3.1 General Configuration

Before You Begin

Before you begin this section you will need to understand two things:

- the basics of Spring and its configuration files as described in the Spring Reference Documentation, sections 3.1 - 3.4
- Java class documentation in the form of javadocs

General Configuration Guide

Pipeline Stages

Currently, all configuration is performed through standard Spring `<beans>` files. Because pipelines are really just a collection of stages we'll start there and then move on to defining the actual pipeline.

Source Stages

The following stages read in information from various sources and add that information to the collection of items to be processed. You pipeline definition will usually start with one of these.

Stage	Description
net.shibboleth.metadata.pipeline.StaticItemSourceStage	This stage allows you to define a static set of items that are added to the processing collection.
net.shibboleth.metadata.dom.DOMFilesystemSourceStage	This stage reads one, or more, XML files from the filesystem and adds the document org.w3c.dom.Element, for each document read, to the item collection.
net.shibboleth.metadata.dom.DOMResourceSourceStage	This stage reads an XML file from a Spring `Resource` and adds the document org.w3c.dom.Element to the item collection.

DOM Processing Stages

The following stages operate on item collections containing org.w3c.dom.Element elements.

Stage	Description
net.shibboleth.metadata.dom.CRDetectionStage 0.9.1, 0.10.0	Examines all XML element text content and all XML attribute values in the item for the presence of a CR character. These can only appear in an XML document as the result of an explicit "&#13;" sequence, and can trigger an issue in the Shibboleth SP. If a CR character is detected, an net.shibboleth.metadata.ErrorStatus is added to the item, allowing subsequent stages to highlight the occurrence or to remove the item.
net.shibboleth.metadata.dom.ElementStrippingStage	This stage strips out all instances of a specified element.

net.shibboleth.metadata.dom.EmptyContainerStrippingStage	This stage strips out all instances of a specified element that do not contain other elements.
net.shibboleth.metadata.dom.MultiOutputXSLTransformationStage	This stage runs an XSL Transformation on each org.w3c.dom.Element in the item collection and replace the item with the resulting org.w3c.dom.Elements.
net.shibboleth.metadata.dom.NamespaceStrippingStage	This stage strips all elements and attributes in the specified XML namespace from each document in the item collection.
net.shibboleth.metadata.dom.XMLSchemaValidationStage	This stage schema validates each org.w3c.dom.Element in the item collection.
net.shibboleth.metadata.dom.XMLSignatureSigningStage	This stage signs each org.w3c.dom.Element in the item collection.
net.shibboleth.metadata.dom.XMLSignatureValidationStage	This stage validates the signature(s) present on each org.w3c.dom.Element, or its descendants, in the item collection.
net.shibboleth.metadata.dom.XPathFilteringStage	This stage evaluates each org.w3c.dom.Element in the item collection and removes those items that match a given XPath v1.0 expression.
net.shibboleth.metadata.dom.XSLTransformationStage	This stage runs an XSL Transformation on each org.w3c.dom.Element in the item collection and replaces the initial element with the result of the transform.
net.shibboleth.metadata.dom.XSLValidationStage	This stage "validates" each org.w3c.dom.Element in the item collection by running an XSL Transformation upon it.

SAML Processing Stages

The following stages operate on item collections that contain org.w3c.dom.Element that are SAML elements. Most stages operate on `EntitiesDescriptor` or `EntityDescriptor` elements.

