Supplement to Authorization API, v0.8
July 16, 2009

Contributors:

Rich Levinson Oracle Corporation
Anil Tappetla Cisco Systems, Inc.

1 Introduction

This document includes supplementary materials for the proposed Java Az API. The Az API is primarily intended to allow Authorization decisions to be made by an XACML-compliant PDP, but could also be used to access other types of PDPs that are capable of making use of the inputs provided. It is intended primarily for use by infrastructure components which have be configured to populate the request context with Attributes and/or invoke Authorization decisions, prior to the execution of specified Methods, however it also may be used by applications which have specialized requirements or need to provide inputs not available to the infrastructure.

There have been some concerns expressed about the proposed Az API being too low-level, missing convenience features, certain constants and so on. The main point here is that the API is not meant to be the only Az API exposed to customers; there may be many higher-level layers built using the API that would provide simpler wrapper methods and constants, specific to particular frameworks or containers, essentially the analog to "checkPermission" or "isAccessAllowed". These would hide some of the low-level details (lists of attributes, categories of subjects etc.) that the current API exposes.

2 Examples

2.1 Basic

This example demonstrates how an AzService is created, then used to create and populate an AzRequestContext, which is submitted via an AzService.decide() call to get an authorization decision returned in an AzResponseContext.

The example shows how different AzCategoryIds are used to differentiate collections of AzAttributes collected in AzEntity elements, where an AzEntity object represents and corresponds to a collection of attributes in a XACML category, such as “access-subject”, “action”, “resource”, and “environment”.

In this example “CONTAINER” represents an application server, which is used in this example as a XACML Issuer of “environment attributes”. Similarly “APPLICATION” represents an enterprise application that is the source and XACML Issuer of both “resource attributes” and the application-defined “action attributes” that specify operations that can be performed on the application resources.

