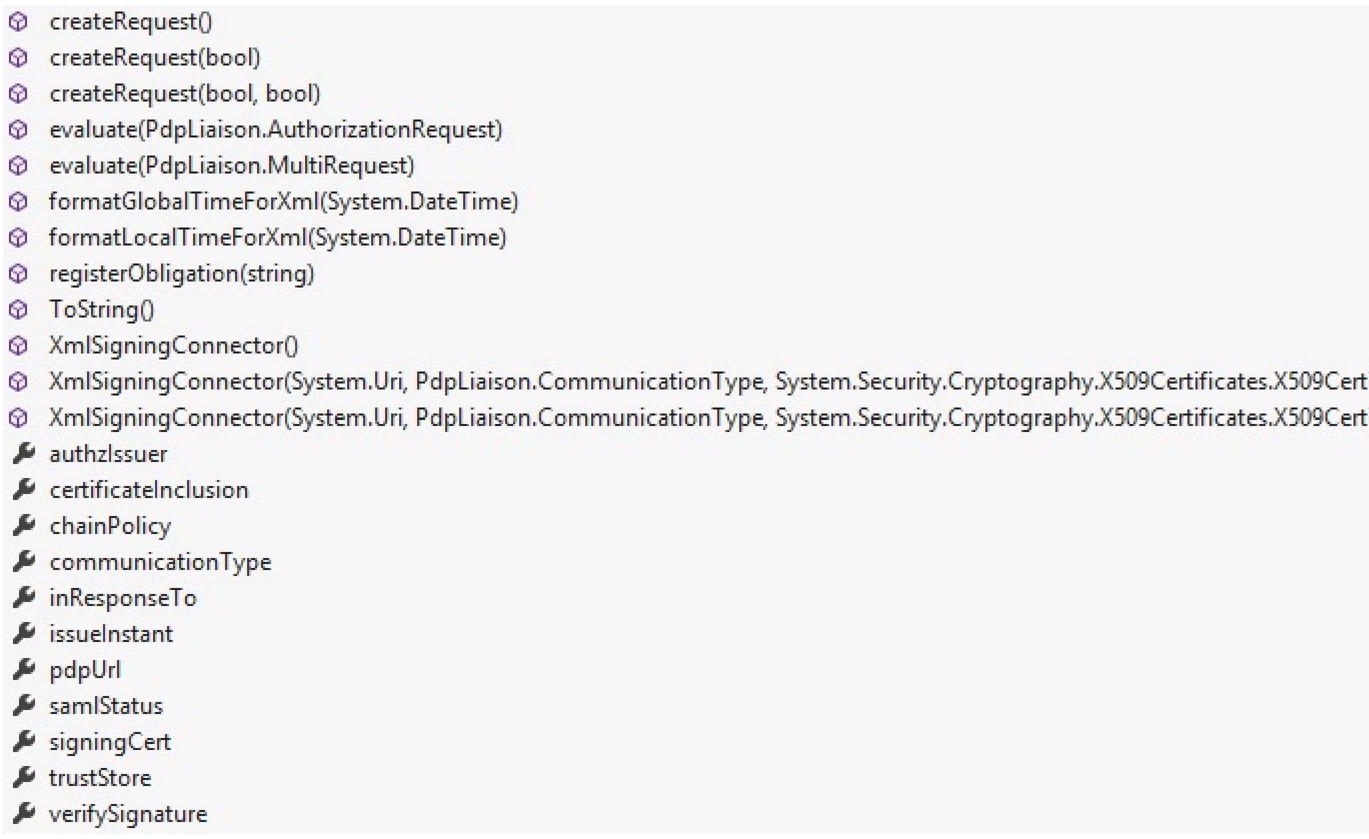
An enumerative list of standard XACML status types.