Stage	Description
net.shibboleth.metadata.dom.saml.ContactPersonFilterStage	This stage filters the types of `ContactPerson` within each `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Element in the item collection.
net.shibboleth.metadata.dom.saml.EntitiesDescriptorAssemblerStage	This stage takes a collection of `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Elements and constructs a (potentially named) `EntitiesDescriptor` from them.
net.shibboleth.metadata.dom.saml.EntitiesDescriptorDisassemblerStage	This stage takes a collection of `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Elements and replaces each `EntitiesDescriptor` with all of its descendant `EntityDescriptor` org.w3c.dom.Elements.
net.shibboleth.metadata.dom.saml.EntityDescriptorItemIdPopulationStage	This stage adds an net.shibboleth.metadata.ItemId, containing the entity ID of an entity descriptor, to an item's metadata.
net.shibboleth.metadata.dom.saml.EntityFilterStage	This stage filters `EntityDescriptor` org.w3c.dom.Elements from a collection of `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Elements based on a white/blacklist.
net.shibboleth.metadata.dom.saml.EntityRoleFilterStage	This stage filters entity roles from a collection of `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Elements based on a white/blacklist.

net.shibboleth.metadata.dom.saml.GenerateIdStage	This stage generates an ID for `EntitiesDescriptor` or `EntityDescriptor` elements within the item collection.
net.shibboleth.metadata.dom.saml.PullUpCacheDurationStage	This stage processes each `EntitiesDescriptor` org.w3c.dom.Elements in the item collection (ignoring `EntityDescriptor` elements) by choosing the shortest `cacheDuration` from all its descendants, placing that duration on the root `EntitiesDescriptor`, and removing the `cacheDuration` from all its descendants.
net.shibboleth.metadata.dom.saml.PullUpValidUntilStage	This stage processes each `EntitiesDescriptor` org.w3c.dom.Element in the item collection (ignoring `EntityDescriptor` elements) by choosing the nearest `validUntil` from all its descendants, placing that on the root `EntitiesDescriptor`, and removing the `validUntil` from all its descendants.
net.shibboleth.metadata.dom.saml.RemoveOrganizationStage	This stage removes the `Organization` information from each `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Element in the item collection.
net.shibboleth.metadata.dom.saml.SetCacheDurationStage	This stage sets a `cacheDuration` for each top level `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Element in the item collection.
net.shibboleth.metadata.dom.saml.SetValidUntilStage	This stage sets a `validUntil` for each top level `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Element in the item collection.
net.shibboleth.metadata.dom.saml.ValidateValidUntilStage	This stage validates that each the `validUntil` attribute for each EntitiesDescriptor and EntityDescriptor has not yet passed. It may also require that such information be present and may enforce a maximum validity period.

MDRPI Processing Stages

The following stages operate on item collections that contain org.w3c.dom.Element that are SAML elements; they implement functionality related to the SAML V2.0 Metadata Extensions for Registration and Publication Information Version 1.0 specification. Most stages operate on `EntitiesDescriptor` or `EntityDescriptor` elements.

Stage	Description
net.shibboleth.metadata.dom.saml.mdrpi.EntityRegistrationAuthorityFilterStage	This stage filters `EntityDescriptor` org.w3c.dom.Elements from a collection of `EntitiesDescriptor` or `EntityDescriptor` org.w3c.dom.Elements based on the entity's registration authority.
net.shibboleth.metadata.dom.saml.mdrpi.RegistrationAuthorityPopulationStage	This stage extracts each entity's registration authority from a collection of `EntityDescriptor` org.w3c.dom.Elements and uses it to add a net.shibboleth.metadata.dom.saml.mdrpi.RegistrationAuthority to the item metadata for the item for access by later stages. Note that `EntityRegistrationAuthorityFilterStage` does not make use of this metadata, but extracts the same information directly.

Item Metadata Stages

The following stages operate upon net.shibboleth.metadata.ItemMetadata associated with a given Item.

Stage	Description
net.shibboleth.metadata.pipeline.ItemMetadataFilterStage	This stage filters out Items that are "tagged" with a particular ItemMetadata
net.shibboleth.metadata.pipeline.ItemMetadataTerminationStage	This stage terminates processing if an Item is "tagged" with a particular ItemMetadata
net.shibboleth.metadata.pipeline.StatusMetadataLoggingStage	This stage logs info, warn, or error messages, for an Item, if tagged with net.shibboleth.metadata.InfoStatus, net.shibboleth.metadata.WarningStatus, or net.shibboleth.metadata.ErrorStatus respectively.

