SMARTER BALANCED ASSESSMENT CONSORTIUM

SBAC SSO Component Design Document

Version 2.0

4/13/2015

This document defines the technical design necessary to support the Smarter Balanced Assessment Consortium single sign-on component.

Created by Bill Nelson and Scott Heger, Identity Fusion, Inc.

Document Changes

Version	Author	Date	Comments
1.0	Kiran Ramineni	5/29/13	Created by Kiran Ramineni
1.1	David Lopez de Quintana	5/31/13	David and AIR Team's review and comments
1.2	Bill Nelson	8/24/13	Document redesign to include more detailed design components and ForgeRock specific content.
1.3	Bill Nelson	8/28/13	The document has been restructured to contain other products supporting the SBAC SSO Component (ForgeRock OpenAM and ForgeRock OpenIDM). This is now the central design document for the SSO Component.
1.4	Scott Heger	10/14/13	Added OpenAM configuration
1.5	Bill Nelson	11/4/13	Removed all references to OpenIDM. All functionality will be performed using OpenAM, OpenDJ, and custom scripts. Removed references to `<AssociatedEntityID>` and `<AssociatedEntity>` elements in the XML data feed. Added section for definition of XML schema.
1.6	Bill Nelson	11/24/13	Added RESET operation to support password reset.
1.7	Bill Nelson	12/18/13	Added SYNC operation to support synchronization between the program generating the XML file and the OpenDJ server. Updated description in MOD action to indicate that certain attributes would not be modified on the MOD action. Added section to indicate how test files would be recognized and processed. Replaced all UUID references of `1234-abcd-….-efabcd-9999` with a valid email address as this format will me more commonly used. Updated the architecture to reflect 3 instances of each SSO server. Added `Log Files` and `Processed Files` sections. Added `Appendix A - Scripts for Generating Sample Data`. Added sections for `SBAC Password Policy`, `Network Communication`, and `Security Questions`.
1.8	Bill Nelson	2/9/14	Updated the `Design Considerations` section to state that the Training Registration component is the single system of record for all user change events. Added the `Attribute Mapping` table section to demonstrate the relationship between attributes provided in the XML dump and those in the OpenDJ server. Added note in the `Account Management` section to clarify dropbox user, dropbox location, and protocol for transferring data to the OpenDJ server. Added text in the Unique Identifiers section to indicate that the OpenDJ uid attribute is populated by the email address received from the Test Registration component. Added sections for `User Accounts` and `Ports`. Added SETPWD operation to allow a Help Desk person to set a user's password to a known value in the Test Registration component. Added section detailing callback response to the Test Registration component.
1.9	Scott Heger	2/11/14	Added section on OpenAM customizations including full list of files that have been customized to provide the Smarter Balanced look and feel.
1.10	Scott Heger	3/28/14	Changed sample SAML attribute statement to include multiple sbacTenancyChain values. Also provided another link to obtain the IDP metadata that is signed by OpenAM
1.11	Bill Nelson	4/1/14	Updated Indexes section to indicate that OpenAM uses the `sbacUUID` attribute for authentication, not the `uid` attribute.
2.0	Scott Heger	4/13/2015	Made changes to the sample user object to reflect new format for the `sbacUUID` attribute, updated Web Application Container JVM Settings with correct path information, updated Java version information, updated IDP Metadata URL information, added note about Policy Agents not being used in the Smarter Balanced environment, updated list of attributes that can be used for Authorization, added up-to-date image showing a proper attribute statement, and updated the list of customized files. Also made references to OpenAM 12.

References:

This design document is created based on the following documents.

- SmaterBalanced_ArchitectureReport_120321.pdf
- SSO Use Cases.docx
- A peak load expectation based on a recent testing season at State of Minnesota

Acronyms Used in this Document

Term	Description
AIR	American Institutes For Research
CDSSO	Cross Domain Single Sign-On. The ability to provide single sign-on to applications that reside in a different DNS domain than the OpenAM server. CDSSO is provided by way of OpenAM policy agents and a servlet running on the OpenAM server.
CN	Common Name. The name by which an object is referenced. In the case of a person object, the CN is the person's full name.
CRUD	Create, Read, Update, and Delete. The term applies to operations performed on data elements.
DC	Domain Component. A portion of an organization's DNS domain. A domain of smarterbalanced.org has two domain components; dc=smarterbalanced and dc=org.
DIT	Directory Information Tree. The naming structure for a directory server. It shows the relationship of the entries in a hierarchical structure.
DN	Distinguished Name. The globally unique reference to an entry in a directory server. It consists of a comma delimited list of all entries starting with the entry you are referring to and including all parents found in the DIT all the way to the root suffix.
IDP	Identity Provider. An entity in a Federated environment that provides authentication services to SP applications.
LDAP	Lightweight Directory Access Protocol. A protocol for accessing or manipulating data found in the directory server.
OU	Organizational Unit. An attribute used to describe an organization grouping of data. It is typically used as a branch point in the DIT to store like objects together.
SAML	Security Assertion Markup Language. SAML is an XML-based open standard that defines the interaction between a Service Provider and Identity Provider for the purpose of authenticating users in a cross domain environment. The SAML assertion is issued by the Identity Provider once a client has authenticated. It contains attribute data that is consumed by the Service Provider for the purpose of defining access within the application.
SBAC	Smarter Balanced Assessment Consortium
SFTP	Secure File Transfer Protocol - the protocol used for transferring XML files from the Test Registration component to the OpenDJ server for account management.
SP	Service Provider. An entity in a Federated environment that provides services and uses the authentication services of a trusted IDP
SSO	Single sign-on. This is the generic name given to the component which provides authentication and authorization capabilities to the Smarter Balanced environment. The SSO component is comprised of a user provisioning engine, an identity store, and a Web single sign-on solution.
ST	State. An attributed used to describe a state (i.e. California); typically used in the context of an address.
UUID	Universally Unique Identifier. A globally unique identifier for users in the Smarter Balanced environment.
WSSO	Web Single Sign-On. The method of providing ones credentials to one server in a

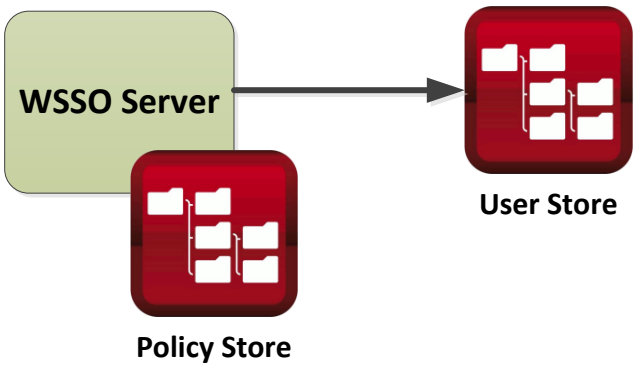
	Web environment and not being required to enter them again with trusted servers in that environment.
XML	Extensible Markup Language. A markup language that defines a set of rules for encoding data in a format that meets the needs of the application for which it has been purposed. A benefit of XML is its ability to structure a hierarchical relationship between entries.

Introduction:

"The Smarter Balanced Assessment Consortium (Smarter Balanced) is a state-led consortium working to develop next-generation assessments that accurately measure student progress toward college- and career-readiness. "

A key component in this initiative is a product that provides Web-based single sign-on ("WSSO") services for Smarter Balanced client components. A WSSO solution provides both the authentication and authorization services necessary for a client component to determine whether a user has successfully authenticated into the Smarter Balanced environment and if so, what permissions they may have within the client component.

Authentication credentials and authorization policies are typically stored in a LDAP-based directory server ("LDAP Server"). It is feasible that both sets of information may be stored within the same LDAP Server, but this is typically not the case. Authentication credentials are oftentimes found in a central, user-based LDAP Server and policies are stored in the WSSO's internal policy store (which is typically an LDAP Server, itself). The following diagram demonstrates the relationship between the WSSO Server (and its internal authorization Policy Store) and the external authentication User Store.



While the User Store could be any type of database, an LDAP Server is suggested for the Smarter Balanced initiative in order to achieve the performance, scalability, and high availability requirements associated with the initiative.

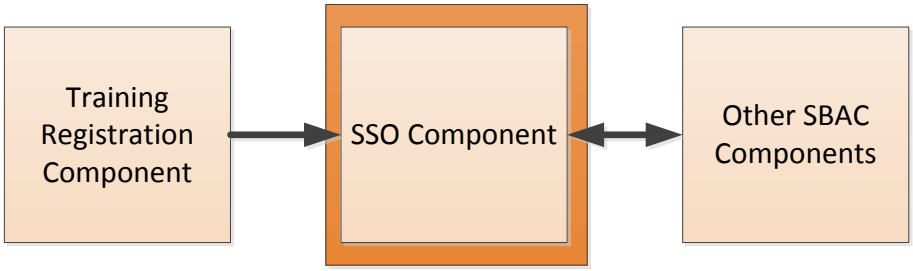
The SBAC Training Registration Component is the authoritative source for data contained in the LDAP Server. As the authoritative source, all changes (creates, updates, and deletes) to user data originate in the Training Registration Component and are provided to the SSO Component for processing. This is accomplished by way of a batch file which contains detailed instructions on recent user updates. The SSO Component processes the directives found in the batch file and updates the LDAP server accordingly. New users are able to sign in to the SBAC environment immediately after their accounts have been created. This is due to the fact that the WSSO component uses the credentials stored in the LDAP server for authentication purposes.

Note: The American Institutes for Research ("AIR") has selected the ForgeRock OpenAM and OpenDJ products to provide Web single sign-on and enterprise directory services, respectively. See www.forgerock.com for more information on these products.

Scope of the Design:

The LDAP Server is the core identity store within the SSO Component. It provides the WSSO solution with the credentials, policies, and user data required for an SBAC client component to determine a user's access within the application.

This document provides the architectural design necessary to implement the SSO Component in a supporting role to other Smarter Balanced components. It includes the interfaces between each component, the rules and business logic for updating user data within the SSO Component, and the data structures necessary to provide SBAC Components with the information they require in order to adequately determine a user's access rights.



This design has been created in such a way as to meet the requirements for a WSSO solution supporting a large user base (approximately 3 million users).

Design Considerations:

The following are some of the key considerations used in developing this design document.

- All account management activities flow into the SSO component from one (and only one) source.
- The Training Registration Component is the system of record for all user data.
- All account management activities (additions, modifications, deletions, etc.) are performed in the Training Registration Component and exposed to the SSO component by way of an exported flat file.
- The flat file will consist of change events only.
- The focus is on capturing and storing user information in an LDAP Server. The LDAP Server provides an extreme read performance where large number of entries is involved.

- The users are typically the staff of a State, District or School. The users include administrative staff as well as teaching staff irrespective of their employment status.
- Special attention was paid in developing the Schema and DIT so that it's flexible to accommodate the structural relationship between entities (i.e., States, Districts and Schools). One of the considerations was to accommodate schools assigned to the State without a District.
- Special attention was paid to minimize data management issues due to changes in the structural relationship between entities. For example, when a school is moved from one district to another, only limited number of updates will be required on the LDAP server. This is an important key requirement that the data should be loaded relatively quickly during the limited window of time during nightly maintenance periods.

Assumptions:

The following assumptions are made in creating this design.

- The ForgeRock Open Identity Stack will be used to provide overall identity-related services as follows:
    - ForgeRock OpenDJ Directory Server - User and Credential Store (enterprise directory)
    - ForgeRock OpenAM Server - Web single-sign on (central authentication and authorization)
- The WSSO component will utilize the OpenDJ Directory Server for authenticating users into the Smarter Balanced environment
- User authentication is based on email address and password (both attributes stored in the OpenDJ Directory Server)
- The WSSO component will utilize the OpenDJ Directory Server for providing client components with authorization data.
- Authorization data will consist of a user's role within the context of the users relationship to a particular entity
- The Test Registration component will be considered the authoritative source for all data flowing into the OpenDJ Directory Server.
- The Test Registration component will generate dumps of user data and copy these files to the OpenDJ server. The operating system will detect uploaded files and launch a custom script to process these files accordingly. All changes detected in the data dump will be updated in the OpenDJ server where users will be able to log in once their accounts have been provisioned.
- Some data (the fine grained permissions that depend on the role or composite role) are not stored in the directory; these are found in the Permissions component

- The OpenAM server will be the only LDAP client to the OpenDJ server

Note: Any modifications to the SSO component that deviates from these design considerations and/or assumptions may require a redesign of key subsystems in the SSO component.