Note: the term “demo” is used in the comments that follow to indicate the api call is simply being demonstrated, but that no constructive advancement of a functional nature is implied at that point.

```
[a01]     package org.example.azapi.test;
[a02]
[a03]     import org.example.azapi.*;
[a04]
[a05]     // Sample code fragment using AzAPI (based on TestAzAPI)
[a06]
[a07]     // Sample code fragment using AzAPI (based on TestAzAPI)
[a08]
[a09]     // variable defns to provide "context" for test program:
[a10]          // defn: "container": manages "environment" and "issues" env attributes
[a11]          public final static String CONTAINER = "orcl-weblogic";   // [a03]
[a12]          // defn: "application": manages "resources" and "issues" resource attrs
[a13]          public final static String APPLICATION = "hr-application-01"; // [a04]

[a14]          // test user; normally would be obtained from session info
[a15]          public final static String SAMPLE_SESSION_USER_NAME = "Joe User";
[a16]          public final static String SAMPLE_SESSION_AUTH_METHOD = "password";
[a17]
```

```
[a18]
[a19]         // Get a reference to the authorization service:
[a20]         AzService azHandle = AzServiceFactory.getAzService();
[a21]
[a22]         // Create an AzRequestContext:
[a23]         AzRequestContext azReqCtx = azHandle.createAzRequestContext();
[a24]
[a25]         // Create an AzEnvironment entity and add an env attr:
[a26]         AzEntity<AzCategoryIdEnvironment> azEnv =
[a27]               azReqCtx.createAzEntity(
[a28]                      AzCategoryIdEnvironment.AZ_CATEGORY_ID_ENVIRONMENT);
[a29]
[a30]
[a31]
[a32]         // Create and obtain a reference to an environment attribute,
[a33]         // automatically added to AzEnv, create an attribute value
[a34]         // of XACML DataType #dateTime, automatically assigned to
[a35]         // to the created attribute in one Java statement.
[a36]
[a37]
[a38]         AzAttribute<AzCategoryIdEnvironment> azEnvAttr =
[a39]               azEnv.createAzAttribute(
[a40]                      CONTAINER,
[a41]                      X_ATTR_ENV_CURRENT_DATE_TIME,
[a42]                      azEnv.createAzAttributeValue(
[a43]                             AzDataTypeIdDateTime.AZ_DATATYPE_ID_DATETIME,
[a44]                             azEnv.createAzDataDateTime(new Date(),0,0,0)));
[a45]
[a46]
[a47]         // Add environment object to AzRequestContext, which is for demo
[a48]         // how replacement would work, but in this case leaves unchanged.
[a49]         azReqCtx.addAzEntity(azEnv);
[a50]
[a51]         // Demo getting an environment attribute from the AzRequestContext:
[a52]         AzAttribute<AzCategoryIdEnvironment> azEnvCurrDateTime =
[a53]               azReqCtx.getAzEnvironment().
[a54]                     getAttributeByAttribId(X_ATTR_ENV_CURRENT_DATE_TIME);

[a55]
[a56]         // Get a Subject AzCategoryId and AzEntity for access-subject attributes
[a57]         AzCategoryIdSubjectAccess azSubjCat =
[a58]               AzCategoryIdSubjectAccess.AZ_CATEGORY_ID_SUBJECT_ACCESS;
[a59]
              AzEntity<AzCategoryIdSubjectAccess> accSubj =
[a60]                      azReqCtx.createAzEntity(azSubjCat);
[a61]
[a62]         // Two steps: 1.create an attributeValue,
[a63]         AzAttributeValueString azAttrValueSubjId =
[a64]               accSubj.createAzAttributeValue(
