Standalone Mode

The Aspire 3.x Connectors implement an interface called RepositoryAccessProvider that specifies the minimum required methods to access, fetch, and scan a given Repository.

The Aspire 3.x Connector Framework layer provides common control code for

- Full / incremental crawling
- Distributed processing
- Group expansion
- Schedules
- Link between the Aspire Admin User Interface and the crawls

by calling the RepositoryAccessProvider method when it needs to access the Repository.

On this page

The separation of the Connector Framework and the Connector Implementation allows a very natural usage of the Connector Implementation outside of the Connector Framework (and outside of Aspire at all).

There are five main tasks that the RepositoryAccessProvider is responsible for:

1. **Initializing the crawl configuration or SourceInfo, from the user configuration properties**
   - Initial URL, username, passwords, etc.
   - Method: newSourceInfo(AspireObject properties)
2. **Extract the initial or root crawl items**
   - The discovered items are sent into a ScanListener
   - Method: processCrawlRoot(SourceItem root, SourceInfo info, ScanListener listener)
3. **Populate the extracted items metadata**
   - Method: populate(SourceItem item, SourceInfo info, RepositoryConnection conn)
4. **Scan a container item**
   - The discovered sub-items are sent into a ScanListener
   - Method: scan(SourceItem item, SourceInfo info, RepositoryConnection conn, ScanListener listener)
5. **Fetch the content stream for an item**
   - Open an input stream of the content of each of the items if available
   - Method: getFetcher()

Putting the Steps Together

Putting it all together is the responsibility of either the Connector Framework or you as a stand-alone developer of a given connector implementation.

The following procedure is based on the aspire-connector-wrapper-master.zip project:

Step 1 - Create a Java Maven project

Import the following dependencies into its pom.xml file:

Pom dependencies

```
<dependency>
   <groupId>com.searchtechnologies.aspire</groupId>
   <artifactId>aspire-connector-services</artifactId>
   <version>3.0</version>
</dependency>
<dependency>
   <groupId>com.searchtechnologies.aspire</groupId>
   <artifactId>aspire-connector-framework</artifactId>
   <version>3.0</version>
</dependency>
<dependency>
   <groupId>com.searchtechnologies.aspire</groupId>
   <artifactId>aspire-services</artifactId>
   <version>3.0</version>
</dependency>
<dependency>
   <groupId>org.osgi</groupId>
   <artifactId>org.osgi.core</artifactId>
   <version>4.2.0</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>org.osgi</groupId>
   <artifactId>org.osgi.compendium</artifactId>
   <version>4.2.0</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>com.searchtechnologies.aspire</groupId>
   <artifactId>THE-CONNECTOR-TO-USE</artifactId>
   <version>3.0</version>
</dependency>
<dependency>
   <groupId>com.searchtechnologies.aspire</groupId>
   <artifactId>aspire-core</artifactId>
   <version>3.0</version>
</dependency>
```

Step 2 - Initialize the crawl and configuration objects

Crawl Initialization

```
public static void main(String[] args) {

    //Get the scan properties from the arguments
    String scanPropertiesFile = args[0];

    //Instantiate a new RepositoryAccessProvider from the connector implementation
    JobFactory.initializeForStandAloneUsage();

    ComponentImpl component = new SP2013RAP();
    component.initialize(emptyDom);
    RepositoryAccessProvider rap = (RepositoryAccessProvider) component;

    //Create a CrawlControllerImpl, as some connectors depends on it
    StandaloneCrawlController crawlCtrlImpl = new StandaloneCrawlController(rap);

    //Load the content-source.xml configuration into an AspireObject
    AspireObject scanProps = new AspireObject("doc");
    AspireObject scanPropsFile = AspireObject.createFromXML(new File(scanPropertiesFile));
    scanProps.add(scanPropsFile);

    //Create and initialize a new SourceInfo from the RAP
    SourceInfo info = rap.newSourceInfo(scanPropsFile);
    info.initialize(scanProps);
    info.setCrawlController(crawlCtrlImpl);
    crawlCtrlImpl.setSourceInfo(info);
```

Step 3 - Start the crawl

Start crawl call

```
    //execute the crawl
    crawl(rap, info, Main::downloadStream);
```

crawl() method

crawl method

```
  private static void crawl(RepositoryAccessProvider rap, SourceInfo info, Consumer<SourceItem> processor)
throws AspireException {
    //The ScanListener which maintain the local processQueue and listen for new items to crawl
    Scanner scanner = new Scanner(rap, info);

    //Create the crawlRoot item to initialize the crawl from the RAP
    SourceItem crawlRoot = new SourceItem("crawlRoot");
    rap.processCrawlRoot(crawlRoot, info, scanner);

    //Crawls the local processQueue while it is not empty
    // when empty, it means the crawl finished
    do {
      scanner.processQueue(processor);
    } while (!scanner.isQueueEmpty());
  }
```

downloadStream() method

downloadStream method

```
private static void downloadStream(SourceItem item) {
  System.out.println("Item: "+item.getName());
  try {
    InputStream is = item.getContentStream();
    if (is != null) {
      FileOutputStream fos = new FileOutputStream(new File("output/"+item.getName()));
      copyStream(is, fos);
      fos.close();
      is.close();
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Step 4 - Call the Scanner class (where the crawl occurs)

When calling the processQueue() method, the following happens:

1. The current items are moved to another queue to be iterated.
2. For each item in the queue:
    a. Checks if the item is a container
    b. Calls the populate method on the items that need to be processed
    c. Calls the fetcher to process the item
    d. If the item is a container, calls the scan() method with it (This adds more items into the queue.)

Scanner class

```
static class Scanner implements ScanListener {

  /**
   * The queue that receives all the new items discovered
   */
  ArrayList<SourceItem> queue;

