Initialization and Configuration

OpenSAML Initialization and Configuration

This topic describes how the OpenSAML library handles initialization of components and how configuration information is stored and retrieved.

Service Provider API

The library initialization and configuration makes use of the Java Service Provider API in several places. For more information see:

- Service provider documentation in the Java JAR File specification
- The API documentation for java.util.ServiceLoader

Library Initialization Service

Various components of the OpenSAML library must be initialized before they can be used. Since the library is in itself modular, the initialization process takes a modular approach use the Java Service API.

The 2 primary components with which to be familiar are:

- org.opensaml.core.config.Initializer
- org.opensaml.core.config.InitializationService

org.opensaml.core.config.Initializer

This interface defines a component which initializes some aspect of the library which requires initialization. This could entail invoking the initialization process for some third-party library that has its own initialization process, such as Apache XML Security (Santuario) and Velocity, as well as internal OpenSAML components which require initialization prior to use, such as the XMLObjectProviderRegistry.

Specific implementations of the Initializer interface should typically be registered within the environment via the Service Provider API, by declaring them as a service provider implementation in the owning module's /META-INF/services/org.opensaml.core.config.Initializer. In this way they will automatically be resolved and invoked by the InitializationService, described below.

org.opensaml.core.config.InitializationService

The InitializationService serves as the service entry point for the API defined by the Initializer interface. When its initialize() method is called, it resolves all registered implementations of the Initializer service, and invokes their init() method in turn. There is no implied ordering to the order in which the Initializer implementations are resolved.

The InitializationService initialize() should be invoked once at application startup time.

Library Configuration Service

The OpenSAML library exposes various bits of configuration state. Since the library is modular, the various kinds of configuration which can be stored are not known in advance. Therefore the library makes use of a configuration service which exposes type-safe configuration beans in a flexible and extensible manner.

The 3 primary components with which to be familiar are:

- org.opensaml.core.config.ConfigurationService
- org.opensaml.core.config.Configuration
- org.opensaml.core.config.ConfigurationPropertiesSource

org.opensaml.core.config.ConfigurationService

The ConfigurationService serves as the service entry point for registering and retrieving configuration beans. Configuration beans are simply plain old Java objects (POJO's) and are not required to implement any specific interface. They are registered and retrieved from the ConfigurationService in a type-safe manner using their associated Class instance as the index into the service.

The ConfigurationService stores the registered beans using an instance of org.opensaml.core.config.Configuration. The implementation of this interface that is used is determined by resolving it using the Service Provider API against an implementation declared via /META-INF/services/org.opensaml.core.config.Configuration. At most 1 implementation should be declared within the environment (effective classpath). If more than 1 is registered, only the first one resolved is used. If no implementation is resolvable using the Service Provider API, then a default implementation of org.opensaml.core.config.provider.MapBasedConfiguration is used, which uses internal static Map-based stored for bean registration.

The ConfigurationService also exposes a configuration Properties set, which is used to configure the configuration and initialization framework itself. The Properties set is obtained from a ConfigurationPropertiesSource, described below. The ConfigurationPropertiesSource effectively used is resolved via the Java Service Provider API against implementations declared via /META-INF/services/org.opensaml.core.config.ConfigurationPropertiesSource. Multiple implementations may be registered. The effective Properties set used is the first non-null Properties instance resolved.

The ConfigurationService itself makes use of the following properties:
- `opensaml.config.partitionName` - this determines the logical configuration partition name to use when registering and retrieving beans.

The Properties set is exposed publicly on the ConfigurationService, allowing Initializer implementations to use properties here if they choose. Other components might make use of these properties, but this use is discouraged. Such components should instead use component-specific configuration beans registered with the ConfigurationService.

Use of configuration properties is optional. If no Properties set is resolved, the ConfigurationService will use internal defaults for properties.

`org.opensaml.core.config.Configuration`  
This interface defines a component which is responsible for actually implementing the bean registration and retrieval strategy used by the ConfigurationService. An implementation might do this using any of a variety of strategies, such as simple internal in-memory Map, use of JNDI or use of context-specific ThreadLocal storage. This allows flexibility to handle different deployment environments, with different classloader layouts (e.g. hierarchical vs flat) and behavior (e.g. parent-first vs. self-first), and deployment of the library in either an application-specific classloader or in a shared classloader.

The Configuration interface also supports the notion of a configuration "partition". This allows multiple "configuration contexts" to live within the ConfigurationService simultaneously.

`org.opensaml.core.config.ConfigurationPropertiesSource`  
This interface defines a component which supplies a Java Properties instance which is used to influence the behavior of the configuration and initialization framework itself. The ConfigurationService may make use of this Properties set internally, and also makes it available for use by Initializers.

An implementation of this interface may use a variety of strategies for resolving the Properties set, such as a classpath resource, a file resource or retrieving from a ThreadLocal. The implementation used will determine the "scope" of the configuration context in effect. For example, a classpath or file-based implementation will be relatively broad, as where a ThreadLocal implementation can supply a different set of properties on a per-Thread basis, determined in an application-specific manner. As with the Configuration interface, this allows flexibility to handle different deployment environments and scenarios.

**Deployment Scenarios**

TODO