[a65]                      AzDataTypeIdString.AZ_DATATYPE_ID_STRING,
[a66]                      SAMPLE_SESSION_USER_NAME);
[a67]
```

```
[a68]
[a69]         // 2. create and add attribute along with attribute value to collection.
[a70]         //    Note: reference is returned as demo of its potential for use.
[a71]         AzAttribute<AzCategoryIdSubjectAccess> azAttrSubjectId =
[a72]                accSubj.createAzAttribute(
[a73]                               CONTAINER,
[a74]                               AzXacmlStrings.X_ATTR_SUBJECT_ID ,
[a75]                               azAttrValueSubjId);
[a76]
[a77]         // One step to create a subject attribute and add it to collection:
[a78]          // Note: both this and above attributes are considered as “issued”
[a79]          // by the application server, “CONTAINER”
[a80]         accSubj.createAzAttribute(
[a81]                CONTAINER,
[a82]                AzXacmlStrings.X_ATTR_SUBJECT_AUTHN_LOC_AUTHENTICATION_METHOD,
[a83]                accSubj.createAzAttributeValue(
[a84]                       AzDataTypeIdString.AZ_DATATYPE_ID_STRING,
[a85]                       SAMPLE_SESSION_AUTH_METHOD));
[a86]
[a87]         // Add the Subject to the AzRequestContext: Note: this is demo of
[a88]          // adding an AzEntity to AzRequestContext, but since already
[a89]          // present, this statement leaves AzRequestContext unchanged
[a90]         azReqCtx.addAzEntity(accSubj);  // [a26]
[a91]
[a92]
[a93]         // Use AzCategoryIdResource to get typed AzEntity
[a94]         AzEntity<AzCategoryIdResource> azResource =
[a95]                azReqCtx.createAzEntity(
[a96]                       AzCategoryIdResource.AZ_CATEGORY_ID_RESOURCE);
[a97]
[a98]          // create a resource-id attribute, w “APPLICATION” as XACML Issuer
[a99]          azResource.createAzAttribute(
[a100]               APPLICATION,
[a101]               AzXacmlStrings.X_ATTR_RESOURCE_ID,
[a102]               azResource.createAzAttributeValue(
[a103]                      AzDataTypeIdString.AZ_DATATYPE_ID_STRING,
[a104]                      "file:C\\toplevel"));
[a105]
[a106]
[a107]        // Create an AzCategoryIdAction AzEntity object
[a108]        AzEntity<AzCategoryIdAction> azAction =
[a109]               azReqCtx.createAzEntity(
[a110]                      AzCategoryIdAction.AZ_CATEGORY_ID_ACTION);
[a111]
[a112]
[a113]        // add an AzCategoryIdAction attribute to the AzEntity
[a114]        azAction.createAzAttribute(APPLICATION,                    // [a38]
[a115]               AzXacmlStrings.X_ATTR_ACTION_ID,
[a116]               azActionA01.createAzAttributeValue(
[a117]               AzDataTypeIdString.AZ_DATATYPE_ID_STRING, "Read"));
[a118]
[a119]
```

```
[a120]    // Add resource-action pairs to the request context
[a121]    AzResourceActionAssociation assoc =
[a122]     azReqCtx.createAndAddResourceActionAssociation(azResource, azAction);
[a123]
[a124]    // its time to decide !
[a125]    AzResponseContext azRspCtx = azHandle.decide(azReqCtx);
[a126]    AzResult azRes = azRspCtx.getResult(assoc);
[a127]
[a128]
[a129]        switch (azRes.getAzDecision()){
[a130]              case AZ_PERMIT:
[a131]                     // processing steps
[a132]                      …
[a133]                     break;
[a134]               case AZ_DENY:
[a135]                     // processing
[a136]       }
[a137]       …
[a138]
```