  /**
   * Temporary queue used for iterate the original queue
   */
  ArrayList<SourceItem> safeQueue;
  private RepositoryAccessProvider rap;
  private SourceInfo info;
  private FetchURL fetcher;

  public Scanner(RepositoryAccessProvider rap, SourceInfo info) throws AspireException {
    this.rap = rap;
    this.info = info;
    queue = new ArrayList<SourceItem>();
    safeQueue = new ArrayList<SourceItem>();
    fetcher = new StandaloneFetchURL(rap);
    fetcher.initialize(emptyDom);
  }

  public void close() {
    fetcher.close();
  }

  @Override
  public void addItem(SourceItem item) {
    //This gets called when the RAP scanner adds an item to crawl
    queue.add(item);
  }

  @Override
  public void addItems(List<SourceItem> items) {
    //This gets called when the RAP scanner adds items to crawl
```

```
    queue.addAll(items);
  }

  public void processQueue(Consumer<SourceItem> processor) throws AspireException {

    RepositoryConnection conn = rap.newConnection(info);

    //Move the items from the original queue into the safeQueue
    //And clear the orignal
    safeQueue.clear();
    safeQueue.addAll(queue);
    queue.clear();

    for (SourceItem item : safeQueue) {
      boolean container = false;

      if (rap.isContainer(item, conn)) {
        container = true;
      }

      if (info.indexContainers() || !container) {
        rap.populate(item, info, conn);
        //Call the fetcher
        Job job = JobFactory.newInstance(item.generateJobDocument());
        job.put("sourceInfo", info);
        job.put("crawlController", info.getCrawlController());
        fetcher.process(job);
        item.setContentStream((InputStream) job.get("contentStream"));
      }

      if (container) {
        rap.scan(item, info, conn, this);
      }
      processor.accept(item);
    }
    safeQueue.clear();
  }

  public boolean isQueueEmpty() {
    return queue.size()+safeQueue.size() == 0;
  }
}
```

Step 5 - Call the Fetcher class

This is required to extend the getComponent method and return the correct RAP object.

Fetcher class

```
private static class StandaloneFetchURL extends FetchURL {

  public static final String RAP = "rap";
  RepositoryAccessProvider rap;

  public StandaloneFetchURL(RepositoryAccessProvider rap) {
    this.rap = rap;
  }

  @Override
  public Component getComponent(String path) {
    if (RAP.equals(path)) {
      return (Component)rap;
    }
    return null;
  }
}
```

StandaloneCrawlController class

This class is only for extending the getNoSQLConnection method needed by some components to return a dummy NoSQLConnection object.

StandaloneCrawlController class

```
  private static class StandaloneCrawlController extends CrawlControllerImpl {

    RepositoryAccessProvider rap;
    public StandaloneCrawlController(RepositoryAccessProvider rap) {
      this.rap = rap;
    }

    @Override
    public RepositoryAccessProvider getRAP() {
      return rap;
    }

    @Override
    public NoSQLConnection getNoSQLConnection(String name, AspireObject properties) {
      return new EmptyNoSQLConnection();
    }
  }
```

EmptyNoSQLConnection class

As the name suggests, this is a dummy NoSQLConnection that doesn't t really do anything.

EmptyNoSQLConnection

```
public class EmptyNoSQLConnection implements NoSQLConnection {

      @Override
      public String getDatabase() {
        return null;
      }

      @Override
      public String getCollection() {
        return null;
      }

      @Override
      public void add(AspireObject item) throws AspireException {
      }

      @Override
      public void update(AspireObject item, String id) throws AspireException {
      }

      @Override
      public void update(AspireObject item, AspireObject filter)
          throws AspireException {
      }

      @Override
      public void updateAll(AspireObject item, AspireObject filter)
          throws AspireException {
      }

      @Override
      public void updateOrAdd(AspireObject item, AspireObject filter)
          throws AspireException {
      }

      @Override
      public boolean delete(String id) throws AspireException {
```

```
  return false;
}

@Override
public boolean delete(AspireObject filter) throws AspireException {
  return false;
}

@Override
public boolean deleteAll(AspireObject filter) throws AspireException {
  return false;
}

@Override
public AspireObject getOneAndUpdate(AspireObject filter,
    AspireObject update) throws AspireException {
  return null;
}

@Override
public AspireObject getOne(AspireObject filter) throws AspireException {
  return null;
}

@Override
public NoSQLIterable<AspireObject> getAll(AspireObject filter)
    throws AspireException {
  return null;
}

@Override
public NoSQLIterable<AspireObject> getAll(AspireObject filter, int skip)
    throws AspireException {
  return null;
}

@Override
public NoSQLIterable<AspireObject> getAll() throws AspireException {
  return null;
}

@Override
public NoSQLIterable<AspireObject> getAll(int skip)
    throws AspireException {
  return null;
}

@Override
public long size() throws AspireException {
  return 0;
}

@Override
public long size(AspireObject filter) throws AspireException {
  return 0;
}

@Override
public void clear() throws AspireException {
}

@Override
public void close() throws AspireException {
}

@Override
public AspireObject getAspireObject(Object obj) throws AspireException {
  return null;
}

@Override
```

```
      public void flush() {
      }

      @Override
      public void setBulkTimeout(long timeout) {
      }

      @Override
      public void setBulkSize(int size) {
      }

      @Override
      public void useBulk(boolean useBulk) {
      }

      @Override
      public AspireObject getOneAndUpdateOrAdd(AspireObject update,
          AspireObject filter) throws AspireException {
        return null;
      }

      @Override
      public AspireObject getOneAndDelete(AspireObject filter)
          throws AspireException {
        return null;
      }
}
```

For Legacy connector standalone crawls see Connector Scanner Stage Test Harness.