Directory Design

This directory design section describes the LDAP objects, attribute types and the structure of the directory (DIT). The existing LDAP attribute types are used where applicable so as to avoid duplication of attribute type definitions.

Schema

To better understand the concepts of directory schema is helps to look at a basic example of a generic directory entry.

Note: This is a generic example for explanatory purposes only. It should not be considered as a recommendation for how a user entry in the Smarter Balanced environment should be modeled. See Smarter Balanced User Object for SBAC-related recommendations.

```
dn: uid=bnelson,ou=People,dc=smarterbalanced,dc=org
cn: Bill Nelson
sn: Nelson
givenname: Bill
uid: bnelson
l: Tampa
mail: bill.nelson@identityfusion.com
telephoneNumber: +1 813 555 1234
objectclass: top
objectclass: person
object tclass: organizationalPerson
objectclass: inetOrgPerson
```

The first line of this directory entry is the DN (or "distinguished name"), which is represented using the `dn:` context at the beginning of the entry. The DN uniquely identifies the entry in the directory database and where it is located in the directory tree hierarchy. The directory entry uses the `uid` (unique identifier) attribute in the distinguished name to uniquely identify the entry within the `People` branch point.

The `cn` (common name), `sn` (surname) and `givenname` attributes describe the entry in more detail. These attributes specify the entry's full name, last name, and first name, respectively.

The `uid` (user identifier) attribute has been selected as the leftmost portion of the distinguished name. This constitutes a "relative" distinguished name (or "RDN") as that attribute value must be unique within this portion of the directory information tree. When an attribute is used as an RDN, the attribute is required despite what the `objectclass` definitions might indicate. This is necessary to ensure that the value is entered during the creation of the directory entry.

The remaining lines of the directory entry contain additional attributes to help further describe the entry. The `l` (locality) attribute specifies the location where the user may be; this is often used to specify the user's city. The `mail` attribute is used to designate the user's email address. The `telephoneNumber` attribute is used to specify the user's telephone number.

The `objectclass` attributes define the rules for the entry. An object class is a special attribute type called `objectclass` that basically defines which attributes are mandatory and which are optional for a specific entry.

The following terms are used when defining attributes associated with particular object classes:

Must	An entry cannot be created without providing this attribute.
May	An entry may be created without providing this attribute. The attribute may be added at a later time.
Singlevalue	Only one attribute value is possible for one entry. Attempts to add additional values will fail.
Multivalue	One or more attribute values are possible for entry. There is no limit on the number of attribute values assigned to entry.

The `objectclass` attribute also establishes the entry type based on its given value. In the example above, you can deduce that this entry is a person by looking at the values given for the `objectclass` attributes (`person`, `organizationalPerson`, and `inetOrgPerson`).

The `top` object class defines the first rule for the entry. This rule states that the entry must use the attribute `objectclass`. This is a self-defining rule and is needed to maintain schema consistency.

The OpenDJ LDAP Server is configured with a default schema that includes object classes and attributes to handle most entry types. Often times, however, it is desirable to extend the default schema to add site specific object classes and attributes. Directory Administrators have the options to name their custom object classes and attributes however they want but best practice is to prefix them with a site specific identifier. This makes it easier to identify modifications to the default schema.

Note: The objects and attributes created for this directory have a prefix of **`sbac`** for easy identification.

Smarter Balanced-Specific Attributes

The following attributes have been created for the Smarter Balanced initiative.

- `sbacUUID` - (single-valued, required) this is the unique identifier for this user. It is unique throughout the Smarter Balanced environment, is maintained by the Identifier Management component, and is included in the data feed from the Test Registration component. See Unique Identifiers for more information.

- sbacTenancyChain - (multi-valued, not required) this attribute contains the user's role within a given scope. The scope may consist of some combination of entities that define where the user is permitted to use this role.

  The format of the sbacTenancyChain is as follows:
  |RoleID|Name|Level|ClientID|Client|GroupOfStatesID|GroupOfStates|StateID|State|GroupOfDistrictsID|GroupOfDistricts|DistrictID|District|GroupOfInstitutionsID|GroupOfInstitutions|InstitutionID|Institution|

Smarter Balanced-Specific Object Classes

The following object classes have been created for the Smarter Balanced initiative.

sbacPerson (Parent: inetOrgPerson):

This object class defines the SBAC specific attributes for a Person entity. It is a structural object class and is extended from inetOrgPerson. This object class will contain the following attribute types.

ObjectClass: sbacPerson

Parent(s): top->person->organizationalPerson->inetOrgPerson

Attribute Name	Source	Syntax	Multi-Valued	REQ
objectClass	top		Y	Y
sn	organizationalPerson	IA5String	Y	Y
telephoneNumber	organizationalPerson	TelephoneNumber	Y	N
mail	inetOrgPerson	IA5String	Y	N
givenName	inetOrgPerson	IA5String	Y	N
uid	inetOrgPerson	IA5String	Y	N
userPassword	inetOrgPerson	OctetString	N	N
sbacUUID	sbacPerson	IA5String	N	Y
sbacTenancyChain	sbacPerson	IA5String	Y	N
inetUserStatus	inetUser	IA5String	N	Y

Smarter Balanced User Object

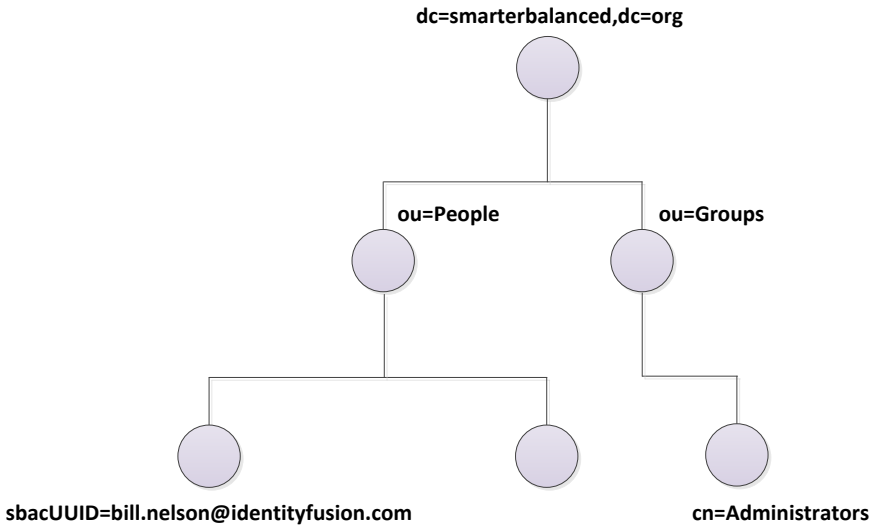
A sample entry using the object class definition defined in the previous section would be as follows:

```
dn: sbacUUID=530fc14ae4b0ccc08efc1d2c,ou=People,dc=smarterbalanced,dc=org
sbacUUID: 530fc14ae4b0ccc08efc1d2c
inetUserStatus: Active
uid: sbac.wi@gmail.com
userPassword: {SSHA}wKUu+/f/Crhk2l+y3lJ0c1g4WC4MK/WkKHWRPg==
objectClass: person
objectClass: sbacPerson
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: inetuser
```

```
objectClass: iplanet-am-user-service
objectClass: top
objectClass: sunFMSAML2NameIdentifier
givenName: Aaron
sn: Rodgers
telephoneNumber: 608-123-4567
mail: sbac.wi@gmail.com
```

Directory Information Tree (DIT)

The following DIT will provide the flexibility needed so that the entities may easily be assigned to one another. The DIT will have a flat structure as follows.



Root Suffix

The root suffix is a common base entry that holds all sub entries for a given directory tree. It is called a root suffix because it is defined at the top of the directory tree within a given Directory Server. The root suffix is the beginning of a new database within the Directory Server. The database for the root suffix contains at least one entry – the root suffix entry, itself.

A root suffix can take on any name, but to avoid naming conflicts when merging directory servers, a best practice is to define a name that is reflective of the organization's domain name. As such, the domain component ("DC") is typically used when creating a name for the root suffix.

The root suffix for the Smarter Balanced LDAP Server is **`dc=smarterbalanced,dc=org`**.

Branch Points

A branch point is any entry that contains sub entries below it. This is also known as a *container*.

The Smarter Balanced DIT will have two main branches: `ou=People` and `ou=Groups`. The `People` container will include entries for all users who are currently active in the environment. The `Groups`

container will allow groups of users to be created for any purpose. This allows for the creation of administrative users or groups of users that OpenAM may need to perform certain tasks.

Leaf Nodes

A leaf node is the end point in the directory information tree and contains no entries beneath it. The Smarter Balanced DIT will have leaf nodes for both user entries and group entries.

Indexes

The LDAP Server database contains both directory entries and any corresponding index entries. The index entries are essentially a catalog of the entire index types specified in the index configuration for a particular attribute, and matching pointers to the actual directory entries within the database. Use of indexes can significantly improve search performance of the server. However, creating, managing, and maintaining indexes may impose an additional load on the server. Therefore, careful consideration of the attributes chosen for indexing is necessary.

OpenAM is the only client to the OpenDJ server and it will perform lookups based on the `sbacUUID` attribute. Since no other applications will be searching the OpenDJ server, it is possible to remove the default indexes for any non-operational attributes. These include the following: `cn`, `givenName`, `mail`, `sn`, and `telephonenumber`. It is not advisable; however, that any indexes for operational attributes be removed as this may impact internal performance on the server.

The following table provides an overview of the indexing strategy that will be performed on the OpenDJ database.

ds-cfg-attribute	presence	equality	substring	ordering
aci	X			
ds-sync-conflict		X		
ds-sync-hist				X
entryUUID		X		
member		X		
objectClass		X		
sbacUUID		X		
uid		X		
uniqueMember		X		

The default maximum of index entries maintained on any one attribute is 4000. This means that the LDAP Server will stop maintaining indexes for an attribute once the number of index entries exceeds 4000. A limit of 4000 is not adequate to handle millions of users but without further information regarding the structure of the `uid`, it cannot be determined what this number should be. Instead, the

server should be monitored during the implementation process to better determine the most efficient number of indexes that can be supported.

OpenAM Design

OpenAM provides authentication and single sign-on services within the SSO Component. It provides the Smarter Balanced branded login pages and upon successful authentication provides client components with the data they need to make authorization and branding decisions.

Web Application Container

OpenAM is a J2EE web application and therefore requires a web application container to be deployed to. OpenAM supports a number of different web application containers with the preferred container being Apache Tomcat.