2.2 Extended

This example extends the previous example to include several advanced features, including (1) specifying a set of actions (or resources) in a single decide call (2) processing a set of returned decisions (3) use of obligations.

```
[a139]    package org.example.azapi.test;
[a140]
[a141]    import org.example.azapi.*;
[a142]
[a143]    // Sample code fragment using AzAPI (based on TestAzAPI)
[a144]
[a145]    // variable defns to provide "context" for test program:
[a146]        // defn: "container": manages "environment" and "issues" env attributes
[a147]        public final static String CONTAINER = "orcl-weblogic";   // [a03]
[a148]        // defn: "application": manages "resources" and "issues" resource attrs
[a149]        public final static String APPLICATION = "hr-application-01"; // [a04]

[a150]        // test user; normally would be obtained from session info
[a151]        public final static String SAMPLE_SESSION_USER_NAME = "Joe User";
[a152]        public final static String SAMPLE_SESSION_AUTH_METHOD = "password";
[a153]
```

```
[a154]
[a155]        // Get a reference to the authorization service:
[a156]        AzService azHandle = AzServiceFactory.getAzService();
[a157]
[a158]        // Create an AzRequestContext:
[a159]        AzRequestContext azReqCtx = azHandle.createAzRequestContext();
[a160]
[a161]        // Create an AzEnvironment entity and add an env attr:
[a162]        AzEntity<AzCategoryIdEnvironment> azEnv =
[a163]               azReqCtx.createAzEntity(
[a164]                      AzCategoryIdEnvironment.AZ_CATEGORY_ID_ENVIRONMENT);
[a165]
[a166]
[a167]
[a168]
[a169]
[a170]
[a171]        // Create a general purpose attribute value
[a172]        // and add it to AzEnv
[a173]
[a174]        AzAttribute<AzCategoryIdEnvironment> azEnvAttr =    // [a13]
[a175]               azEnv.createAzAttribute(
[a176]                      CONTAINER,
[a177]                      X_ATTR_ENV_CURRENT_DATE_TIME,
[a178]                      azEnv.createAzAttributeValue(
[a179]                             AzDataTypeIdDateTime.AZ_DATATYPE_ID_DATETIME,
[a180]                             azEnv.createAzDataDateTime(new Date(),0,0,0)));
[a181]
[a182]
[a183]        // Add environment object to AzRequestContext
[a184]        azReqCtx.addAzEntity(azEnv);  // [a15]
[a185]
[a186]        // Get and print the attribute from the AzRequestContext:
[a187]        AzAttribute<AzCategoryIdEnvironment> azEnvCurrDateTime =
[a188]               azReqCtx.getAzEnvironment().
[a189]                      getAttributeByAttribId(X_ATTR_ENV_CURRENT_DATE_TIME);
[a190]
[a191]        System.out.println("
[a192]               TestAzAPI.azEnv.idIdentityCounter = " + azEnv.getId());
[a193]
[a194]        // Get a Subject Category and AzEntity for access-subject attributes
[a195]        AzCategoryIdSubjectAccess azSubjCat =
[a196]               AzCategoryIdSubjectAccess.AZ_CATEGORY_ID_SUBJECT_ACCESS;
[a197]
              AzEntity<AzCategoryIdSubjectAccess> accSubj =
[a198]                      azReqCtx.createAzEntity(azSubjCat);   // [a17]
[a199]
[a200]        // Two steps to create a subject attribute and add it to collection:
[a201]        AzAttributeValueString azAttrValueSubjId =
[a202]               accSubj.createAzAttributeValue(
[a203]                      AzDataTypeIdString.AZ_DATATYPE_ID_STRING,
[a204]                      SAMPLE_SESSION_USER_NAME);
```

```
[a205]
[a206]        AzAttribute<AzCategoryIdSubjectAccess> azAttrSubjectId =
[a207]              accSubj.createAzAttribute(
[a208]                            CONTAINER,
[a209]                            AzXacmlStrings.X_ATTR_SUBJECT_ID ,
[a210]                            azAttrValueSubjId);
[a211]
[a212]        System.out.println("TestAzApi azAttrSubjectId.getAttributeId: " +
[a213]                     azAttrSubjectId.getAttributeId());
[a214]
[a215]        // One step to create a subject attribute and add it to collection:
[a216]        accSubj.createAzAttribute(   // [a23]
[a217]              CONTAINER,
[a218]              AzXacmlStrings.X_ATTR_SUBJECT_AUTHN_LOC_AUTHENTICATION_METHOD,
[a219]              accSubj.createAzAttributeValue(
[a220]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING,
[a221]                     SAMPLE_SESSION_AUTH_METHOD));
[a222]
[a223]        // Add the Subject (demo, no effect) to the AzRequestContext:
[a224]        azReqCtx.addAzEntity(accSubj);  // [a26]
[a225]
[a226]
[a227]        // Get a Resource Category and AzEntity
[a228]        AzEntity<AzCategoryIdResource> azResource =
[a229]              azReqCtx.createAzEntity(
[a230]                     AzCategoryIdResource.AZ_CATEGORY_ID_RESOURCE); // [a31]
[a231]
[a232]        // Create a resource id attribute:
[a233]        azResource.createAzAttribute(
[a234]              APPLICATION,
[a235]              AzXacmlStrings.X_ATTR_RESOURCE_ID,
[a236]              azResource.createAzAttributeValue(
[a237]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING,
[a238]                     "file:C\\toplevel"));
[a239]
[a240]        // Create six AzAction Category AzEntity objects
[a241]        AzEntity<AzCategoryIdAction> azActionA01 =          // [a35]
[a242]              azReqCtx.createAzEntity(
[a243]                     AzCategoryIdAction.AZ_CATEGORY_ID_ACTION);
[a244]        AzEntity<AzCategoryIdAction> azActionA02 =          // [a36]
[a245]              azReqCtx.createAzEntity(
[a246]                     AzCategoryIdAction.AZ_CATEGORY_ID_ACTION);
[a247]        AzEntity<AzCategoryIdAction> azActionA03 =          // [a36]