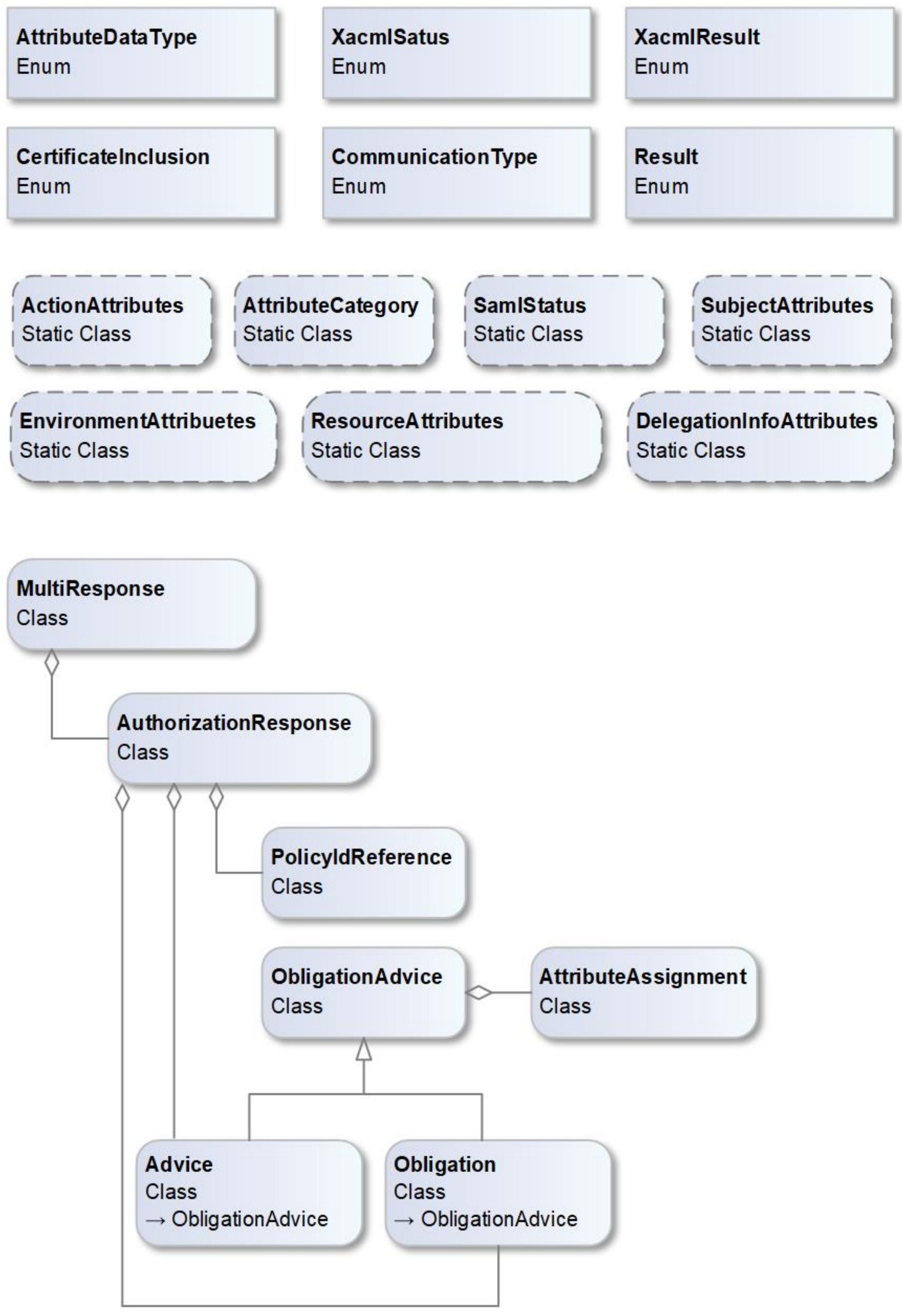
- missingAttribute
- notDefined
- ok
- processingError
- syntaxError

