

Minio

# Table of Contents

Architecture.....	2
URL format .....	4
Integration solution (eg Apache Camel) .....	5
Classpath .....	5
Apache Camel Example Processor.....	5
Apache Camel Spring-based config.....	6
Apache Isis application.....	9
SPI .....	9
Classpath .....	10
Bootstrapping.....	10
Known issues / Limitations .....	11
Dependencies .....	12

This module (`incode-module-minio-dom`) and its various submodules provides a library for archiving BLOBs from a domain entity and persisting them instead using [Minio](#).

The idea is that domain entities such as documents, that large blob/clob content initially persist that content "in-line". At some later point, that content is copied from the domain entity into Minio, and then that content is removed. In its place the URL of the content as persisted in Minio is stored as a pointer.

Most of the functionality provided by this module is intended to be used by an integration solution - such as Apache Camel - rather than an Apache Isis domain application, and can be used "as-is".

The module also provides an SPI for the Apache Isis domain app to implement, acting as a bridge to the specifics of the domain app and the entities it may require archiving.

# Architecture

The diagram below sketches the components provided by this module, and how they interact:

The central algorithm is implemented by `MinioArchiver` (in the `incode-module-mini-o-mini-oarchlib` submodule). This:

- ¥ calls `DocBlobClient` to obtain a list of all documents that need to be archived.

The `DocBlobClient` is the `incode-module-mini-o-docclient` submodule.

- ¥ The `DocBlobClient` in turn makes a REST call to `DocBlobService`, hosted by the Apache Isis webapp.

The `DocBlobService` is in the `incode-module-mini-o-docserver` submodule, and so this is the one dependency that an Apache Isis webapp has on the Minio library.

- ¥ within the Apache Isis webapp, the `DocBlobService` delegates to the `DocBlobServiceBridge`.

This is an SPI that the consuming application is required to implement. Typically this is likely to delegate to some repository service to obtain entity/ies that have blobs, eg documents.

- ¥ for each of the returned blobs, the `MinioArchiver` then calls the `MinioBlobClient` to upload the blob into Minio itself. This returns back a URL.

The `MinioBlobClient` is in the `incode-module-mini-o-mini-oclient` submodule.

- ¥ for each blob and its corresponding URI, the `MinioArchiver` then calls the `DocBlobClient` once more, this time to indicate that the blob has been archived.

- ¥ the `DocBlobClient` in turn again makes a REST call up to `DocBlobService`, hosted on the Apache Isis webapp.

- ¥ once more the `DocBlobService` delegates to the `DocBlobServiceBridge` SPI.

The implementation of this (provided by the consuming application) will typically delete the blob from the original entity, and in its place store the URI as a pointer to Minio.

!

It's the responsibility of the consuming application to download the content from Minio (using the URL); this library does not (currently) provide any utility services for this.

The `MinioArchiver` described above is packaged as a library, so will typically be called periodically from some sort of integration solution. As the diagram above suggests, this could be accomplished using Apache Camel, whereby a `MinioArchiverProcessor` (not part of this library) acts as a simple wrapper that calls the `MinioArchiver`. That processor in turn might be scheduled to run periodically, say once an hour.

When first deploying this solution, there will probably be a need to archive historical blobs. The `Main` utility in the `incode-module-mini-o-mini-oarchtool` is a standalone utility that simply calls the

Mini oArchi ver to archive all.

# URL format

The format of the URLs created by `MinioBlobClient` is:

<http://minioserver/prod/myapp/cust.Customer/1234>

where:

- ¥ "http://minioserver" is the base URL which hosts the server
- ¥ "prod" is the S3 bucket to use; typically this represents an environment such as dev, test or production
- ¥ "myapp" is a fixed prefix. This represents the original app from which the Blob was obtained, and therefore how to interpret the remainder of the URL
- ¥ "cust.Customer/1234" is the identifier of (in this case) a customer. It corresponds to the bookmark of the Apache Isis application (having replaced '/' with ':').

!

The above scheme means that only one blob can be persisted per object instance. A future enhancement would be to allow multiple blobs to be persisted (corresponding to different properties of the original entity in the Apache Isis webapp).

# Integration solution (eg Apache Camel)

This section describes how to configure and use the minio library within the integration solution (eg Apache Camel), ie that periodically invokes the [Mi ni oArchiver](#).

## Classpath

Update your classpath by adding these dependencies to your [pom.xml](#) :

```
<dependency>
  Ê   <groupId>org.incode.module.minio</groupId>
  Ê   <artifactId>incode-module-minio-minio-client</artifactId>
</dependency>
<dependency>
  Ê   <groupId>org.incode.module.minio</groupId>
  Ê   <artifactId>incode-module-minio-archlib</artifactId>
</dependency>
<dependency>
  Ê   <groupId>org.incode.module.minio</groupId>
  Ê   <artifactId>incode-module-minio-docclient</artifactId>
</dependency>
```

Check for later releases by searching [Maven Central Repo](#).

## Apache Camel Example Processor

If using Apache Camel as the integration solution, then the code below can be used as a basis for a processor:

```

import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.incode.module.minio.minioarchlib.MinioArchiver;
import Lombok.Setter;

public class MinioArchiverProcessor implements Processor {

    private static final int MAX_ITERATIONS = 5;

    @Setter
    private MinioArchiver minioArchiver;

    @Override
    public void process(final Exchange exchange) {

        try {
            for (int i = 0; i < MAX_ITERATIONS; i++) {
                int numArchived = minioArchiver.archive("camel");
                LOG.info(numArchived + " archived");
                if (numArchived == 0) {
                    break;
                }
            }
        } catch (Throwable ex) {
            LOG.error(ex.getMessage());
        }
    }
}

```

This invokes the `MinioArchiver` up to 5 times. The idea here is to allow the archiving to be performed in batches, avoiding very large database updates during initial migration of blobs from the Apache Isis webapp and into Minio.