Pipeline Structure Stages

The following stages are used for creating advanced pipeline structures (e.g., splitting one pipeline into multiple pipelines).

Stage	Description
net.shibboleth.metadata.pipeline.CompositeStage	This stage compose multiple stages in to a single named and reusable unit.

net.shibboleth.metadata.pipeline.PipelineDemultiplexerStage	This stage takes an item collection, and generates a number of new collections which are then passed on to associated pipelines for further processing.
net.shibboleth.metadata.pipeline.PipelineMergeStage	This stage combines the results of multiple pipelines.
net.shibboleth.metadata.pipeline.SplitMergeStage	This stage splits a given collection in to two collection, runs each through a pipeline, and then merges the two collections back together afterwards.

Other Stages

The following stages are miscellaneous stages that just don't fit in to a category at the moment.

Stage	Description
net.shibboleth.metadata.pipeline.ScriptletStage	This stage executes a JSR-223 compliant script against each item in the item collection.
net.shibboleth.metadata.pipeline.SerializationStage	This stage serializes the content of an item collection out to file be means of an net.shibboleth.metadata.ItemSerializer.

Pipeline Configuration

Pipelines are defined as anything that implements the net.shibboleth.metadata.pipeline.Pipeline interface but, for most cases, the only implementation you'll need is the net.shibboleth.metadata.pipeline.SimplePipeline.

To configure this pipeline you must set the following properties:

- `id`
- `stages`

Error Handling

Non-fatal Errors

In general, it's often not advantageous to halt the processing when on part of a given input is invalid. Usually you just want to mark it as invalid, proceed on, and remove the invalid content when you're done. Stages that do this will mark items with instances of net.shibboleth.metadata.StatusMetadata. You can then use the item metadata stages listed above to filter out, log, and/or terminate processing based on the status of the item.

Fatal Errors

Some errors, such as when attempting to read in XML data that is invalid, are considered fatal errors and will halt the processing of the pipeline.

Spring Specific Information

Object Initialization

All stages and pipelines and some other objects (e.g., parser pools) must be initialized before use and Spring provides a couple options for doing this.

The easiest option is to add the attribute `default-init-method="initialize"` to the root `<beans>` element in your configuration. This will cause Spring to call the `initialize()` method on any bean that has one.

Alternatively, if you need very fine-grained control for some reason, you can add the attribute `init-method="initialize"` to every bean that requires initialization.

Utility Namespace

Spring provides extensions to the standard `<beans>` notation through custom namespaces. In particular, the util extension can be very helpful when you need to construct collections of things. Just be sure to add the appropriate namespace declaration on the root `<beans>` element if you use this extension.

Expression Language

In most cases, anywhere you can put a direct value (e.g., a int, a string) you can also use a Spring expression. This can be helpful if you want to compute or derive a value.

Of particular note, you can use the type operator, `T(...)` to get access to a `java.lang.Class` instance for a given type. For example:

```
<property name="foo" value="#{T(net.shibboleth.metadata.InfoStatus)}"/>
```

Helper Classes

When working with Spring it is sometimes difficult to convert from some string designation of an object into the object itself (e.g., changing the file path for a private key into a `PrivateKey` object). The following factory beans are available to help with this:

Factory Class	Description
net.shibboleth.ext.spring.factory.DOMDocumentFactoryBean	Bean that produces a org.w3c.dom.Document given a file path and a parser.
net.shibboleth.ext.spring.factory.PrivateKeyFactoryBean	Bean that produces a java.security.PrivateKey given a file path.
net.shibboleth.ext.spring.factory.PublicKeyFactoryBean	Bean that produces a java.security.PublicKey given a file path.
net.shibboleth.ext.spring.factory.X509CertificateChainFactoryBean	Bean that produces a java.security.cert.X509Certificate chain given a file path.
net.shibboleth.ext.spring.factory.X509CertificateFactoryBean	Bean that produces a java.security.cert.X509Certificate given a file path.