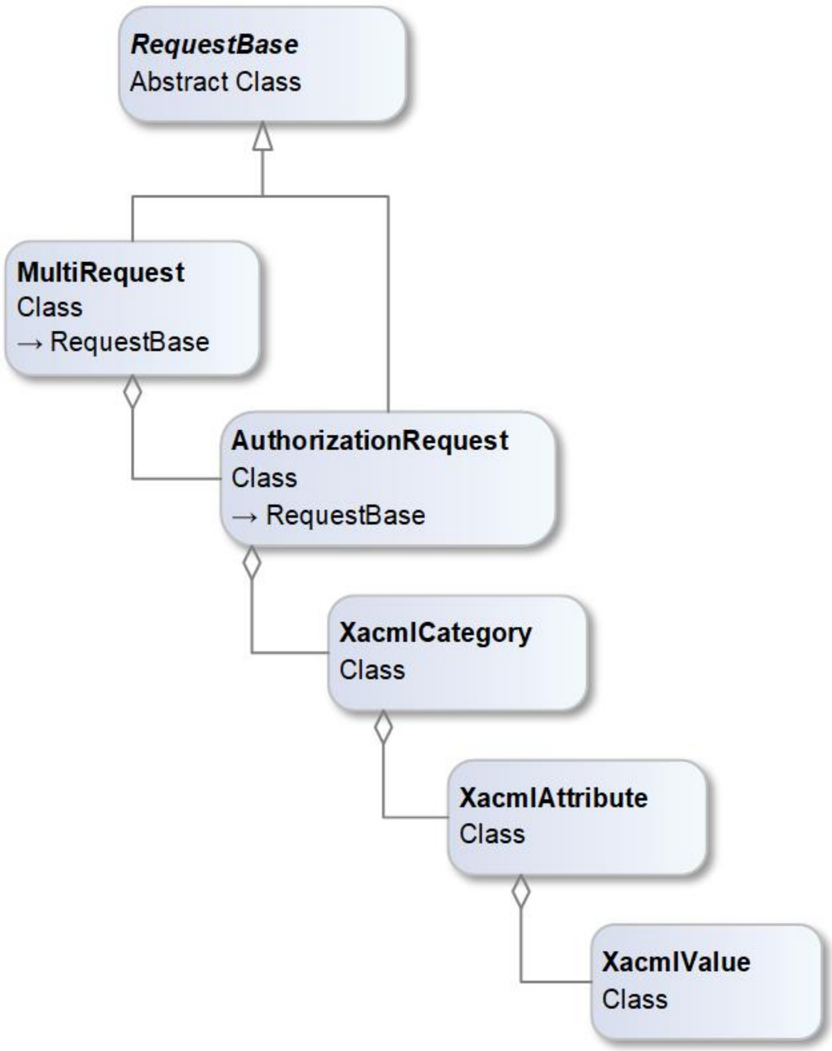
XmlSigningConnector

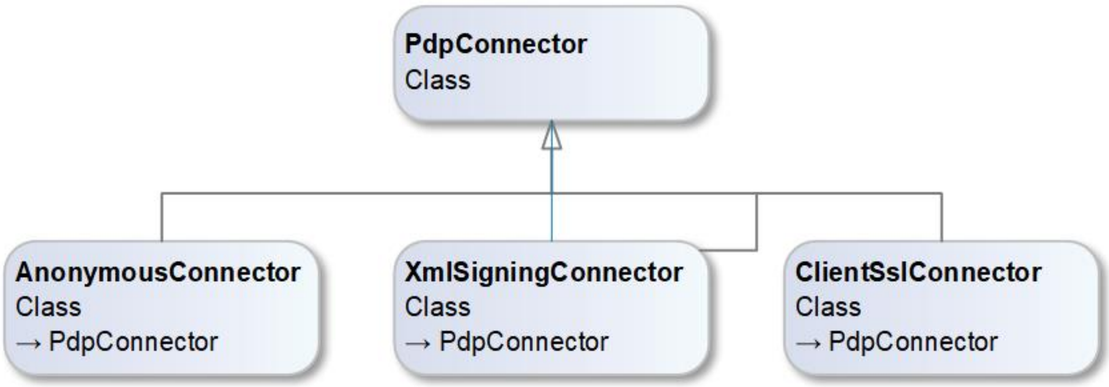
This class is used to configure a connection to a PDP through which signed authorisation request can be sent. An object of this type should be instantiated at the beginning to be used to sign authorisation requests using XML digital signatures before sending them to the PDP. The public members of this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