In the Smarter Balanced environment OpenAM will be deployed to Apache Tomcat version 7. Each instance of Apache Tomcat for OpenAM in the Smarter Balanced environment will be configured with the following JVM parameters for optimal performance:

```
# Configrable JVM config
JAVA_OPTS="$JAVA_OPTS -server"
JAVA_OPTS="$JAVA_OPTS -XX:NewRatio=3"
JAVA_OPTS="$JAVA_OPTS -Xms3072M"
JAVA_OPTS="$JAVA_OPTS -Xmx3072M"
JAVA_OPTS="$JAVA_OPTS -XX:PermSize=256m"
JAVA_OPTS="$JAVA_OPTS -XX:MaxPermSize=256m"
JAVA_OPTS="$JAVA_OPTS -Dsun.net.client.defaultReadTimeout=60000"
JAVA_OPTS="$JAVA_OPTS -Dsun.net.client.defaultConnectTimeout=30000"

# Garbage Collecton
JAVA_OPTS="$JAVA_OPTS -verbose:gc"
JAVA_OPTS="$JAVA_OPTS -Xloggc:/opt/tomcat/logs/gc.log"
JAVA_OPTS="$JAVA_OPTS -XX:+PrintClassHistogram"
JAVA_OPTS="$JAVA_OPTS -XX:+PrintGCTimeStamps"
JAVA_OPTS="$JAVA_OPTS -XX:+PrintGCDetails"
JAVA_OPTS="$JAVA_OPTS -XX:+UseParNewGC"
JAVA_OPTS="$JAVA_OPTS -XX:+UseConcMarkSweepGC"
JAVA_OPTS="$JAVA_OPTS -XX:+UseCMSCompactAtFullCollection"
JAVA_OPTS="$JAVA_OPTS -XX:+CMSClassUnloadingEnabled"
JAVA_OPTS="$JAVA_OPTS -XX:+DisableExplictGC"

# Heap Dumps
JAVA_OPTS="$JAVA_OPTS -XX:+HeapDumpOnOutOfMemoryError"
JAVA_OPTS="$JAVA_OPTS -XX:HeapDumpPath=/opt/tomcat/logs/heapdump.hdprof"
```

JAVA

OpenAM server software runs in a Java EE Web container, and requires a Java Development Kit.

OpenAM supports the Oracle Java Development Kit 6, 7, or 8. It is recommended to use the most recent Java update, with the latest security fixes. In the Smarter Balanced environment the JDK that OpenAM will use is:

```
java version "1.7.0_76"
Java(TM) SE Runtime Environment (build 1.7.0_76-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.76-b04, mixed mode)
```

Realms

A realm is the unit that OpenAM uses to organize configuration information. The data stored in a realm can include, but is not limited to:

- One or more subjects (a user, a group of users, or a collection of protected resources)
- A definition of one or more data stores to store subject (user) data
- Authentication details identifying, for example, the location of the authentication repository, and the type of authentication required.
- Responder configurations that allows applications to personalize the user experience, once the user has successfully authenticated and been given access.
- Administration data for realm management

/ (Top Level Realm)

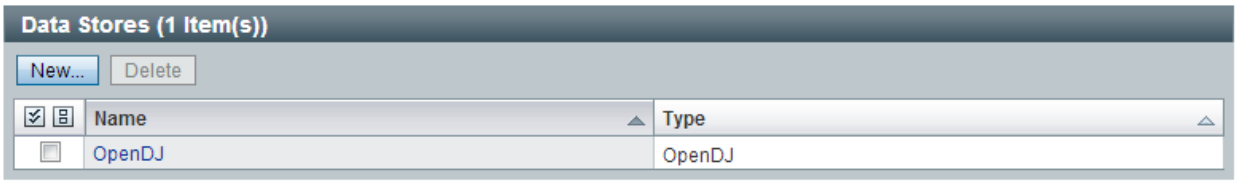
When OpenAM is first installed the `/ (Top Level Realm)` is created and contains the initial OpenAM configuration as well as the administrative account called `amadmin`. In the Smarter Balanced environment this realm will remain mostly at its default settings except for Agents. All policy agent profiles, both Web and J2EE, will be defined in this realm.

sbac Realm

To separate administrative functions from regular user functions and to provide the custom Smarter Balanced authentication interface, a sub realm called `sbac` will be configured as shown here.

Data Stores

Within the `sbac` sub realm the OpenDJ LDAP server will be configured as the Data Store for the realm.



This Data Store configuration consists of parameters that define how to connect to the Data Store and how to search for subjects (users and groups). The following table lists the parameters that will be configured for the OpenDJ Data Store.

Parameter	Value
Server Settings	
LDAP Server	localhost:1389
LDAP Bind DN	cn=SBAC Admin
LDAP Bind Password	**********
LDAP Organization DN	dc=smarterbalanced,dc=org
LDAP Connection Pool Minimum Size	10
LDAP Connection Pool Maximus Size	65
User Configuration	
LDAP Users Search Attribute	sbacUUID
Persistent Search Controls	
Persistent Search Base DN	dc=smarterbalanced,dc=org
Cache Control	
Caching	enabled
Maximum Size of the Cache	1048576

All other parameters will be left at their default values.

Authentication

OpenAM supports many authentication modules. In the Smarter Balanced environment the LDAP authentication module will be used. The OpenAM LDAP module provides password reset capabilities and support for password policies provided by OpenDJ. The LDAP authentication module will be configured in an authentication chain called `sbacChain` and the criteria will be set to `REQUIRED`. The `sbacChain` will be set as the Default Authentication Chain for users in the `sbac` realm.

Federation

To support single sign-on with applications across different DNS domains that won't use the proprietary Cross Domain Single Sign-On (CDSSO) capability of the OpenAM policy agents, OpenAM will be configured as a SAMLv2 Identity Provider (IDP) in the Smarter Balanced environment. The IDP and all Service Provider (SP) applications will be configured in a Circle of Trust (CoT) called `sbac`. The `sbac` CoT will reside in the `/sbac` realm.

OpenAM exposes a URL for obtaining metadata for the SAML entities in its configuration. The URL format is:

```
http(s)://fqdn:port/<openamcontext>/saml2/jsp/exportmetadata.jsp
```

From the comments from within the JSP:

This JSP is used to export standard entity metadata,
there are three supported query parameters:

- **role** -- role of the entity: sp, idp or any
- **realm** -- realm of the entity
- **entityid** -- ID of the entity to be exported
- **sign** -- sign the metadata if the value is "true"

If none of the query parameter is specified, it will try to export the first hosted SP metadata under root realm. If there is no hosted SP under the root realm, the first hosted IDP under root realm will be exported. If there is no hosted SP or IDP, an error message will be displayed.

Examples:

Specifying a realm:
http://sso-dev.opentestsystem.org:8080/auth/saml2/jsp/exportmetadata.jsp?metalias=idp&realm=sbac

Specifying a realm and requesting metadata to be signed:
http://sso-dev.opentestsystem.org:8080/auth/saml2/jsp/exportmetadata.jsp?metalias=idp&realm=sbac&sign=true

Session Synchronization

While the SAML 2.0 specification doesn't provide a mechanism for maintaining session activity between entities, OpenAM IDPs have an advanced configuration option called Session Synchronization. When enabled this allows the IDP to notify all SPs when a user's session times out or if the user explicitly logs out. In the Smarter Balanced environment this option will be enabled.

Additional information on SAML 2.0 and access to all SAML 2.0 specification documentation can found here:

http://en.wikipedia.org/wiki/SAML_2.0

Client Component Application Integration

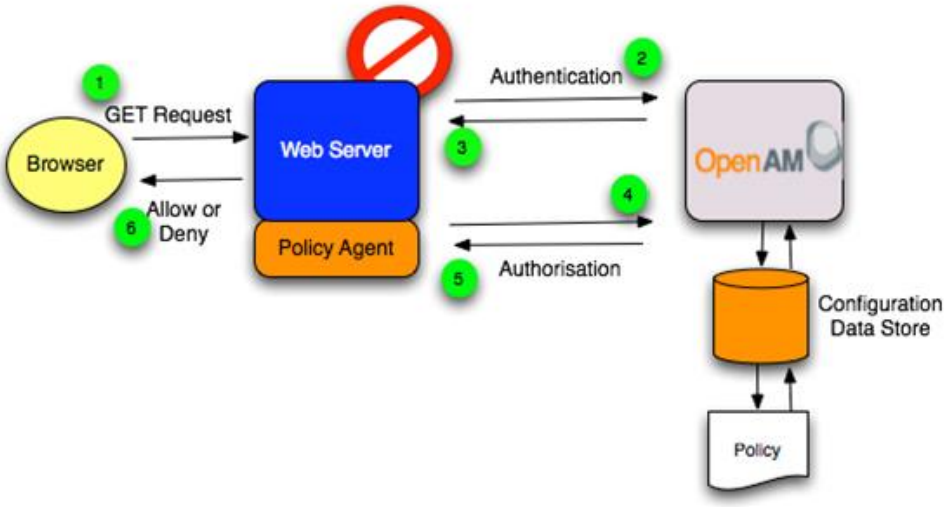
To use the single sign-on services of the SSO system, client component applications must be integrated into the environment. In the Smarter Balanced environment OpenAM will provide authentication services only. Each client component application will be responsible for handling authorization. OpenAM will provide user information for this purpose. OpenAM provides a number of different approaches for integrating applications so they can utilize the authentication services provided by it. In the Smarter Balanced environment two methods are supported:

- OpenAM Policy Agents
- SAML 2.0

Both approaches will require some programming on the client component application side to consume the information provided. The OpenAM Policy Agents will require the least amount of programming on the client component application side but they provide single sign-on using a proprietary language between the agent and the OpenAM server. Using SAML 2.0 will require more programming work to integrate but the language used to provide single sign-on is a well-known standard defined by the SAML 2.0 specification.

Policy Agents

Policy Agents are modules or plug-ins that are installed into the container that hosts a client component application. It is the function of the Policy Agent is to examine each incoming request to the container and ensure that users are authenticated (and optionally authorized) to access the application. If the user is not authenticated they will be redirected to the OpenAM login interface to authenticate.



The flow is described as follows:

1. The browser sends a request to an application protected by a Policy Agent.
2. The agent checks the validity of the user's session.
3. OpenAM informs the agent of the user's session validity.
4. If the user's session is valid, the agent sends a policy evaluation request to OpenAM (if configured). OpenAM evaluates the request.
5. OpenAM responds with a policy decision.
6. The agent interprets the policy decision and allows or denies access to the application.

OpenAM Policy Agents come in two different varieties:

- Web Policy Agents
- J2EE Policy Agents

Web Policy Agents provide one of the fastest and lightest integration paths. If the application resides in a supported web server then a Web Policy Agent can be used to provide single sign-on services of OpenAM.

The list of supported Web Servers can be found here:

http://openam.forgerock.org/openam-documentation/openam-doc-source/doc/web-release-notes/index.html#web-server-requirements-web-agents

If the application resides in a supported J2EE application server then a J2EE Policy Agent can be used for application integration. The J2EE Policy Agent allows for a tighter degree of integration between applications and the OpenAM server as the OpenAM client SDK is made available but would require a degree of Java coding in order to use it.

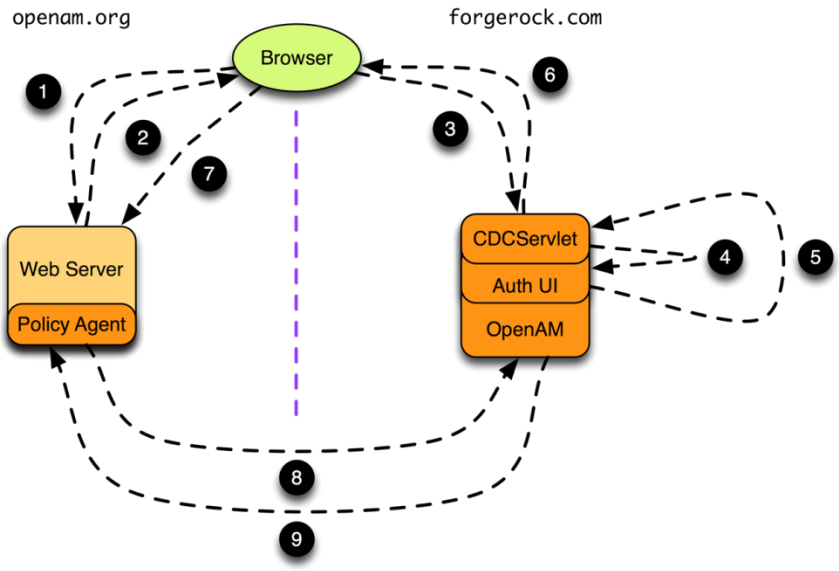
The supported list of J2EE servers can be found here:

http://openam.forgerock.org/openam-documentation/openam-doc-source/doc/jee-release-notes/index.html#web-container-requirements-javaee-agents

In the Smarter Balanced environment Policy Agents are not being used.

