SubjectCanonicalization

Error rendering macro 'toc' : [com.ctc.wstx.exc.WstxLazyException] com.ctc.wstx.exc.WstxParsingException: Undeclared namespace prefix "xlink" (for attribute "href") at [row,col {unknown-source}]: [7,167]

Overview

Various profiles and subsystems within the IdP, especially including the authentication process, need to be able to turn a potentially complex representation of a subject into a canonical "string" form that represents the principal internally and consistently. In particular, it is a property of this canonical form that two non-equal forms **MUST** refer to a different subject.

In many cases, this is a very simple, or at least self-contained, process. But for generality, and in particular to support the ability to leverage the flexibility of the AttributeResolver components, the Subject Canonicalization process is implemented as a subsystem that runs a Spring Web Flow excuted against a context tree.

The c14n subflow operates on a **SubjectCanonicalizationContext** child context, which contains a Java Subject property to operate on. The Subject will contain one or more Java Principal subtypes that the subflow must be coded to look for and operate on to produce a result.

Upon completion, the subflow populates the context with either a resulting principal name or signals an error event, in most cases resulting in additional subflows being attempted, or ultimately failing altogether.

There are at present two uses for this subsystem:

1. Normalizing the authenticated Subject into a simple username (referred to as "post-login" canonicalization, see also AuthenticationConfiguration)
2. Mapping a SAML 1 `<NameIdentifier>` or SAML 2 `<NameID>` element into a simple username (referred to as NameID consumption, see also NameIDConsumptionConfiguration)

Even though these are wildly different use cases, the same basic process is used to orchestrate the work. The difference arises from the specific subflows used by the master c14n subflow to do the actual work. The design will accomodate any future use cases in which something that's not necessarily a (single) string has to be turned into a username that is a string to match against the internal representation of users in an organization's systems.

Subject Canonicalization Process

The master c14n subflow is modeled after the master Authentication subflow in that there can be a number of different subflows described by a **SubjectCanonicalizationFlowDescriptor**. The only real information of interest there is a predicate that can control the conditions under which the flow will execute, usually by examining the Subject at hand to see if there is compatible work for the flow to do.

Much like the authentication flow, a set of potential c14n subflows is populated by an action into the **SubjectCanonicalizationContext**, and then a selection action iterates over them looking for a candidate subflow to run. Specific events can result in a retry of alternative subflows until success or a fatal error occurs, or no more subflows are available to try.

Subject Canonicalization Context Tree

The layout of the contexts used during c14n follows:
Graphviz output could not be displayed here.

Obviously, this is very simple. The input and output of the c14n flow are both carried within a single child context. There certainly are additional contexts present in the tree depending on which c14n use case is involved. For example, during authentication the tree will look as the Authentication page describes, but most of the time the extra information isn't relevant.

Programming Guide to Extending Canonicalization

Extending the c14n features of the IdP generally involves creating a custom c14n flow for the system to execute. There is virtually no limit on what such a flow can do, as it has full control of the IdP once it has been run, though it should be noted that for some c14n use cases there may be inherent constraints such as the lack of a browser client.

A c14n flow is a subflow that is assigned a flow ID that starts with "c14n/" and is further defined to the system with a bean of type **net.shibboleth.idp.authn.SubjectCanonicalizationFlowDescriptor** in a list in *conf/c14n/subject-c14n.xml*.

> While you may deliver a custom flow in a relatively "drop-in", self-contained jar, you **MAY NOT** manipulate the state of the IdP at

runtime to install the necessary descriptor bean because it is impossible to guarantee that your modification will take place early enough to be seen by other objects in the system. There is no publically supported mechanism to extend any of the beans defined inside the "root" web app context, and so you **MUST** rely on the deployer making the necessary adjustments to define custom flows to the system via the associated type of FlowDescriptor.

Internal Contract

C14n flows must interact with the system by accessing and mutating the context tree in specific ways.

The input contract is as follows:

1. An **SubjectCanonicalizationContext** will exist and be populated with an input Subject to operate on.
2. A called c14n flow will be present in the collection of available c14n flows (that is, if a flow is run, it can assume it was supposed to run).
3. A called c14n flow can assume that its ActivationCondition was successfully evaluated.
4. There may intermediate state present in the tree, but flows should generally assume that they are free to act as if no such state exists unless their logic is dependent on recognizing and acting on that state. In other words, if you don't care about intermediate results, you don't have to care about intermediate results.

The output contract is as follows:

1. If a c14n flow completes with a "proceed" event, then it MUST satisfy the following requirements:
    a. The **SubjectCanonicalizationContext**'s `principalName` property MUST be set.
2. If a c14n flow completes with any other event, it should assume that its outcome will be treated as unsuccessful. Flows may signal specific behavior back to the c14n master flow:
    a. ReselectFlow – tells the master c14n flow to choose another eligible flow to run (i.e., fall through)
    b. InvalidSubject – tells the master c14n flow that the input Subject was not appropriate for the flow to act on, and to choose another eligible flow to run (i.e., fall through)
    c. Anything Else – c14n will fail and the event will be reflected back as the result of the c14n master flow, to be interpreted by the calling parent flow (signaling a custom event requires that *conf/c14n/subject-c14n-events-flow.xml* be modified)

Programming Guide to Using Canonicalization

The below material is applicable to V3.3.0 and later of the IdP

If you're creating a custom profile/protocol feature that needs c14n, a web flow that wishes to invoke the c14n subsystem must do the following:

1. Create a **SubjectCanonicalizationContext** and populate (at least) the `subject` and `potentialFlows` properties, along with any other inputs available.
2. Attach the **SubjectCanonicalizationContext** as a child of the ProfileRequestContext.
3. Transfer control to the "c14n" subflow and transition on the possible results back to your own flow.

The following events worthy of special note may occur as a result of invoking the subsystem:

proceed	Successful c14n.
NoPotentialFlow	No c14n flow is configured for use or was able to operate on the input.
SubjectCanonicalizationError	The input was recognized but an error occurred trying to operate on it.

Various other events signifying more low-level error conditions may also occur.

In any case other than "proceed", the caller **MUST NOT** expect any results to be valid, and `canonicalizationError` **MAY** be populated.

In the case of "proceed", **SubjectCanonicalizationContext**'s `principalName` property **MUST** be set.

Note that the above outline holds for **any** potential use case for this feature. It is immaterial what the c14n process is actually operating on because that is abstracted into the Java Subject, and will be manifest in the specific c14n subflows that are populated as potential flows to use.