createRequest()
createRequest(bool)
createRequest(bool, bool)
evaluate(PdpLiaison.AuthorizationRequest)
evaluate(PdpLiaison.MultiRequest)
formatGlobalTimeForXml(System.DateTime)
formatLocalTimeForXml(System.DateTime)
registerObligation(string)
ToString()
XmlSigningConnector()
XmlSigningConnector(System.Uri, PdpLiaison.CommunicationType, System.Security.Cryptography.X509Certificates.X509Cert
XmlSigningConnector(System.Uri, PdpLiaison.CommunicationType, System.Security.Cryptography.X509Certificates.X509Cert
authzIssuer
certificateInclusion
chainPolicy
communicationType
inResponseTo
issueInstant
pdpUrl
samlStatus
signingCert
trustStore
verifySignature

AIK Class Diagram



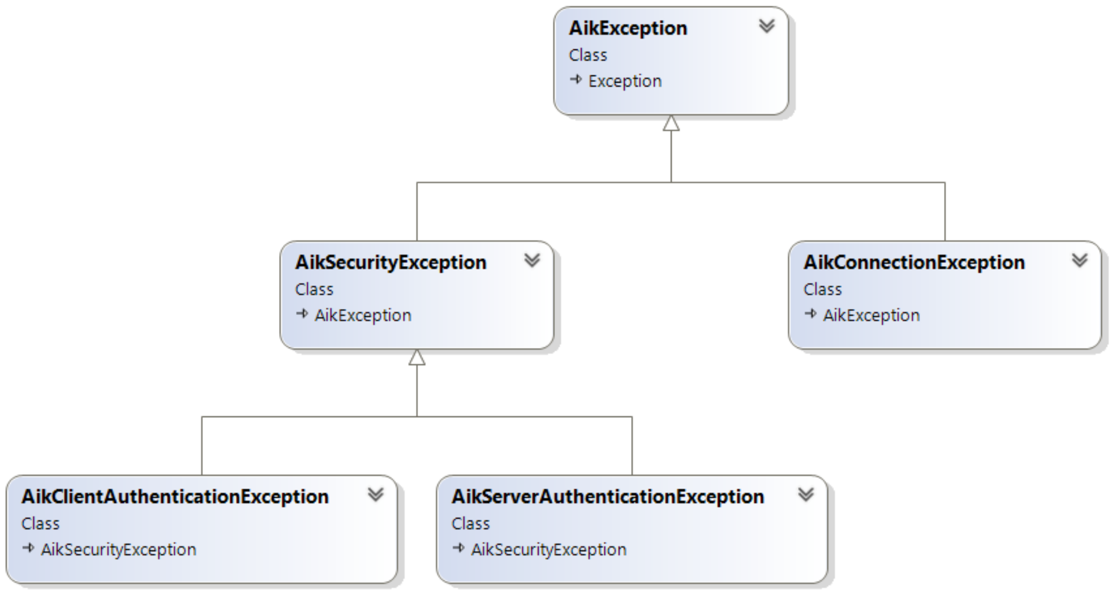

RequestBase
Abstract Class
MultiRequest
Class
→ RequestBase
AuthorizationRequest
Class
→ RequestBase
XacmlCategory
Class
XacmlAttribute
Class
XacmlValue
Class


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: " + inner-WebException.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:

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:

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."