Cross Domain Single Sign-On (CDSSO)

OpenAM uses cookies as a way to pass session tokens between applications for providing single sign-on. Cookies have strict security restrictions that control the domains into which cookies can be set. Cookies can only be set in the same domain as the server issuing the cookie and cookies will only be sent to the same domain being access by the user's browser. CDSSO is a feature of OpenAM and OpenAM Policy Agents that can be used to integrate applications in different DNS domains into a single sign-on environment. The CDSSO flow can be seen in the following diagram:



The flow is as follows:

1. The user's browser makes a request to an application protected with a Policy Agent (without a valid session)
2. The Policy Agent redirects the client to the `CDCServlet`.
3. The client follows the redirect and accesses the `CDCServlet`.
4. The `CDCServlet` determines that the user is unauthenticated and proxies the request through to the authentication interface. The `CDCServlet` updates the `goto` URL to ensure the user is redirected back to the `CDCServlet`.
5. The user authenticates to the authentication interface and then redirected back to the `CDCServlet`. OpenAM sets a session cookie in the user's browser in the DNS domain where the OpenAM server resides.
6. The `CDCServlet` processes the request and returns the generated Assertion to the client via the `CDCClientServlet`.
7. The browser auto-submits the Assertion to the Policy Agent for processing.
8. The Policy Agent processes the Assertion and sends the extracted token to the OpenAM server for validation.
9. OpenAM responds to the session validation request and the Policy Agent processes the response and then sets a session cookie in the browser in the DNS domain where the Policy Agent resides.

In the Smarter Balanced environment CDSSO will be used where necessary to achieve single sign-on across DNS domains where Policy Agents are used for integration.

Additional information about CDSSO and how to configure it can be found here:

http://docs.forgerock.org/en/openam/10.1.0/admin-guide/index/chap-cdsso.html

SAML 2.0

Security Assertion Markup Language (SAML) is an XML-based protocol that uses security tokens called assertions to pass information about authenticated users from Identity Providers to Service Provider applications. SAML 2.0 was ratified as an OASIS standard in 2005 and remains an extremely popular and well supported standard for providing web-based authentication and single sign-on across different DNS domains.

The authentication flow using SAML 2.0 is nearly identical to the proprietary CDSSO flow described above except that the Assertion contents are defined and standardized by the SAML 2.0 specification and are provided by the Federation SOAP services of OpenAM. In addition there is no agent on the SP application side. It is up to the SP client component application developer to build the SAML 2.0 functionality into their client component application. At a minimum the SP client component application will need to provide the following:

- **Assertion Consumer Service** – a service that will receive the SAML 2.0 assertion from the IDP and process its contents for the purposes of establishing a local session for the user and for authorization of the user as described below.
- **Single Logout Service** – a service that will receive SAML 2.0 logout requests from the IDP and terminate the user's local session when called.
- **Service Provider Metadata** – An XML file to be provided to and consumed by the IDP to establish the SP as a trusted entity within a CoT. This file must conform to SAML 2.0 metadata specifications and define the services offered by the SP.

Authorization

As stated above, each client component application will be responsible for providing user authorization. OpenAM will provide integrated client component applications with data about the authenticated user that can be used to determine authorization. Once a user has been authenticated, OpenAM then obtains the following user attributes from OpenDJ and makes them available to the client component application for authorization purposes:

- `telephoneNumber` (Telephone Number)
- `cn` (Common Name)
- `sbacUUID` (Unique User Identifier)
- `sbacTenancyChain` (Mapping of role to scope)
- `mail` (Email Address)
- `sn` (Last Name)
- `givenName` (First Name)

Policy Agent Integration

OpenAM Policy Agents have the ability to inject user attribute data into the HTTP request that is being sent to the application. This data can be injected in the form of cookies or HTTP headers. In the Smarter Balanced environment this data will be supplied as HTTP headers with the names of the attributes shown above. The client component application will need to look for these headers in the HTTP requests and act upon them accordingly to provide authorization.

SAML 2.0 Integration

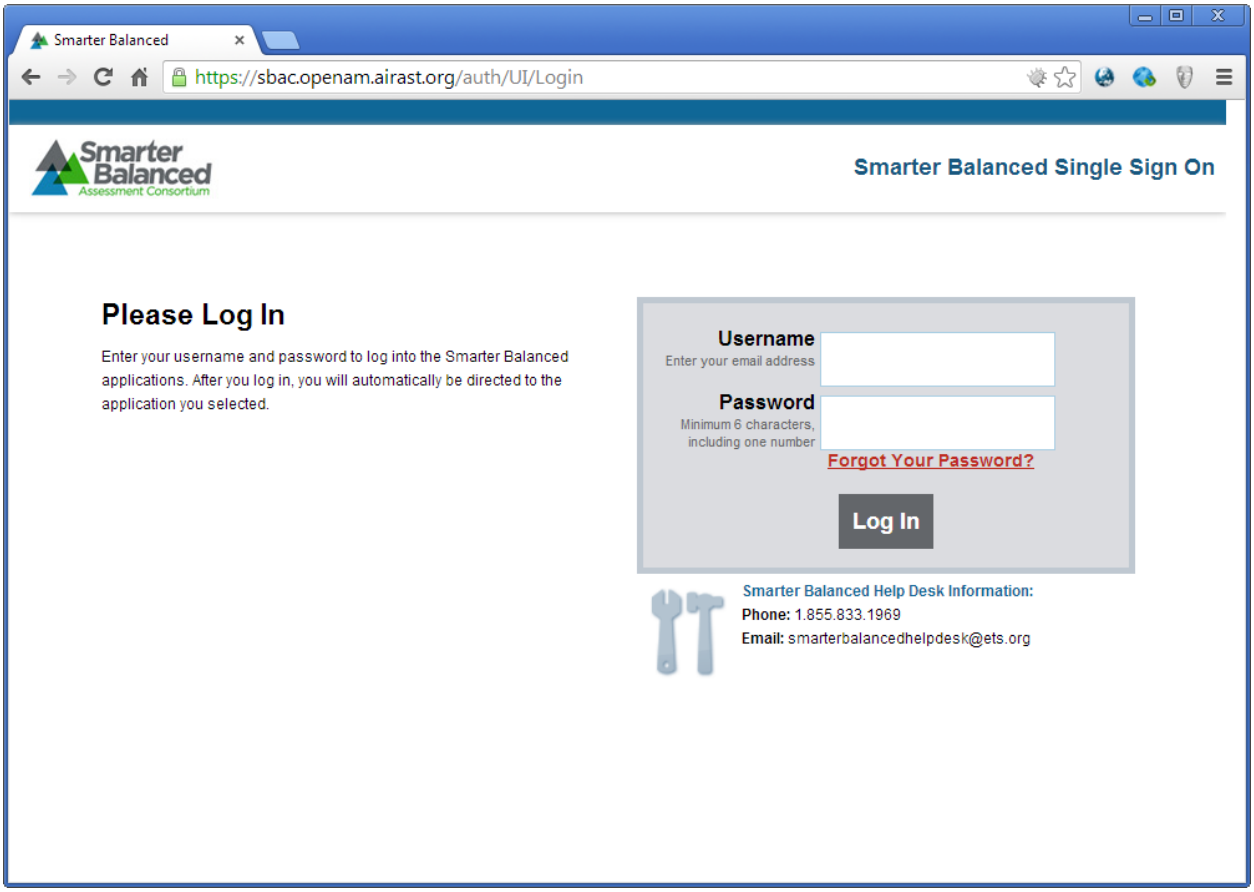
SAML 2.0 Assertions can contain an optional `AttributeStatement` that can contain attribute data about the user defined in the Assertion. In the Smarter Balanced environment OpenAM will provide an `AttributeStatement` with the attributes defined above. The attribute names in the `AttributeStatement`, however, will be different than the actual LDAP attribute names. The following is an example `AttributeStatement` showing the format:

```
- <saml:AttributeStatement>
  - <saml:Attribute Name="mail">
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">sbac.wi@gmail.com</saml:AttributeValue>
    </saml:Attribute>
  - <saml:Attribute Name="sbacUUID">
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">530fc14ae4b0ccc08efc1d2c</saml:AttributeValue>
    </saml:Attribute>
  - <saml:Attribute Name="sn">
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">Rodgers</saml:AttributeValue>
    </saml:Attribute>
  - <saml:Attribute Name="cn">
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">Aaron Rodgers</saml:AttributeValue>
    </saml:Attribute>
  - <saml:Attribute Name="telephoneNumber">
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">608-123-4567</saml:AttributeValue>
    </saml:Attribute>
  - <saml:Attribute Name="sbacTenancyChain">
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">|SB|Test Specification Read|STATE_GROUP|1000|QA
        Client|SB|test||||||||||</saml:AttributeValue>
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">|WI|Test Author|STATE|1000|QA
        Client|||WI|Wisconsin|||||||||</saml:AttributeValue>
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">|SB|Test Author|STATE_GROUP|1000|QA
        Client|SB|test||||||||||</saml:AttributeValue>
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">|WI|Tib Viewer|STATE|1000|QA
        Client|||WI|Wisconsin|||||||||</saml:AttributeValue>
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">|SB|Tib Viewer|STATE_GROUP|1000|QA
        Client|SB|test||||||||||</saml:AttributeValue>
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">|WI|Test Specification Read|STATE|1000|QA
        Client|||WI|Wisconsin|||||||||</saml:AttributeValue>
    </saml:Attribute>
  - <saml:Attribute Name="givenName">
      <saml:AttributeValue xsi:type="xs:string" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema">Aaron</saml:AttributeValue>
    </saml:Attribute>
</saml:AttributeStatement>
```

The client component application will need to look for the `AttributeStatement` in the Assertion and act upon the data within to provide authorization.

Customizations

OpenAM allows for the customization of End User pages so you can brand them with the look and feel that you desire. In the Smarter Balanced environment a number of pages, including the Login page shown below, have been created and/or modified to match the Smarter Balanced branding.



In addition to the Login page, the following pages have also been branded to match this same look and feel:

- Logout
- Session Timeout
- Authentication Failed
- Inactive User
- Forgot Password
- Change Password
- Security Questions

These pages are Java Server Pages (JSPs) that use a combination of Cascading Stylesheets (CSS), JavaScript, XML, Images and properties files to deliver the look and feel. The complete list of modified files in the Smarter Balanced environment broken down by type is:

JSP

- Password Reset
  - $WEB_APP/password/ui/PWResetSuccess.jsp
  - $WEB_APP/password/ui/PWResetQuestion.jsp

- $WEB_APP/password/ui/PWResetUserValidation.jsp

- Change Password & Security Questions
  - $WEB_APP/console/idm/EndUser.jsp
  - $WEB_APP/console/user/UMChangeUserPassword.jsp
  - $WEB_APP/console/user/UMUserPasswordResetOptions.jsp
- Login, Logout, etc.
  - $WEB_APP/config/auth/default/services/sbac/html/user_inactive.jsp
  - $WEB_APP/config/auth/default/services/sbac/html/Logout.jsp
  - $WEB_APP/config/auth/default/services/sbac/html/Login.jsp
  - $WEB_APP/config/auth/default/services/sbac/html/session_timeout.jsp
  - $WEB_APP/config/auth/default/services/sbac/html/login_failed_template.jsp
- SAML
  - $WEB_APP/saml2/jsp/idpSSOFederate.jsp

Images

- $WEB_APP/images/tools.png
- $WEB_APP/images/favicon.ico
- $WEB_APP/images/logo_sbac.jpg
- $WEB_APP/images/errorandalert.png
- $WEB_APP/images/greenCheck.png
- $WEB_APP/images/successandalert.png
- $WEB_APP/images/SBAC_IconSprite.png

CSS

- $WEB_APP/css/sbac_style.css

JavaScript

- $WEB_APP/js/browser.js
- $WEB_APP/js/modernizer.js
- $WEB_APP/js/jquery.cookie.js
- $WEB_APP/console/js/sbacLogin.js
- $WEB_APP/console/js/sbacUMChangeUserPassword.js
- $WEB_APP/console/js/sbacEndUser.js
- $WEB_APP/console/js/sbacUMUserPasswordResetOptions.js
- $WEB_APP/console/js/jquery.base64.min.js

