UIMA Version 3 User's Guide

Written and maintained by the Apache UIMA™ Development Community

Version 3.0.0-SNAPSHOT

Publication date October, 2016

Table of Contents

Chapter 1. UIMA Resources

1.1. What is a UIMA Resource?

UIMA uses the term `Resource` to describe all UIMA components that can be acquired by an application or by other resources. These are typically written by users, and are not part of the UIMA framework itself.

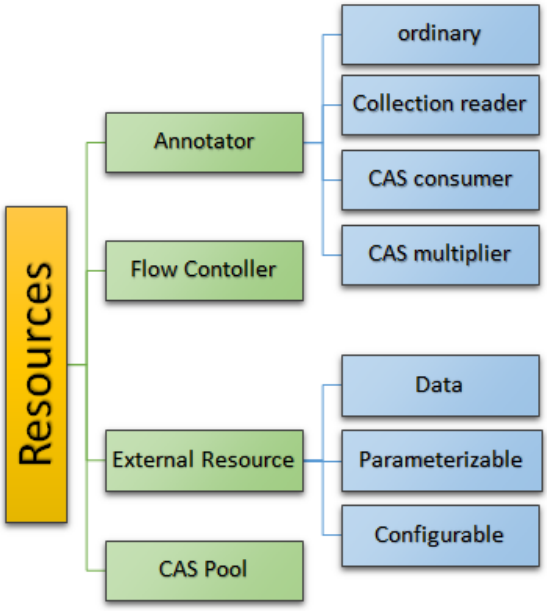

Figure 1.1. Resource Kinds

Resource kinds include:

Annotator
a user written component, receives a CAS, does some processing, and returns the possibly updated CAS. Variants include CollectionReaders, CAS Consumers, CAS Multipliers.

Flow Controller
a user written component controlling the flow of CASes within an aggregate.

External Resource
a user written component. Variants include:

- Data - includes special lifecycle call to load data
- Parameterized - allows multiple instantiations with simple string parameter variants; example: a dictionary, that has variants in content for different languages
- Configurable - supports configuration from the XML specifier

CAS Pools
This is a UIMA framework resource, providing a pooling mechanism to efficiently share CAS instances with identicial type systems with multiple UIMA pipelines.

1.1.1. Resources-framework versus Resources

It is easy to confuse UIMA framework classes and methods designed to support resources, with the resources themselves. There are many different specializations of resources, and corresponding parts of the UIMA framework that support these.

Here's a small part of the UIMA Frame support classes, in green, showing some of the User resources that have been (in this example) instantiated, in yellow. The yellow components have their own superclass hierarchy, indicated by the upwards pointing arrow, independent from the resource framework implementation. Each Resource has its own set of framework class instances, starting with a `Resource_Impl` instance, and also its own User code instance. For clarity, the multiple instances are omitted in the middle of the diagram.

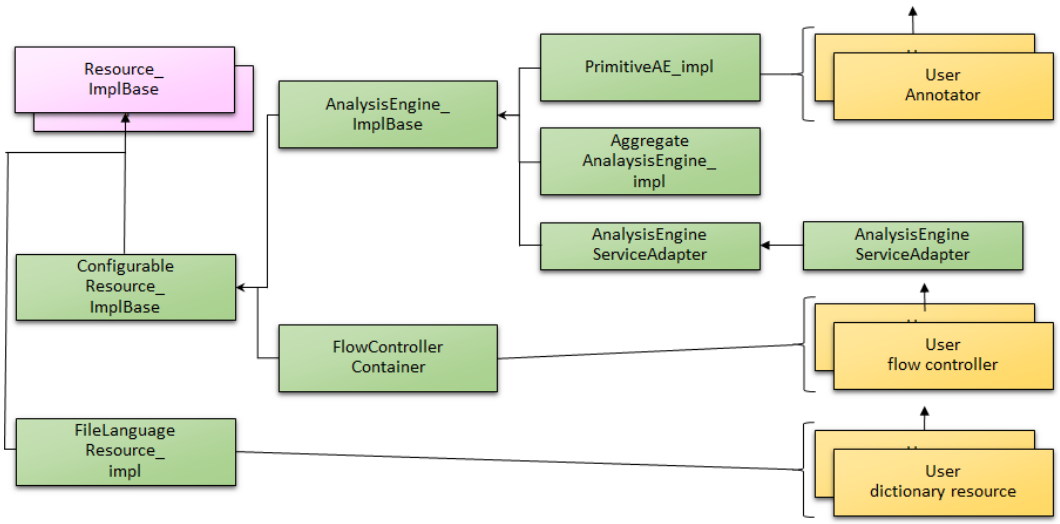

Figure 1.2. Resource Kinds

1.2. Resource Specifiers

Resources are instantiated from specifications contained in XML Resource Specifiers. These are described in the reference chapter on component descriptors. A common `initialize` method that is part of UIMA framework classes that supports Resources, takes the `ResourceSpecifier` (the internal Java form of the XML resource specifier), plus a key-value map of arbitrary additional parameters, and is responsible for configuring the instance of the UIMA framework support class so that it can respond to subsequent method calls to get an instance of the resource.