[a248]              azReqCtx.createAzEntity(
[a249]                     AzCategoryIdAction.AZ_CATEGORY_ID_ACTION);
[a250]        AzEntity<AzCategoryIdAction> azActionA04 =          // [a36]
[a251]              azReqCtx.createAzEntity(
[a252]                     AzCategoryIdAction.AZ_CATEGORY_ID_ACTION);
[a253]        AzEntity<AzCategoryIdAction> azActionA05 =          // [a36]
[a254]              azReqCtx.createAzEntity(
[a255]                     AzCategoryIdAction.AZ_CATEGORY_ID_ACTION);
[a256]        AzEntity<AzCategoryIdAction> azActionA06 =          // [a36]
```

```
[a257]              azReqCtx.createAzEntity(
[a258]                     AzCategoryIdAction.AZ_CATEGORY_ID_ACTION);
[a259]
[a260]        AzCategoryId azCat = azActionA01.getAzCategoryId();
[a261]        System.out.println("azCat: class type = " + azCat.getClass().getName());
[a262]
[a263]    azActionA01.createAzAttribute(APPLICATION,                    // [a38]
[a264]              AzXacmlStrings.X_ATTR_ACTION_ID,
[a265]              azActionA01.createAzAttributeValue(
[a266]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING, "Read"));
[a267]    azActionA02.createAzAttribute(APPLICATION,                    // [a38]
[a268]              AzXacmlStrings.X_ATTR_ACTION_ID,
[a269]              azActionA02.createAzAttributeValue(
[a270]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING, "Write"));
[a271]    azActionA03.createAzAttribute(APPLICATION,                    // [a38]
[a272]              AzXacmlStrings.X_ATTR_ACTION_ID,
[a273]              azActionA03.createAzAttributeValue(
[a274]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING, "Delete"));
[a275]    azActionA04.createAzAttribute(APPLICATION,                    // [a38]
[a276]              AzXacmlStrings.X_ATTR_ACTION_ID,
[a277]              azActionA04.createAzAttributeValue(
[a278]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING, "Read"));
[a279]    azActionA05.createAzAttribute(APPLICATION,                    // [a38]
[a280]              AzXacmlStrings.X_ATTR_ACTION_ID,
[a281]              azActionA05.createAzAttributeValue(
[a282]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING, "Write"));
[a283]    azActionA06.createAzAttribute(APPLICATION,                    // [a38]
[a284]              AzXacmlStrings.X_ATTR_ACTION_ID,
[a285]              azActionA06.createAzAttributeValue(
[a286]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING, "Read"));
[a287]
[a288]        // Create set of actions
[a289]        Set<AzEntity<AzCategoryIdAction>> azActionSet =
[a290]              new HashSet<AzEntity<AzCategoryIdAction>>();
[a291]
[a292]        azActionSet.add(azActionA01);   // [a42]
[a293]        azActionSet.add(azActionA02);   // [a42]
[a294]        azActionSet.add(azActionA03);   // [a42]
[a295]        azActionSet.add(azActionA04);   // [a42]
[a296]        azActionSet.add(azActionA05);   // [a42]
[a297]        azActionSet.add(azActionA06);   // [a42]
[a298]
[a299]        azReqCtx.addResourceActionAssociation(
[a300]                              azResource, azActionSet);   // [a44]
[a301]
[a302]    // its time for some decisions !
[a303]    AzResponseContext azRspCtx = azHandle.decide(azReqCtx);
[a304]    AzResult azResult = null;
[a305]    Iterator<AzResult> itResults = azRspCtx.getResults().iterator();
[a306]
[a307]    while (itResults.hasNext()) {      // [a50]
[a308]
```

```
[a309]         azResult = itResults.next();  // [a52]
[a310]
[a311]
[a312]            // Obtain the corresponding action and resource objects
[a313]        AzResourceActionAssociation assoc =
[a314]                            azResult.getAzResourceActionAssociation()
[a315]         AzResource r1 = assoc.getAzResource();
[a316]          AzAction a1 =  assoc.getAzAction();
[a317]
[a318]         switch (azResult.getAzDecision()){
[a319]                case AZ_PERMIT:
[a320]                       // processing steps
[a321]                            ...
[a322]                            // process obligations
[a323]                       Iterator<AzObligation> itOb =
[a324]                       azResult.getAzObligations().iterator();
[a325]
[a326]                        while (itOb.hasNext()){
[a327]                            AzObligation azObligation = itOb.next();
[a328]                            Iterator<AzAttribute<?>> itAttr =
[a329]                                   azObligation.iterator();
[a330]                            while (itAttr.hasNext()){
[a331]                                 // process obligation attributes
[a332]                                      // found in itAttr.next(),
[a333]                                      }
[a334]                         }
[a335]                       break;
[a336]                case AZ_DENY:
[a337]                       break;
[a338]                case AZ_NOTAPPLICABLE:
[a339]                       break;
[a340]                case AZ_INDETERMINATE:
[a341]                       switch (azResult.getAzStatusCode()) {
[a342]                              case AZ_SYNTAX_ERROR:
[a343]                                     break;
[a344]                              case AZ_PROCESSING_ERROR:
[a345]                                     break;
[a346]                              case AZ_MISSING_ATTRIBUTE:
[a347]                                     // process missing attributes
[a348]                                     // and re-make decide call
[a349]                                     }
[a350]                       } // end switch
[a351]
[a352]
[a353]
[a354]
[a355]
[a356]
[a357]
[a358]
```