XML

- $WEB_APP/config/auth/default/services/sbac/html/LDAP.xml

.properties

- $WEB_APP/WEB-INF/classes/amConsole.properties
- $WEB_APP/WEB-INF/classes/amPasswordReset.properties
- $WEB_APP/WEB-INF/classes/amPasswordResetModuleMsgs.properties
- $WEB_APP/WEB-INF/classes/amPasswordResetModuleMsgs_en.properties
- $WEB_APP/WEB-INF/classes/amAuthUI.properties
- $WEB_APP/WEB-INF/classes/amAuthUI_en.properties

$WEB_APP = /opt/tomcat/webapps/auth

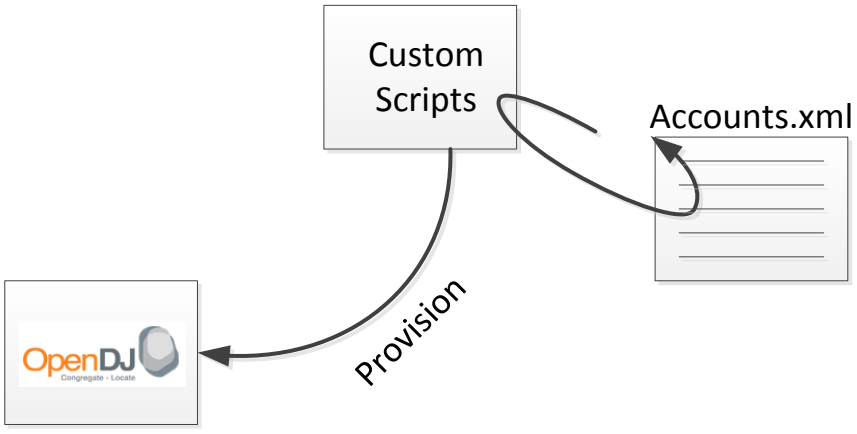
Note that while OpenAM 12 offers a new REST based XUI interface, the original SBAC design was built on OpenAM 10.1-Xpress. To more easily carry forward the customized files into OpenAM 12 the XUI interface has been disabled in the Smarter Balanced environment.

Account Management

All Create, Read, Update, and Delete operations (collectively called "CRUD" operations) affecting LDAP Server data will be initiated from custom processing scripts. The Smarter Balanced Test Registration component provides an export of account objects on a periodic basis. It places these objects into an XML data file (referenced generically as `Accounts.xml` in the diagram, below) and places that file into a dropbox folder on the server hosting the OpenDJ instance.

Note: The dropbox folder can be found on any of the OpenDJ instances at the following location: `/opt/dropbox`. This is the default home directory of the `dropbox` user - the account used for uploading the XML file to the OpenDJ server using SFTP.

The operating system has been configured to monitor the dropbox folder via an iNotify script. Once it detects that a new file has been uploaded, the `sbacProcessDataFile.pl` script reads the file for processing. The custom script will apply business logic (rules) to the data contained in the file, and the appropriate entries in OpenDJ will be created, updated, or deleted (collectively called "Provisioning") as demonstrated in the following diagram.



The format of the data file is XML ("Extensible Markup Language"); this allows for a relational hierarchy between data elements.

The remainder of this section further describes the data contained in the XML data file.

Unique Identifiers

The distinguished name attribute uniquely identifiers the entry in the OpenDJ server. When selecting an attribute to use to create the distinguished name, it is a best practice to select one that will not change for the life of the entry. This approach minimizes the impact to the database and any indexes that might be associated with those DNs. The UUID element uniquely identifies a user within the Smarter Balanced environment and will not be reused for other users. As such, this identifier is used to create a unique distinguished name (DN) within the OpenDJ Server as follows:

```
dn:
sbacUUID=bill.nelson@identityfusion.com,ou=People,dc=smarterbalanced,dc=org
```

Note: You cannot have two entries with the same DN in the OpenDJ Server or this will cause what is called a "naming clash". As such, it is essential that the Test Registration Component maintain unique UUID values for all Smarter Balanced users and these unique values are not reused.

The example shown above demonstrates the use of the email address as the sbacUUID. This is currently the case as the Test Registration component requires the user's email to be unique and not to change for the lifetime of the user's account. The Email element received from the Test Registration component is currently used to populate both the mail and uid attributes in the OpenDJ server. This indicates that the user's email address will be used to login to the SSO component and fulfils one of the original design considerations. A user's email address may change. This will update both their mail and their uid attributes but will have no effect on the distinguished name, as that is derived from the UUID element.

Attribute Mapping

The following table provides a mapping of attributes received from the Test Registration component to those in the LDAP Server. It should be noted that some source elements currently map to multiple destination attributes:

Source	Destination	REQ	Notes
UUID	sbacuuid	Y	MUST be unique throughout entire set of users.
UUID	dn	Y	dn: sbacuuid=UUID,ou=People,dc=smarterbalanced,dc=org
FirstName	givenName	Y	
LastName	sn	Y	
Email	mail	Y	
Email	uid	Y	Used as the user's credential for authentication.
Phone	telephoneNumber	N	
Role	N/A	N	An indication that a role is included in the user's data. If the role is detected then the individual role components are required values.
RoleID	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
Name	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
Level	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
ClientID	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.

Client	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
GroupOfStatesID	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
GroupOfStates	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
StateID	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
State	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
GroupOfDistrictsID	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
GroupOfDistricts	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
DistrictID	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
District	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
GroupOfInstitutionsID	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
GroupOfInstitutions	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
InstitutionID	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.
Institution	sbacTenancyChain	Y	A component of the sbacTenancyChain. Required if an only if a role has been included in the user's data.

Add Operations

Add operations must contain all the data necessary to create a user account in OpenDJ. This includes both core user information (First Name, Last Name, Email, and Phone Number) and the information necessary to create the tenancy chain (this is included in the `<Role>` element). Each `<Role>` element includes a unique ID; which identifies each role that the user possesses. This ID is unique for the life of the role and is used to accurately select the role during modifications or deletions of the role. The Testing Registration component maintains the role ID for the lifetime of the role.

The following is an example of a user who has two roles within the same tenancy chain.

```
<User Action="ADD">
    <UUID>bill.nelson@identityfusion.com</UUID>
    <FirstName>Sonja</FirstName>
    <LastName>Hubbard</LastName>
    <Email>bill.nelson@identityfusion.com</Email>
    <Phone>900-900-9000</Phone>
    <Role>
      <RoleID>23_848887</RoleID>
      <Name>BTC</Name>
      <Level>INSTITUTION</Level>
      <ClientID>3</ClientID>
      <Client>Utah</Client>
      <GroupOfStatesID />
      <GroupOfStates />
      <StateID>836813</StateID>
```

```
      <State>Ohio Department of Education</State>
      <GroupOfDistrictsID />
      <GroupOfDistricts />
      <DistrictID>836814</DistrictID>
      <District>Laurent Clerc National Deaf Education Center</District>
      <GroupOfInstitutionsID />
      <GroupOfInstitutions />
      <InstitutionID>848887</InstitutionID>
      <Institution>Laurent Clerc National Deaf Education Center</Institution>
    </Role>
    <Role>
      <RoleID>25_1043294</RoleID>
      <Name>Item Author</Name>
      <Level>INSTITUTION</Level>
      <ClientID>9968288</ClientID>
      <Client>Smarter Balanced</Client>
      <GroupOfStatesID>8820315</GroupOfStatesID>
      <GroupOfStates>Cascadia</GroupOfStates>
      <StateID>1326608</StateID>
      <State>CA</State>
      <GroupOfDistrictsID>2037212</GroupOfDistrictsID>
      <GroupOfDistricts>Central Region Association</GroupOfDistricts>
      <DistrictID>7062025</DistrictID>
      <District>Glendale Unified</District>
      <GroupOfInstitutionsID>2171081</GroupOfInstitutionsID>
      <GroupOfInstitutions>Main Street Schools</GroupOfInstitutions>
      <InstitutionID>4368641</InstitutionID>
      <Institution>Glendale Middle School</Institution>
    </Role>
</User>
```

Note: The order in which the Role appears in the user's data is not important; only that it is unique within that user's roles in the Test Registration component. That means that the Test Registration system needs to maintain the roles and tenancy for each user and associate that relationship with a unique Role ID.

An account creation action will generate an email to the person whose account has just been created in OpenDJ. The email will contain a randomly generated temporary passwords and a link to the user profile page in OpenAM. The user is required to change their password upon first login.

Modify Operations

Given the manner in which user records are maintained in the Training Registration component, AIR has determined that it is easier to provide a complete dump of a user's record rather than just the changes. As such, the format of a modify operation is similar to that of an add operation with the exception that a modify operation uses the `MOD` action in the `<User>` element as follows:

```
<User Action="MOD">
```

Existing entries in the OpenDJ server consist of attributes that should not be changed – even if the MOD operation includes this data. The following attributes will not be changed in existing user records (even if they are included in the data): `sbacUUID`, `inetUserStatus`, `userPassword`, or `objectClass`.

- A change to sbacUUID causes a change in the DN; therefore a moddn() operation is required directly on the OpenDJ server.

Note: A user's sbacUUID may also be changed through the data feed by specifying an action of DEL followed by an ADD. This essentially deletes the current user and adds them back with the correct sbacUUID. This operation should never be performed if the user has an active OpenAM session.

- A change to objectClass is uncessary as all objectclasses are already populated
- A change to the inetUserStatus is facilitated with the LOCK or UNLOCK operations
- The userPassword is set on entry creation or on password reset facilitated in the XML dump.
- The email/uid and sbacUUID attributes are currently the same, but this may not always be the case. Allow the sbacUUID to be decoupled from the email/uid values if necessary (hence they are updateable).

Account Locking and Unlocking Operations

Accounts can be locked or unlocked by specifying the LOCK or UNLOCK actions in the `<User>` element. If an account is locked, the user's `inetUserStatus` in the OpenDJ Server is set to "`Inactive`". OpenAM sees this value and prevents the user from logging in. If an account is unlocked, the `inetUserStatus` is set to "`Active`". This allows the user to once again log in to the OpenAM interface. OpenDJ will be configured to send the user an email indicating a change in the status of their account.

Locking an Existing Account

The following example demonstrates a user's account being locked.

```
<User Action="LOCK">
  <UUID>bill.nelson@identityfusion.com</UUID>
</User>
```

Note: Other than a change to the `inetuserstatus` attribute, the user's data remains untouched. Once the account is unlocked, they will be able to use the same data elements currently associated with their OpenDJ user account.

Unlocking an Existing Account

The following example demonstrates a user's account being unlocked.

```
<User Action="UNLOCK">
  <UUID>bill.nelson@identityfusion.com</UUID>
</User>
```

Delete Operations

Delete operations need only contain the unique identifier of the user object that is being deleted. This will cause the user object to be deleted from the OpenDJ Directory Server.

Note: Delete operations apply only to account objects, themselves, not to content contained within the object. For instance, role deletions are processed as a modify operation as they modify an existing account object.

The following is an example that indicates that the user identified by the UUID of **bill.nelson@identityfusion.com** should be deleted from the OpenDJ Directory Server.

```
<User Action="DEL">
  <UUID>bill.nelson@identityfusion.com</UUID>
</User>
```

Any pointers to this user's entry from another object in the OpenDJ Server will be removed as well. This is most commonly found when their DN is the value of a `uniqueMember` for a static group). This functionality is part of the referential integrity plug-in.

Password Reset Operations

A user's password may be reset by an SBAC Help Desk Administrator; this is initiated by the user calling a 1-888 number. In such cases, the password reset operation may be presented to the SSO environment within the data dump. When this occurs, the user's password will be reset to a temporary random value and emailed to them. The user is required to change their password upon first login using the temporary password.

