Application Configuration

The Aspire Application XML file (application.xml) specifies the components (and their configurations) that together implement your application.

The system configuration file is also known as the "Component Manager" configuration file, since it is, technically, the component manager that reads the file and initializes the components.

On this page:

Load the Configuration File

The configuration file can be loaded in two ways:

- Through the System Administration user interface.
  - Using a configuration file stored on disk
  - By downloading and installing an App Bundle from Maven
- Automatic launching on startup via the Settings file. See General Settings for more details.

To load a system configuration file through the user interface, do the following:

1. Go to the Aspire home page for your server (typically http://localhost:50505/aspire).
2. Locate the option to "Load a new Application."
3. Specify the configuration file to be loaded. This should be relative to the Aspire Home location, for example "config/application.xml."
4. Click "start."

Note: Loading a configuration file via the System Administration user interface utilizes an application servlet command. See Advanced Configuration Loading below.

Loading multiple configuration files

You can load multiple configuration files - as many as you'd like. This is a very useful technique for organizing your application into small, easily manageable and configurable sections - one per system configuration file.

Configuration File Basics

The template for the standard configuration file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<application name="YOUR-APP-NAME" typeFlags="feeder,sub-jobs">
  <components>
    ... components to create/initialize go here ...
  </components>
</application>
```

Notes:

- The application name (<config name="YOUR-CONFIG-NAME">) will be used to name the application that holds all of the components that it creates.
  - This name will be used by the System Administration user interface to reference the application.
  - Applications with the same name (on different servers in a cluster) will be assumed to perform the same function.

    - This name will also be prefixed to all of the names of the components created by the configuration manager.
    - See below for more details.
- The attribute @typeFlags is optional. It is used to identify this configuration to the administration user interfaces. See below for more details.

Components

Every application contains a simple list of components. Each component can be configured with configuration XML.

Note that some components are component managers, such as the pipeline manager. These managers can have sub-components nested within them. Nested components are just like any other component and can be created and configured with XML.

The basic structure for configuring components is shown below:

```
<components>
  <component name="NAME1" subType="SUBTYPE" factoryName="FACTORY-NAME"> ... </component>
  <component name="NAME2" subType="SUBTYPE" factoryName="FACTORY-NAME"> ... </component>
  <component name="NAME3" subType="SUBTYPE" factoryName="FACTORY-NAME"> ... </component>
  <component name="NAME4" subType="SUBTYPE" factoryName="FACTORY-NAME"> ... </component>
</components>
```

Notes:

- All components in the list of components will be created and initialized.
- Multiple copies of the any type of component can be created.
- All components are given a name. The names must be unique (within the component manager in which they are created - like files within a directory must be unique). The name is used to reference the component within the component manager.
- See the wiki documentation for each individual component for details on how individual components are configured.

Enable and disable components

Components can be disabled using @enable or @disable attributes. If both @enabled and @disabled flags are specified, the value of @enable takes precedence. Disabled components are completely removed from the system, as if they had never been written into the XML file at all.

These flags are useful for turning on or off components in response to property settings (either as an App Bundle or via property settings specified in the settings.xml file).

Example:

```
 <!-- The next two components are declared, but never initialized. -->
<component enable="false" .../>
<component disable="true" .../>
```

If neither @enable or @disable are present, then it is assumed that the component is enabled.

If a component manager (such as a pipeline manager) is disabled, then it will not load itself, nor will it load or initialize any of its sub-components.

Component names

The following example shows a system configuration file that contains two components:

```
 <?xml version="1.0" encoding="UTF-8"?>
<application name="Example">
  .
  .
  .
  <components>
    <component name="FeedOne" subType="feedOne" factoryName="aspire-tools">
        .
        .
        .
    </component>