2.3 Query

The AzAPI includes support for a (restricted) query method. The method supports queries of the form: what are the resource-action pairs that a subject can perform in a certain scope? Thus, for example, in a portal application we could query to find out what services should be useable by a certain user over the internet, with the goal of displaying the services to the end-user. Scope is expressed by an abstract identifier (string) and it is assumed that this is resolved by some means by the Az API provider or the authorization engine.

```
[a359]    package org.example.azapi.test;
[a360]
[a361]    import org.example.azapi.*;
[a362]    // Initialize RequestContext azReqCtx with information about
[a363]    // the subject and environment, as done in the previous
[a364]    // examples.
[a365]    //
[a366]    // No associations are added to the RequestContext
[a367]
[a368]    // Sample code fragment using AzAPI (based on TestAzAPI)
[a369]    // variable defns to provide "context" for test program:
[a370]         // defn: "container": manages "environment" and "issues" env attributes
[a371]         public final static String CONTAINER = "orcl-weblogic";   // [a03]
[a372]         // defn: "application": manages "resources" and "issues" resource attrs
[a373]         public final static String APPLICATION = "hr-application-01"; // [a04]

[a374]         // test user; normally would be obtained from session info
[a375]         public final static String SAMPLE_SESSION_USER_NAME = "Joe User";
[a376]         public final static String SAMPLE_SESSION_AUTH_METHOD = "password";
[a377]
[a378]         // define the SCOPE string used in the query
[a379]        public final static string SCOPE = “portalServices”;
[a380]
[a381]
[a382]
```

```
[a383]
[a384]        // Get a reference to the authorization service:
[a385]        AzService azHandle = AzServiceFactory.getAzService();
[a386]
[a387]        // Create an AzRequestContext:
[a388]        AzRequestContext azReqCtx = azHandle.createAzRequestContext();
[a389]
[a390]        // Create an AzEnvironment entity and add an env attr:
[a391]        AzEntity<AzCategoryIdEnvironment> azEnv =
[a392]              azReqCtx.createAzEntity(
[a393]                     AzCategoryIdEnvironment.AZ_CATEGORY_ID_ENVIRONMENT);
[a394]
[a395]
[a396]
[a397]
[a398]
[a399]        // Create a general purpose attribute value
[a400]        // and add it to AzEnv
[a401]
[a402]        AzAttribute<AzCategoryIdEnvironment> azEnvAttr =    // [a13]
[a403]              azEnv.createAzAttribute(
[a404]                     CONTAINER,
[a405]                     X_ATTR_ENV_CURRENT_DATE_TIME,
[a406]                     azEnv.createAzAttributeValue(
[a407]                            AzDataTypeIdDateTime.AZ_DATATYPE_ID_DATETIME,
[a408]                            azEnv.createAzDataDateTime(new Date(),0,0,0)));
[a409]
[a410]
[a411]        // Add environment object to AzRequestContext
[a412]        azReqCtx.addAzEntity(azEnv);  // [a15]
[a413]
[a414]        // Get and print the attribute from the AzRequestContext:
[a415]        AzAttribute<AzCategoryIdEnvironment> azEnvCurrDateTime =
[a416]              azReqCtx.getAzEnvironment().
[a417]
[a418]
[a419]        // Get a Subject Category and AzEntity for access-subject attributes
[a420]        AzCategoryIdSubjectAccess azSubjCat =
[a421]              AzCategoryIdSubjectAccess.AZ_CATEGORY_ID_SUBJECT_ACCESS;
[a422]
              AzEntity<AzCategoryIdSubjectAccess> accSubj =
[a423]                     azReqCtx.createAzEntity(azSubjCat);   // [a17]
[a424]
[a425]
[a426]        // Two steps to create a subject attribute and add it to collection:
[a427]        AzAttributeValueString azAttrValueSubjId =
[a428]              accSubj.createAzAttributeValue(
[a429]                     AzDataTypeIdString.AZ_DATATYPE_ID_STRING,
[a430]                     SAMPLE_SESSION_USER_NAME);
```

```
[a431]         AzAttribute<AzCategoryIdSubjectAccess> azAttrSubjectId =
[a432]               accSubj.createAzAttribute(
[a433]                      CONTAINER,
[a434]                             AzXacmlStrings.X_ATTR_SUBJECT_ID ,
[a435]                             azAttrValueSubjId);
[a436]
[a437]         // One step to create a subject attribute and add it to collection:
[a438]         accSubj.createAzAttribute(   // [a23]
[a439]               CONTAINER,
[a440]               AzXacmlStrings.X_ATTR_SUBJECT_AUTHN_LOC_AUTHENTICATION_METHOD,
[a441]               accSubj.createAzAttributeValue(
[a442]                      AzDataTypeIdString.AZ_DATATYPE_ID_STRING,
[a443]                      SAMPLE_SESSION_AUTH_METHOD));
[a444]
[a445]         // Add (demo only) the Subject to the AzRequestContext:
[a446]         azReqCtx.addAzEntity(accSubj);  // [a26]
[a447]
[a448]
[a449]    // Once the the request context has been initialized,
[a450]    // we cam query to find the resources and actions allowable for this user
[a451]    // with reference to SCOPE
[a452]    Set<AzResourceActionAssociation> setResActAssoc =
[a453]                        query(SCOPE, azReqCtx);
[a454]
[a455]    // enumerate the information returned by the query
[a456]    while (setResActAssoc.hasNext()) {
[a457]
[a458]         AzResourceActionAssociation azAssoc = setResActAssoc.next();
[a459]        // service name
[a460]         AzResource r1 = assoc.getAzResource();
[a461]         // allowed action - invoke
[a462]         AzAction a1 =  assoc.getAzAction();
[a463]    ….
[a464]         // add service and action to webpage
[a465]      }
[a466]
```