The following is an example that indicates that the user identified by the UUID of **bill.nelson@identityfusion.com** has requested their password to be reset. The temporary password will be sent to the email address listed in the request.

```
<User Action="RESET">
  <UUID>bill.nelson@identityfusion.com</UUID>
  <Email>bill.nelson@identityfusion.com</Email>
</User>
```

Password Change Operations

A user's password may be changed to a known value by an SBAC Help Desk Administrator; this is initiated by the user calling a 1-888 number. In such cases, the password change operation may be presented to the SSO environment within the data dump. When this occurs, the user's password will be changed to the known (temporary) value but unlike the password reset operation, the password will not be emailed to the user. The user is required to change their password upon first login using the temporary password.

The following is an example that indicates that the user identified by the UUID of **bill.nelson@identityfusion.com** has requested their password to be changed by the Help Desk Administrator. This password is currently known by both the Help Desk Administrator and the user so it is not necessary to email the user their password.

```
<User Action="SETPWD">
  <UUID>bill.nelson@identityfusion.com</UUID>
  <Email>bill.nelson@identityfusion.com</Email>
  <Password>password123</Password>
</User>
```

Note: Passwords provided by Help Desk Administrators must adhere to the password policies set in the Directory Server.

Synchronization Operations

A SYNC operation is used when the generator of the XML data file questions the integrity of the data found on the directory server. This should only occur if the generator of the XML data believes it is out of synch with the Directory Server and wants to place the directory server in a known state.

This is accomplished by updating existing users' accounts with the data found in the XML data file. If the user's account does not exist, then new users will be created based on the data found in the XML data file. No attempt is made to detect users in the directory server that do not exist in the XML data file so this is not actually a 'true' synchronization event.

Note: The SYNC operation should be used sparingly. Processing of extremely large files is akin to new data load and may take time to process. This should not impact users with existing OpenAM sessions but use of the SYNC operation will most likely be masking problems with the program responsible for generating the XML file. This option should really be used as a last resort.

The following is an example that indicates that the record of the user identified by the UUID of **bill.nelson@identityfusion.com** will be replaced by the information found in the XML file. This is equivalent to the MOD operation. If the user does not currently have a record in the OpenDJ server, one will be created for them using the information contained in the file. This follows the same process as the ADD operation.

```
<User Action="SYNC">
    <UUID>bill.nelson@identityfusion.com</UUID>
    <FirstName>Bill</FirstName>
    <LastName>Nelson</LastName>
    <Email>bill.nelson@identityfusion.com</Email>
    <Phone/>
    <Role>
      <RoleID>848887</RoleID>
      <Name>BTC</Name>
      <Level>INSTITUTION</Level>
      <ClientID>3</ClientID>
      <Client>Utah</Client>
```

```
      <GroupOfStatesID />
      <GroupOfStates />
      <StateID>836813</StateID>
      <State>Ohio Department of Education</State>
      <GroupOfDistrictsID />
      <GroupOfDistricts />
      <DistrictID>836814</DistrictID>
      <District>Laurent Clerc National Deaf Education Center</District>
      <GroupOfInstitutionsID />
      <GroupOfInstitutions />
      <InstitutionID>848887</InstitutionID>
      <Institution>Laurent Clerc National Deaf Education Center</Institution>
    </Role>
 </User>
```

XML Schema

The following schema will be used to define the user extract from the Test Registration Component.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Users">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="User" maxOccurs="unbounded" minOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="UUID" maxOccurs="1" minOccurs="1"/>
              <xs:element type="xs:string" name="FirstName" maxOccurs="1"
minOccurs="1"/>
              <xs:element type="xs:string" name="LastName" maxOccurs="1"
minOccurs="1"/>
              <xs:element type="xs:string" name="Email" maxOccurs="1" minOccurs="1"/>
              <xs:element type="xs:string" name="Phone" maxOccurs="1" minOccurs="1"/>
              <xs:element name="Role">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="RoleID" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="Name" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="Level" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="ClientID" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="Client" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="GroupOfStatesID" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="GroupOfStates" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="StateID" maxOccurs="1"
minOccurs="1"/>
```

```
                    <xs:element type="xs:string" name="State" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="GroupOfDistrictsID"
maxOccurs="1" minOccurs="1"/>
                    <xs:element type="xs:string" name="GroupOfDistricts" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="DistrictID" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="District" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="GroupOfInstitutionsID"
maxOccurs="1" minOccurs="1"/>
                    <xs:element type="xs:string" name="GroupOfInstitutions"
maxOccurs="1" minOccurs="1"/>
                    <xs:element type="xs:string" name="InstitutionID" maxOccurs="1"
minOccurs="1"/>
                    <xs:element type="xs:string" name="Institution" maxOccurs="1"
minOccurs="1"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:simpleType name="Actions">
              <xs:restriction base="xs:string">
                <xs:enumeration value="ADD"/>
                <xs:enumeration value="MOD"/>
                <xs:enumeration value="DEL"/>
                <xs:enumeration value="LOCK"/>
                <xs:enumeration value="UNLOCK"/>
                <xs:enumeration value="SYNC"/>
                <xs:enumeration value="RESET"/>
              </xs:restriction>
            </xs:simpleType>
            <xs:attribute type="xs:string" name="Action" use="required"
type="Actions"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The rules contained in this schema can be described as follows:

1. Each element must appear on a separate line.
2. There MUST be 1 (and only 1) <Users> element. This element contains all individual users contained in the dump.
3. There MUST be at least 1 <User> element, but there is no upper limit to the number of <User> elements contained in the file.
4. Every <User> element MUST have 1 (and only 1) "Action" attribute.

5. "Action" attributes MAY be one of the following values: ADD, MOD, DEL, LOCK, UNLOCK, SYNC, and RESET.
6. Within the body of the <User> element, there must be 1 (and only 1) of each of the following elements: <UUID>, <FirstName>, <LastName>, <Email>, and <Phone>.
7. Within the body of the <User> element, there may be 0 or more <Role> elements.
8. Each <Role> element specified must have 1 (and only 1) of each of the following elements: <RoleID>, <Name>, <Level>, <ClientID>, <Client>, <GroupOfStatesID>, <GroupOfStates>, <StateID>, <State>, <GroupOfDistrictsID>, <GroupOfDistricts>, <DistrictID>, <District>, <GroupOfInstitutionsID>, <GroupOfInstitutions>, <InstitutionID>, and <Institution>.
9. All elements are defined as type string to allow flexibility of data values.

Test Files

Emails are generated by the sbacProcessDataFile.pl script during ADD operations. This is not desirable while testing the SSO solution, however, and as such, a method of detecting and altering the processing of test files has been devised. If the name of the XML data file contains the string "testfile", then that file will be treated as a non-production file and no email will be sent to end-users. Additionally, the password for user contained in that file will be set to "password".

Examples of filenames that would be recognized as test files include testfile.xml, testfile-12182013.xml, and studentData.testfile, the only requirement is that the string, "testfile" appear somewhere in the name of the file.

Note: It is highly advisable that for any testing files containing ADD actions that a corresponding test file containing a DEL action also be created for any users contained in the ADD test file. This is necessary to remove test accounts from the OpenDJ server before going into production.

Log Files

The results of the user data file processing is contained in log files that may be found in the /opt/scripts/logs folder. New log files are created on a daily basis; the names of the log file are as follows: sbaclogfile-*yyyymmdd* (where *yyyy* is the four digit year, *mm* is the two digit month, and *dd* is the two digit day).

These files contain useful information for debugging potential problems observed during the user data file processing. The following demonstrates the successful processing of the Users.xml data file. The file contained 7 user objects of which 6 users were added and one object generated an error. The results of the error message can be seen in the WARN message.

```
[12/23/2013:14:15:38] INFO "SBAC user processing initiated."
[12/23/2013:14:15:38] INFO "Input file = /opt/dropbox/Users.xml"
```

```
[12/23/2013:14:15:40] WARN "An error occurred while processing MOD on
sbacUUID=Schoolassoc@air.org,ou=People,dc=smarterbalanced,dc=org. The server cannot
find an object specified in the request
[12/23/2013:14:15:41] INFO "Results: Total(7); Added(6); Modified(0); Deleted(0);
Reset(0); Locked(0); Unlocked(0); Synchronized(0); Errors(1)."
[12/23/2013:14:15:41] INFO "SBAC user processing completed. Elapsed Time: 3 Seconds"
[12/23/2013:14:15:41] INFO "******************************"
[12/23/2013:14:15:41] INFO "/opt/dropbox/Users.xml has been moved to
/opt/scripts/sbacXMLFiles/Users.xml-20131223T14_15_38."
```

The format of the log file is as follows:

`[`*`TIMESTAMP`*`]  `*`MESSAGETYPE`*`  "`*`MESSAGE`*`"`

The *`MESSAGETYPE`* will be one of the following:

- `INFO` - informational messages
- `WARN` - warning messages; processing of the data file will continue
- `ERROR` - error message; processing of the data file will terminate when an error occurs.

The following is an example of an error that warranted early termination of file processing:

```
[12/26/2013:20:26:17] INFO "SBAC user processing initiated."
[12/26/2013:20:26:17] INFO "Input file =
/opt/dropbox/USER_EXPORT_OFFLINE_D122613T152130.xml"
[12/26/2013:20:26:17] ERROR "Invalid action detected ()!  Corrupt XML file."
```

You can extend the amount of information sent to the log file by setting the `$extendedLoggingd` variable to a value of 1 in the `sbacProcessDataFile.pl` script.

File Processing Callback

The `sbacProcessDataFile.pl` script can be configured to send an HTTP response to the Test Registration server once the processing of the XML file has completed.

Note: The `$sendHTTPResponse` variable enables or disables the response. This response is sent by default as the variable is set to a value of `1`. The `$httpResponseServer` variable in the `sbacProcessDataFile.pl` script contains the address of the server where the response will be sent.

The response is an XML document consisting of the following information:

- **DateProcessed:** The date the file was processed.
- **FileName:** The name of the file processed.
- **DateStarted:** The timestamp when the processing was started
- **UUID:** The SBAC UUID that incurred the error
- **Error:** The error that was detected on the UUID

- **TotalRecordsProcessed:** The number of records processed (including records that experienced errors)

The response is sent to the Test Registration server using the POST method.

The following is an example of a file processing callback where no errors where experienced during file processing:

```
<OpenamACKStatus>
    <DateProcessed>2014-02-05T20:46:50</DateProcessed>
    <FileName>USER_EXPORT_OFFLINE_D020514T154604.xml</FileName>
    <DateStarted>2014-02-05T20:46:46</DateStarted>
    <ErrorsWithUID/>
    <TotalRecordsProcessed>8</TotalRecordsProcessed>
</OpenamACKStatus>
```

The following is an example of a file processing callback where errors where experienced. The `sbacProcessDataFile.pl` script received three requests to add users that already exist in the OpenDJ server.

```
<OpenamACKStatus>
    <DateProcessed>2014-02-04T18:52:24</DateProcessed>
    <FileName>USER_EXPORT_OFFLINE_D020414T135141.xml</FileName>
    <DateStarted>2014-02-04T18:52:22</DateStarted>
    <ErrorsWithUID>
        <UUIDError>
            <UUID>lpetroski@rsd13.org</UUID>
            <Error>The client attempted to add an entry that already exists. This can
occur as a result of   * An add request was submitted with a DN that already exists
* A modify DN requested was submitted, where the requested new DN already exists   *
The request is adding an attribute to the schema and an attribute with the given OID
or name already exists </Error>
        </UUIDError>
        <UUIDError>
            <UUID>rfielding@rsd13.org</UUID>
            <Error>The client attempted to add an entry that already exists. This can
occur as a result of   * An add request was submitted with a DN that already exists
* A modify DN requested was submitted, where the requested new DN already exists   *
The request is adding an attribute to the schema and an attribute with the given OID
or name already exists </Error>
        </UUIDError>
        <UUIDError>
            <UUID>anathan@nbfacademy.org</UUID>
            <Error>The client attempted to add an entry that already exists. This can
occur as a result of   * An add request was submitted with a DN that already exists
* A modify DN requested was submitted, where the requested new DN already exists   *
The request is adding an attribute to the schema and an attribute with the given OID
or name already exists </Error>
        </UUIDError>
    </ErrorsWithUID>
    <TotalRecordsProcessed>8</TotalRecordsProcessed>
</OpenamACKStatus>
```

Note: In addition to the callback, the XML response is included in the `sbacProcessDataFile.pl` log file.