    <component name="MyProcessor" subType="default" factoryName="aspire-groovy">
        .
        .
        .
    </component>
  </components>
</config>
```

Note: *The new standard is for all component names to start with an Uppercase Letter.*

Technically, the above configuration will create three components:

/Example	This is the name of a component manager itself (all Aspire Applications are, in fact, component managers which contain nested components).
/Example/FeedOne	The name of the feed one component.
/Example /MyProcessor	The name of the job processor written in Groovy.

Notice how the subcomponents all have the name of the component manager prefixed to them. In this way, all component names are hierarchical - like directory and file names. See Naming Components for more information.

Factory names

Factory names in Aspire specify the Maven Artifact which is the JAR file that implements the required Aspire component factory. This can be specified in the Aspire System Configuration file in three different ways:

- *artifactId* - (for example: "aspire-rdb") This specifies the Maven artifact ID of the component to be loaded into Aspire.
    - "com.searchtechnologies.aspire" be used for the group ID.
    - The version number will be the default as specified in the settings.xml file.
- *groupId:artifactId* - (for example: "com.customer:aspire-xyx") This specifies the maven group ID and artifact ID of the component bundle to load into into Aspire.
    - The version number will be the default as specified in the settings.xml file.
- *groupId:artifactId:version* - (for exmaple: "com.searchtechnologies.aspire:aspire-rdb:2.0") This specifies the full Maven coordinates of the component JAR file to load into aspire.

Note that the Maven packaging (i.e. "jar", "bundle" etc.) is not needed and can not be specified.

Example:

```
    <component name="FeedOne" subType="feedOne"
                factoryName="com.searchtechnologies.aspire:aspire-tools:2.0">
       .
       .
   </component>
```

Sub-types

Component factories correspond to Bundles, i.e., actual physical JAR files that are dynamically loaded into Aspire.

Each JAR file can actually produce multiple different types of components, which correspond to different Java Classes that implement the Aspire Component interface. For example, the "aspire-rdbfeeder" bundle can create different components for pulling records from relational databases (RDB Feeder), creating sub-jobs based on an RDB query (RDB Sub Job Feeder) and extracting additional metadata from an RDB to be added to an existing job (RDB Row Extractor).

Sub Types are correlated to implementation classes within the component via the component's ComponentFactory.xml file.

Sub Types are desirable for many reasons. They reduce the number of JAR files required, they leverage third-party Jars incorporated into Bundle Jars for multiple component types, and they help make configuration (and coding) simpler.

The subType value required for each component can be found on the wiki page for that component.

Property Substitution

Properties can be specified with ${propertyName} inside the system.xml file. These properties will be substituted when the configuration is loaded.

See Properties for more details on substituting properties.

settings.xml properties

Properties can be specified inside the <properties> tag inside the settings.xml file. All of these properties can be used for property substitution inside the settings.xml configuration file. See system.xml properties for more information.

Standard Aspire property names

Several standard Aspire property names are available to all configurations. These include:

- **${aspire.home}** - The home directory of the Aspire distribution for the Aspire node that is running the application.
- **${appbundle.home}** - The directory where the Application Bundle was unpacked. Currently this is ${aspire.home}/cache/appbundles/<group-id>/<artifact-id>/<version> based on the Maven coordinates of the App Bundle.
    - If ${appbundle.home} is used in a standard configuration file (in other words, it's not deployed and downloaded from Maven), then ${appbundle.home} == ${aspire.home}. This will allow you to test App Bundles inside of standard distributions before they are deployed to Maven.
- **${app.name}** - This is the name of the application, which will be the same as the top-level component name.
    - For example, if you install an application.xml and give it the top-level name of "/CIFSConnector" (for example), then ${app.name} will be "CIFSConnector".
- **${app.data.dir}** - The standard location where the application should store it's data. In order that applications do not step on each other's data, every application should store its data into ${app.data.dir}. Currently, this will be the same as ${aspire.home}/data/${app.name} .
- **${shared.data.dir}** - The standard location for data which is shared across applications. Currently the same as ${aspire.home}/data/shared .

Environment variables

Environment variables can used for property names. For example, ${JAVA_HOME}.

Java system properties

All Java system properties are also available. For example, ${user.home} will be the home directory for the user who started up Aspire.

Loading values from Java properties files

You can also load properties in to Aspire from a Java properties file. The properties file would be of the form:

```
 rdbDbName=research
rdbUser=SYSTEM
rdbPassword=hello
my.rdb.password=helloAgain
```

Rather than directly loading the entire properties file (as these are quite often large and contain a large number of properties that that are irrelevant to Aspire), you configure Aspire to read just the properties you are interested in. These *Java* properties are then loaded to *Aspire* properties and the subsitution is performed as described above.

To load the desired properties, include an *external* tag in the *properties* section and then specified the *filename* in which the property is found.

For example, to load the *my.rdb.password* property from the file above, use:

```
  <properties>
   <property name="crawlDataBase">data/crawler</property>
   <external name="my.rdb.password" filename="testdata/com.searchtechnologies.aspire.framework/Properties
/test.properties"/>
   <external name="badReference" filename="testdata/com.searchtechnologies.aspire.framework/Properties
/doesnot.exist"/>
 </properties>
```

NOTE: if the filename you give is not found, the Aspire property will be set to indicate this:

(unable to open external properties file testdata/com.searchtechnologies.aspire.framework/Properties/doesnot.exist)

Advanced Configuration Loading

Using the UI to load a configution file invokes an underlying Application servlet command *start*. If the user chooses to load a file named config/system.xml (by typing config/system.xml in the input box), the URL sent to the application is:

```
http://localhost:50505/aspire?cmd=start&config=config/system.xml
```

The *start* command

Other options that are not available via the UI are available using the command *start*. Using *xml* instead of *config* allows the user to specify an **application** xml fragment and load this configuration:

```
   <application name="CSManager" config="com.searchtechnologies.appbundles:cs-manager:1.0-SNAPSHOT">
    <properties>
      <property name="debug">true</property>
      <property name="managerExternalRDB">false</property>
      <property name="managerRDB">CSRDB</property>
      <property name="managerExternalJDBCUrl"></property>
      <property name="managerExternalJDBCDriverJar"></property>
      <property name="managerExternalJDBCUser"></property>
      <property name="managerExternalJDBCPassword"></property>
    </properties>
  </application>
```

The following URL loads the above configuration:

```
 http://localhost:50505/aspire?cmd=start&xml=%3Capplication%20name=%22CSManager%22%20config=%22com.
searchtechnologies.appbundles:cs-manager:1.0-SNAPSHOT%22%3E%3Cproperties%3E%3Cproperty%20name=%22debug%22%
3Etrue%3C/property%3E%3Cproperty%20name=%22managerExternalRDB%22%3Efalse%3C/property%3E%3Cproperty%20name=%
22managerRDB%22%3ECSRDB%3C/property%3E%3Cproperty%20name=%22managerExternalJDBCUrl%22%3E%3C/property%3E%
3Cproperty%20name=%22managerExternalJDBCDriverJar%22%3E%3C/property%3E%3Cproperty%20name=%
22managerExternalJDBCUser%22%3E%3C/property%3E%3Cproperty%20name=%22managerExternalJDBCPassword%22%3E%3C
/property%3E%3C/properties%3E%3C/application%3E
```

Note that this is simply the URL encoded version of the xml:

```
  <       -> %3C
 "       -> %22
 >       -> %3E
 <space> -> %20
```

You may also indicate that the new configuration should be written to the settings.xml file in the <autoStart> section. Use the *autoStart* flag for this:

```
 http://localhost:50505/aspire?cmd=start&autoStart=true&xml=%3Capplication%20config=%22com.
searchtechnologies.appbundles:cs-manager:1.0-SNAPSHOT%22%3E%3Cproperties%3E%3Cproperty%20name=%22debug%22%
3Etrue%3C/property%3E%3Cproperty%20name=%22managerExternalRDB%22%3Efalse%3C/property%3E%3Cproperty%20name=%
22managerRDB%22%3ECSRDB%3C/property%3E%3Cproperty%20name=%22managerExternalJDBCUrl%22%3E%3C/property%3E%
3Cproperty%20name=%22managerExternalJDBCDriverJar%22%3E%3C/property%3E%3Cproperty%20name=%
22managerExternalJDBCUser%22%3E%3C/property%3E%3Cproperty%20name=%22managerExternalJDBCPassword%22%3E%3C
/property%3E%3C/properties%3E%3C/application%3E
```

The *remove* command

Use the *remove* command to remove a component manager from the application's configuration:

```
http://localhost:50505/aspire?cmd=remove&name=/CWSManager
```

The *name* passed will be the name attribute from the file or app bundle loaded (or name from the component manager xml fragment if specified). When using *remove* you may also pass *autoStart=false* to remove the configuration from the settings file

Application Names and Type Flags

At the top of the application XML file you can specify a name for the entire application, and some type flags, which identify the purpose of the configuration for the administration user interface:

```
 <?xml version="1.0" encoding="UTF-8"?>
<application name="MyName" typeFlags="feeder,sub-jobs">
  .
  .
  .
</application>
```

System configuration name

The name specified at the top of the application XML file is used for the following purposes:

- As the top level name for all components within the configuration
  - See Component_Names above for more details.
- To identify this configuration within an aspire cluster
  - It is assumed that the same application loaded onto two different nodes within an Aspire cluster will represent the same functionality. For example, a pipeline module can branch a job to "/MyName/MyPipeline" and this job could be sent to any machine within the Aspire Cluster which has the same name. Therefore, names are critically important to identify equal functionality within a distributed Aspire system.
- In the admin interface, to specify job routing tables
  - The administration interface in Aspire 0.5 allows jobs to be routed using a routing table. This table is usually a simple list of top-level application names (although a nested component can be specified). Therefore, the application name is important because it will identify how the application appears in the administration user interface so that jobs can be routed to it.

Note that the name specified in the application.xml file is the **default** name. A new name can be specified in the System Administration user interface when the configuration file or app bundle is loaded.

The typeFlags

The type flags in the system configuration file is specified as follows:

```
 <application name="MyName" typeFlags="feeder,subjobs">
  .
  .
  .
</application>
```

The flags are optional and are used to specify how a configuration can be used to process jobs for the System Administration user interfaces.

Application type flags currently defined

blocked URL

The following flags are currently defined:

- **scheduled** - Specifies that the application or App Bundle can receive jobs from the Aspire Scheduler.
  - Not only does this include scheduled jobs, but usually it also indicates that the configuration can accept "start", "stop", "pause", "resume", and "restart" commands to provide dynamic control of long-running jobs.
- **feeder** - Indicates that the configuration is waiting for jobs to come from the outside.
  - For example, a feeder may expect jobs from a JMS Queue, from an HTTP Feeder, or may be automatically polling a third-party system looking for new jobs to process.
- **job-input** - Is used for configurations that expect a job as input, then process the job.
  - Typically used for configurations that process documents for extracting or normalizing metadata.
- **subjobs** - Used for configurations that produce sub-jobs.
  - Jobs with routing tables processed by these types of configurations will need nested routing tables to specify how the sub-jobs are to be routed.
- **group-expansion** - Used for configurations that can process group expansion requests
  - These applications are expected to have a *GroupExpansionPipelineManager* pipeline manager to handle group expansion logic
- **query-federation** - Is used for configurations that expect a job as input, then perform a search engine query, placing the results in the job.
  - Typically used for configurations that will be used by the Federation Dispatcher.
- **publisher** - Specifies that the application or App Bundle is a publisher that can generate an index dump.
  - Used for filtering among the rest of installed application in the auditing tool.

Although @typeFlags can be used as part of as part of application XML files, they are really designed to be used with Application Bundles (or App Bundles for short). App Bundles are units of functionality which can be made up of multiple pipeline managers and many components. App Bundles are specified with a system configuration file (as described on this wiki page) and can be bundled with other files into JAR files and deployed to Maven repositories.

How the UI uses type flags

The System Administration user interface can create routing tables, which are attached to jobs, and specify how they are routed from configuration to configuration (or, more typically, from App Bundle to App Bundle). The @typeFlags attribute is intended to provide enough information to the user interface so that these routing tables can be created.

Specifically:

- **scheduled** - The job is stored in the scheduler database. The administrator can specify a job submission schedule (e.g., daily at 2am, weekly on Saturday at 3am, every hour, etc.) for the job.
  - In this case, the Aspire Scheduler is automatically configured, and the configuration with the "scheduled" flag will occur first in the routing table.
- **feeder** - Indicates that the application or App Bundle can produce jobs based on outside events.
  - When feeders create jobs, they will automatically attach the routing table to them. (*feature not currently being used*)
- **job-input** - Allows for the configuration to be placed in the middle of a routing table, to receive jobs from up-stream configurations or app bundles.
- **subjobs** - Identifies that the configuration or App Bundle produces sub-jobs. This means the user interface will need to allow the administrator to specify a nested pipeline that will be used to route the sub-jobs produced by this configuration or App Bundle.
- **publisher** - Identifies the applications or App Bundles that are publishers and can perform index dumps. This is used for the Auditing section of the Admin Interface