Note that this `initialize` method is different from the `initialze(uimaContext)` method that is part of the API for AnalysisEngines.

1.3. Sharing Resources, even across pipelines

UIMA applications run one or more UIMA Pipelines. Each pipeline has a top-level Analysis Engine, which may be an aggregation of many other Analysis Engine components. The UIMA framework instantiates Annotator resources as specified to configure the pipelines.

Sometimes, many identical pipelines are created (for example, in order to exploit multi-core hardware by processing multiple CASes in parallel). In this case, the framework would produce

multiple instances of those Annotation resources; these are implemented as multiple instances of the same Java class.

Multiple resources in addition to the Annotators are set up and kept in a single instance of the ResourceManager; this instance serves to allow sharing of 3 types of things across one or more pipelines. These are:

- The UIMA Extension ClassLoader (if specified) - used to find the resources

  The `External Resources`

  The CAS Pool

In typical use, no existing ResourceManager used when creating a pipeline; this results in a new ResourceManager being created and used for that pipeline. However, in many cases, it may be advantageous to share the same Resources across multiple pipelines; this is easily doable by passing a common instance of the ResourceManager to the pipeline creation methods (using the additional parameters).

For PEAR wrapper usage, a special extra version of an existing ResourceManager is created, called the `ResourceManagerPearWrapper`, which keeps all the same resources, except that it has a separate value for the UIMA Extension Classloader. This is used to support the classpath isolation feature of PEARs.

1.4. Resource lifecycles

The lifecycle for resources includes several events.

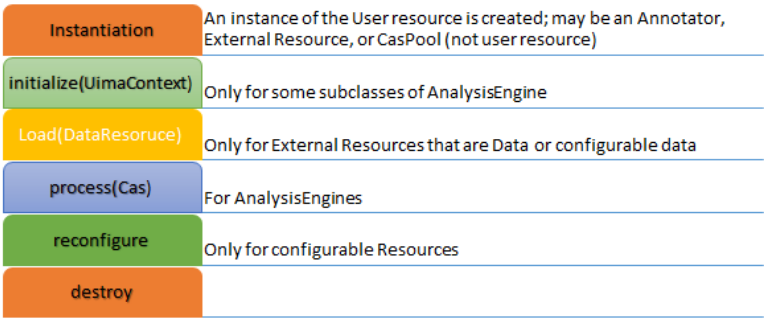

Figure 1.3. Resource Lifecycles

Annotators are `instantiated` by the various flavors of `produceResource`, and their `initialize(UimaContext)` method is called. While the pipe line is running, each new CAS that arrives is passed in via the `process` method. The UIMA Framework doesn't generate a `destroy` call on its own, because only the application code making use of the UIMA Framework knows when a pipeline is finished and can be destroyed.

External Resources are instantiated as a side effect of running `produceResource` to produce a pipeline. Instances of DataResource have their `load` method called during this time. Instances of ConfigurableDataResource defer their call to `load` until a call is made to get the resource - this call supplies a parameter, such as a language code, used to pick one of several inputs to load.

CasPools are instantiated lazily, when a Cas is requested from the pipeline, or when a pool of a given size needs to be set up for some pipeline configurations. For instance, the MultiprocessingAnalysisEngine configuration sets up a pool with the size equal to the number of parallel pipelines being configured.

(New as of UIMA 2.10.0) For both External Resources and the Cas Pool, `destroy` is not generated internally by the UIMA framework, because it doesn't know when the application (which might be sharing the ResourceManager's resources among multiple pipelines) is finished. The application may call `destroy` on the ResourceManager instance, which will then forward this to the External Resources, and the CAS Pool.

1.5. ResourceManager and PEARs

1.5.1. Lifecycle for PEARs

1.6.

1.6.1. Multiple Parameterized Instances of a particular resource

1.7. Resource Configuration

1.7.1. Configuration of External Resources

1.8. CAS Pools