Processed Files

Files that have completed processing are moved to the `/opt/scripts/sbacXMLFiles` folder with a timestamp appended at the end of the file as follows:

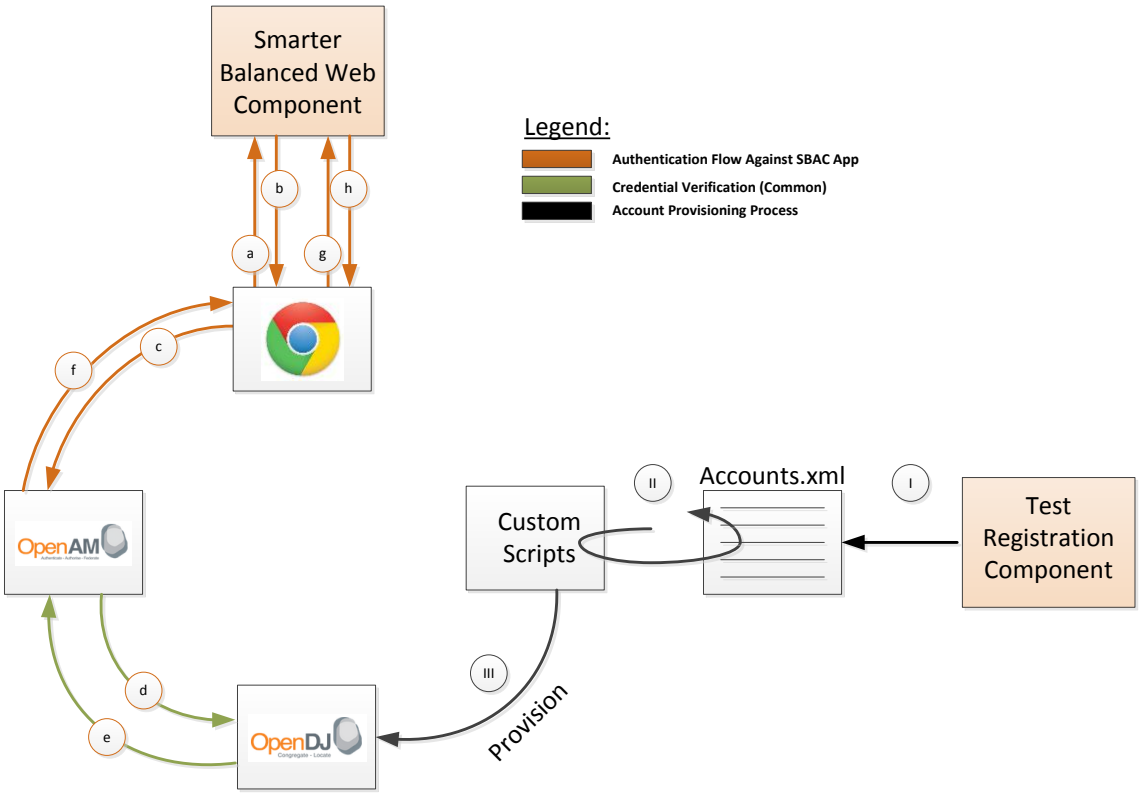
```
[12/23/2013:14:15:41] INFO "/opt/dropbox/Users.xml has been moved to
/opt/scripts/sbacXMLFiles/Users.xml-20131223T14_15_38."
```

High Level Architecture

There exists a synergistic relationship between the components that make up what has been called the SSO component of the Smarter Balanced environment.

Note: In actuality, the SSO component is comprised of two separate products: an identity store (ForgeRock OpenDJ) and a Web single sign-on solution (ForgeRock OpenAM).

The following diagram demonstrates the relationship between these products. An explanation of the interactions between each component follows the diagram.



Interaction between SSO Components

Interaction between the various components can be described in the following processes.

Account Provisioning Process

I. The Test Registration component exports account modifications to the `Accounts.xml` file. This file is placed in a dropbox folder on the OpenDJ Server using secure file transfer protocol (SFTP).

II. The Operating System uses iNotify on the OpenDJ Server to monitor the dropbox folder for any new files. Once a new Accounts.xml file is detected iNotify launches a custom script to process the changes contained in the file (see Account Management for more details).

III. Any changes appearing in the `Accounts.xml` file are reflected in the OpenDJ Directory Server. The user's account-specific data includes the tenancy chain which provides authorization information to Smarter Balanced client components.

Authentication and Authorization Process (Client Component)

a. The user launches a browser and attempts to access a Smarter Balanced client component.

b. Client components are protected by OpenAM policy. If the identity of the user is not known, they are sent an HTTP redirect status to an authentication interface.

c. The browser follows the redirect to the login screen of the authentication interface (OpenAM). The user enters their credentials (email address + password).

d. The credentials are validated against the authentication database (OpenDJ).

e. The results of the authentication attempt are sent back to OpenAM.

f. OpenAM generates a cookie if using policy agents or a SAML assertion if implementing federation. In federated environments, the SAML assertion will include the attributes necessary for the client component to authorize the user within the application. This includes the unique ID, the user's first and last name, and the tenancy chain. It passes the cookie (or assertion) back to the browser with a redirect (or an auto-form if using SAML).

g. In policy agent environments, the browser provides the cookie to the policy agent protecting the client component. The policy agent then connects with the OpenAM Server where it will determine if the user is permitted access to the client component. This is accomplished by way of a policy evaluation. If the evaluation is successful, then OpenAM will provide the user's unique ID, their first and last name, and their tenancy chain to the policy agent along with the results of the policy evaluation. The policy agent inserts these attributes into header fields that are passed to the client application. In cases where SAML is utilized, the assertion already contains this information and the assertion is provided directly to the Smarter Balanced client component.

Note: Client components must be "SAML aware" in order to process assertions received from the browser.

---

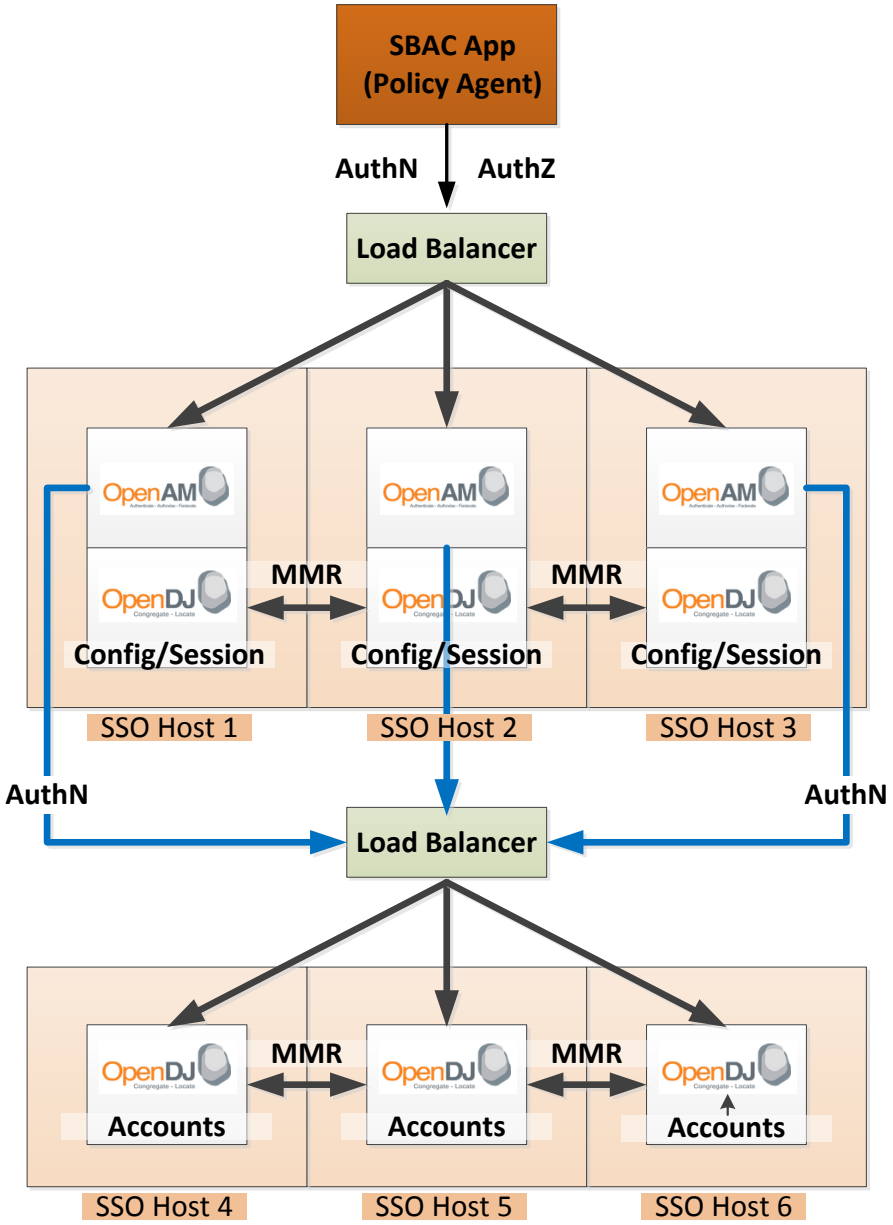
h. The Smarter Balanced client component reviews the content given to it (headers or SAML assertion). It pays particular attention to the information contained in the tenancy chain

attribute(s) to determine what the user is allowed to access within the application based on their role within a particular tenancy chain. If the user is permitted access, the web page content is returned to the user. If they are not, then an unauthorized message is returned to the user.

Note: The `sbacTenancyChain` is a multivalued attribute (see Smarter Balanced-Specific Object Classes). That means that we will send the client component every role/chain relationship that is stored in the user's object. The client component can search for particular roles as well as the scope of the role (the tenancy chain) and then determine what to allow/deny based on what it finds.

High Availability

The SSO Component is critical as it is the sole point of authentication and authorization within the Smarter Balanced environment. As such, the OpenAM and OpenDJ instances must be highly available at all times. The following diagram provides an overview of a highly available SSO Component for the Smarter Balanced environment.



This architecture includes the following principals:

- The SSO Component is comprised of the following six (6) Rackspace instances:
  - SSO Host1: OpenAM with embedded OpenDJ Configuration/Session Store
  - SSO Host2: OpenAM with embedded OpenDJ Configuration/Session Store
  - SSO Host3: OpenAM with embedded OpenDJ Configuration/Session Store
  - SSO Host4: OpenDJ Identity and Credential Data Store
  - SSO Host5: OpenDJ Identity and Credential Data Store
  - SSO Host6: OpenDJ Identity and Credential Data Store

- The OpenAM and OpenDJ instances are installed on separate SSO Hosts. Each instance is duplicated to provide a highly available environment.
- The Web interfaces for OpenAM are accessed through a load balancer. The load balancer is configured to maintain a "sticky session" between clients and the OpenAM server in which the instance first established its session.
- SSL is terminated at the load balancer for inbound traffic to the OpenAM server.
- The load balancers expose standard ports to the outside world; the OpenAM and OpenDJ instances are running on non-standard ports behind the load balancer.
- OpenAM communicates to the OpenDJ servers for authentication purposes through the load balancer.
- OpenDJ data is synchronized between server instances through a multi-master replication agreement.
- The overall architecture is designed in such a way as to grow horizontally by adding new SSO Host instances as necessary.

The SSO Component will be distributed to Smarter Balanced clients in a grouping of servers as indicated above. This grouping will be referred to as an SSO "pod".

Security

User Accounts

The following accounts are used within the SBAC SSO component environments. Passwords for these accounts are available to necessary personnel on an as needed basis.

Account Name	Environment	Level	Purpose
opendj	Linux	Operating System	The OpenDJ application runs as this user.
dropbox	Linux	Operating System	
openam	Linux	Operating System	The OpenAM application runs as this user.
root	Linux	Operating System	The iNotify script runs as the root user.
cn=SBAC Admin	OpenDJ	Application	This is the default administrative user for the OpenDJ application.
amadmin	OpenAM	Application	This is the default administrative user for the OpenAM application.
user accounts	OpenDJ	Application	Each user in the SBAC environment has an account in the OpenDJ server. The credentials for each user include their email address and their password.

Ports

The following table describes the ports used in the SSO components.

Application	Port	SSL	Instance	Description
opendj	1389	N	OpenDJ	Port used for LDAP communication (standard server).
opendj	4444	Y	OpenDJ	Port used for LDAP communication (administration server).

sftp	22	Y	OpenDJ	Port for uploading XML files to OpenDJ instance.
tomcat	8080	N	OpenAM	Port used for HTTP communication to the OpenAM server.
opendj	50389	N	OpenAM	Port used for LDAP communication to the OpenAM Configuration server (an embedded OpenDJ instance)
opendj	4444	Y	OpenAM	Port used for LDAP communication to the administrative interface of the OpenAM Configuration Server.

Network Communication

All traffic to OpenAM is directed through an Internet accessible load balancer. Direct access to individual OpenAM instances is not allowed and if attempted, should be routed back through the load balancer. There is no access to OpenDJ instances from the Internet. Communication to OpenDJ instances is facilitated through the load balancer, but connectivity is only available from servers behind the firewall (i.e. OpenAM). Communication up to the load balancer is encrypted with SSL. Communication behind the load balancer, however, is not encrypted.

SBAC User Password Policy

SBAC users must adhere to the SBAC User Password Policy as defined in OpenDJ. This policy has the following properties:

Password Property	Password Value	Notes
Password Storage Schema	Salted SHA-1	Passwords will be stored in a one way hashing algorithm using the Salted SHA-1 algorithm. The salt is an 8 bytes (64-bit) random string and is used with the password to produce the 20 bytes message digest.
Force Change on Add	True	Users will be forced to change their password the first time they log in to the SBAC Portal.
Minimum Password Length	6	A user's password must consist of at least 6 characters.
Force Change on Reset	True	Users will be forced to change their passwords the next time they login after a password reset has been performed. Note: A password reset is defined as a password change operation performed by anyone other than the user, themselves.
Password Field	userpassword	This is the attribute used for storing the user's password in the OpenDJ server.

Passwords policies are automatically assigned to individuals based on the following rules:

1. The user must exist beneath the `ou=people,dc=smarterbalanced,dc=org`
2. The user must have the `sbacPerson` object class.

Users meeting this criteria will be assigned a "virtual attribute" of `ds-pwp-password-policy-dn` with a value of `cn=SBAC User Password Policy,cn=Password Policies,cn=config`. This utilizes a sub-entry method where password policies can be individually assigned to users. The fact that we are auto assigning this attribute based on the criteria above ensures that policies are enforced universally.

Security Questions

Users who have forgotten their passwords may respond to a security question in order to regain access to their accounts. Security questions are not maintained as part of the SBAC Password Policy; instead they are managed by

OpenAM and stored in the `iplanet-am-user-password-reset-question-answer` attribute for each user. The value of this attribute consists of an encrypted string that contains the following components:

- The question identifier
- The user's answer
- An indicator to specify whether the question is enabled or not

The following default security questions have been defined within the OpenAM server:

1. In what city or town was your first job?
2. What is your birth city?
3. What is the name of your first pet?

Users are not allowed to define their own security questions.

Appendix A - Scripts for Generating Sample Data

The `sbacGenerateSampleData.pl` script has been added to generate sample data for the SBAC environment. Its usage is:

`./sbacGenerateSampleData.pl` *`numberOfEntries`*

Where *`numberOfEntries`* is the number of XML user objects that you wish to generate for testing purposes. The execution of this script will produce the following three files (NOTE: *`XXX`* refers to the *`numberOfEntries`* produced when running the script).

- `add`*`XXX`*`entries.testfile` - An XML file containing *`XXX`* user objects ready to be added to the SBAC environment.
- `del`*`XXX`*`entries.testfile` - An XML file containing *`XXX`* user objects ready to be deleted from the SBAC environment. The sbacUUID attributes in this file correspond to those contained in the `addXXXentries.testfile`.
- `del`*`XXX`*`entries.ldif` - A file consisting of DNs of all users contained in the `addXXXentries.testfile`. This file is used as input into the `sbacBulkDeleteUsers.sh` script.

The following provides an overview of the use of these scripts:

Sample Data Creation

Use the script to generate 100 user entries as follows:

```
root@sbacssotest4:/opt/scripts/sampleData# ./sbacGenerateSampleData.pl 100

Generating sample data (100 entries).

Processing:
..........................................................................................
..............

Processing completed. 100 entries created.  Elapsed Time: 0 Seconds
```

The `sbacGenerateSampleData.pl` script reads data from the following files to produce random entries:

- `first.names` - list of user first names
- `last.names` - list of user last names
- `roles` - list of roles used in building a tenancy chain
- `states` - list of states used in building a tenancy chain
- `districts` - list of districts used in building a tenancy chain
- `schools` - list of schools used in building a tenancy chain

You can add additional data to these files to meet your needs or create additional files to support your environment. If you create additional files, they must be added to the `sbacGenerateSampleData.pl` script in the appropriate locations.

When you execute the script, you will see the following three files created in the folder:

- `add100entries.testfile`
- `del100entries.testfile`
- `del100entries.ldif`

Note: the text, "testfile" tells the data processing script that this is test data. As such, no attempt will be made to send email to these users during processing.

Adding Sample Users to the SBAC Environment

To add the users contained in the `add100entries.testfile` to the SBAC environment, copy the file to the `/opt/dropbox` folder as follows:

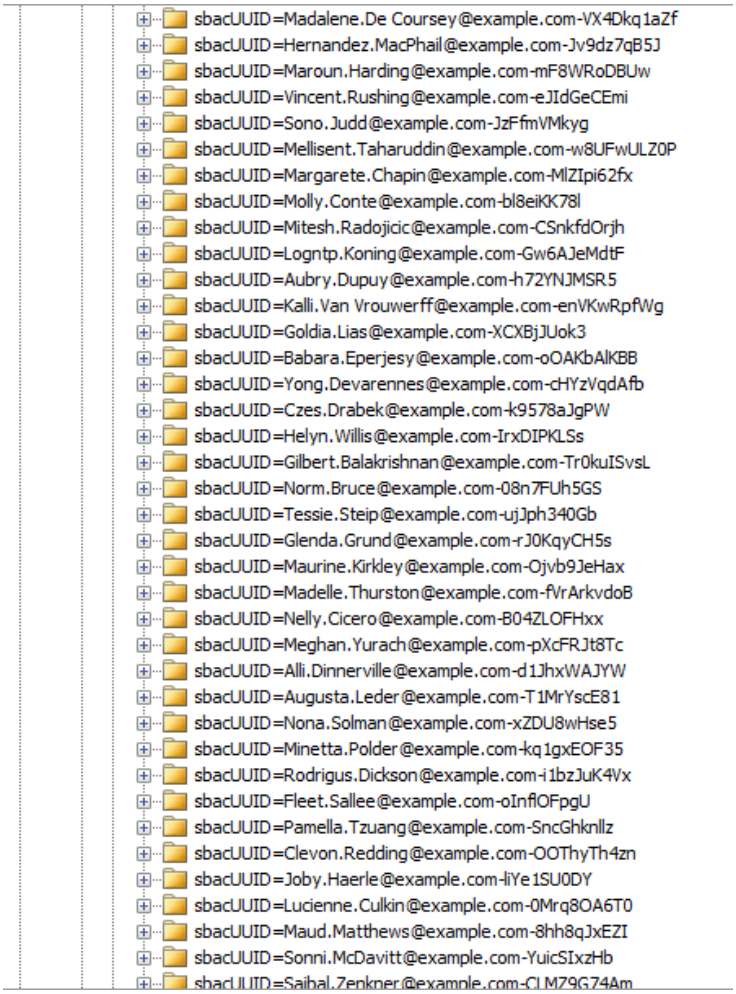
```
root@sbacssotest4:/opt/scripts/sampleData# cp add100entries.testfile /opt/dropbox/
```

You can verify that the users were added successfully by reviewing the contents of the current log file.

Note: log files may be found in the `/opt/script/logs` folder. Log files names consist of the following format: `sbaclogfile-`*`yyyymmdd`*. (where *`yyyy`* is the 4 digit year, *`mm`* is the 2 digit month, and *`dd`* is the two digit day). Logs are produced on a daily basis. The current log file will have today's date on it.

```
[12/31/2013:01:54:12] INFO "*******************************"
[12/31/2013:02:07:02] INFO "SBAC user processing initiated."
[12/31/2013:02:07:02] INFO "Input file = /opt/dropbox/add100entries.testfile"
[12/31/2013:02:07:02] INFO "This file is used for testing only; no email will be sent
to users"
[12/31/2013:02:07:03] INFO "100 user objects have been processed in the XML file."
[12/31/2013:02:07:03] INFO "/opt/dropbox/add100entries.testfile has been moved to
/opt/scripts/sbacXMLFiles/add100entries.testfile-20131231T02_07_02."
[12/31/2013:02:07:03] INFO "Administrator notified of run results
(bill.nelson@identityfusion.com)"
[12/31/2013:02:07:03] INFO "Results: Total(100); Add(100); Mod(0); Del(0); Reset(0);
Lock(0); Unlock(0); Synch(0); Errors(0)."
[12/31/2013:02:07:03] INFO "SBAC user processing COMPLETED. Elapsed Time: 1 Seconds"
[12/31/2013:02:07:03] INFO "*******************************"
```

In reviewing the log file, you will be able to determine the name of the file that was processed, and the results of the processing. If you have an LDAP client available, you can review the actual data in the directory server as follows:



Deleting Sample Users from the SBAC Environment (SBAC Method)

To delete the users contained in the `del100entries.testfile` from the SBAC environment, copy the file to the `/opt/dropbox` folder as follows:

```
root@sbacssotest4:/opt/scripts/sampleData# cp del100entries.testfile /opt/dropbox/
```

You can verify that the users were deleted successfully by reviewing the contents of the current log file.

```
[12/31/2013:02:13:50] INFO "******************************"
[12/31/2013:02:14:22] INFO "SBAC user processing initiated."
```

```
[12/31/2013:02:14:22] INFO "Input file = /opt/dropbox/del100entries.testfile"
[12/31/2013:02:14:22] INFO "This file is used for testing only; no email will be sent
to users"
[12/31/2013:02:14:23] INFO "100 user objects have been processed in the XML file."
[12/31/2013:02:14:23] INFO "/opt/dropbox/del100entries.testfile has been moved to
/opt/scripts/sbacXMLFiles/del100entries.testfile-20131231T02_14_22."
[12/31/2013:02:14:23] INFO "Administrator notified of run results
(bill.nelson@identityfusion.com)"
[12/31/2013:02:14:23] INFO "Results: Total(100); Add(0); Mod(0); Del(100); Reset(0);
Lock(0); Unlock(0); Synch(0); Errors(0)."
[12/31/2013:02:14:23] INFO "SBAC user processing COMPLETED. Elapsed Time: 1 Seconds"
[12/31/2013:02:14:23] INFO "*******************************"
```

Deleting Sample Users from the SBAC Environment (Alternate Method)

To delete the users contained in the `del100entries.ldif` from the SBAC environment, run the following command:

```
root@sbacssotest4:/opt/scripts/sampleData# ./sbacBulkDeleteUsers del100entries.ldif
```

Use of this script has the advantage of removing users from the directory server more quickly as it bypasses the SBAC user processing scripts entirely.