## Apache Camel Spring-based config

If running inside of Apache Camel and using Spring to configure the components:



```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:camel="http://camel.apache.org/schema/spring"
      xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/spring
        http://camel.apache.org/schema/spring/camel-spring.xsd">

  <bean id="minioArchiverProcessor"
        class="org.incode.ecp.est2minio.route.MinioArchiverProcessor">
    <property name="minioArchiver" ref="minioArchiver"/>
  </bean>

  <bean id="minioArchiver"
        class="org.incode.module.minio.minioarchlib.MinioArchiver">
    <property name="docBlobClient" ref="docBlobClient"/>
    <property name="minioBlobClient" ref="minioBlobClient"/>
  </bean>

  <bean id="minioBlobClient"
        class="org.incode.module.minio.minioclient.MinioBlobClient"
        init-method="init">
    <property name="url" value="${minio.baseUrl}"/>
    <property name="accessKey" value="${minio.accessKey}"/>
    <property name="secretKey" value="${minio.secretKey}"/>
    <property name="bucket" value="${minio.bucket}"/>
    <property name="prefix" value="${minio.prefix}"/>
  </bean>

  <bean id="docBlobClient"
        class="org.incode.module.minio.docclient.DocBlobClient"
        init-method="init">
    <property name="base" value="${apacheSisWebapp.baseUrl}"/>
    <property name="username" value="${apacheSisWebapp.username}"/>
    <property name="password" value="${apacheSisWebapp.password}"/>
  </bean>

</beans>

```

This requires the following configuration properties to be defined:

- ¥ `minio.baseUrl` - base URL for minio server (to upload to)
- ¥ `minio.accessKey` - user account to access minio
- ¥ `minio.secretKey` - corresponding password for the minio user account
- ¥ `minio.bucket` - as explained in the [above section](#) on the URL format, typically indicates the "environment"

- ¥ `minio.prefix` - as explained in the [above section](#) on the URL format, typically indicates the source of the blob
- ¥ `apacheIsisWebapp.baseUrl` - base URL for the Apache Isis webapp (to read blobs from)
- ¥ `apacheIsisWebapp.username` - user account to access Apache Isis webapp
- ¥ `apacheIsisWebapp.password` - corresponding password for the Apache Isis webapp

The Camel route that invokes the `MinioArchi verProcessor` (in the same file) is defined as:

```
<beans ...>
  ...

  <camelContext xmlns="http://camel.apache.org/schema/spring" id="minio">
    <route id="minioFromQuartz">
      <from uri="quartz://camel/estatioToMinio?cron=30+*+7-18+?+*+MON-FRI"/>
      <camel:process ref="minioArchi verProcessor"/>
    </route>
  </camelContext>
</beans>
```

Here the configuration for the `quartz` source is to run once an hour, from 7am to 6pm:

- ¥ `30` - seconds: at 30 seconds past the minute only
- ¥ `*` - minutes: every minute
- ¥ `7-18` - hours: from 7 til 18. First three parts imply therefore running every minute, 7:00 to 18:00
- ¥ `?` - day-of-month: omit, because cannot specify both this and also day-of-week (below)
- ¥ `*` - month-of-year: every month of the year
- ¥ `MON-FRI` - day-of-week: only Mondays to Fridays

!

'+' separates the parts (same as URL encoding a space)

# Apache Isis application

This section describes the responsibilities of the Apache Isis webapp that has domain entities with blobs that are to be archived.

## SPI

The consuming Apache Isis application is required to implement the `DocBlobServiceBridge` SPI:

*DocBlobServiceBridge.java*

```
public interface DocBlobServiceBridge {

    @Programmatic
    List<DocBlob> findToArchive(String caller);

    @Programmatic
    Blob blobFor(DocBlob docBlob);

    @Programmatic
    void archive(String docBookmark, String externalUrl);
}
```

where `DocBlob` is a view model that in effect just wraps the identity of the source entity which holds the blob:

*DocBlob.java*

```
@DomainObject(
    nature = Nature.VIEW_MODEL,
    objectType = "incodeminiio.DocBlob"
)
public class DocBlob implements ViewModel {

    public DocBlob(final String docBookmark) {
        this.docBookmark = docBookmark;
    }

    @Getter
    private String docBookmark;    !

    ...
}
```

! Bookmark of the persisted entity which holds the blob to be archived (or may have been archived).

In the implementation of this SPI, the application can create `DocBlob` instances simply using the regular `BookmarkService`:

```
final Bookmark bookmark = bookmarkService.bookmarkFor(entity);
return new DocBlob(bookmark.toString());
```

!

The `DocBlobServiceBridge` SPI is slightly inconsistent; the `archive()` method ought to take a `DocBlob` rather than a `docBookmark`.

## Classpath

Update your classpath by adding these dependencies to your `pom.xml`:

```
<dependency>
  <groupId>org.incode.module.minio</groupId>
  <artifactId>incode-module-minio-docserver</artifactId>
</dependency>
```

Check for later releases by searching [Maven Central Repo](#).

## Bootstrapping

The SPI service implementation must be included in the application bootstrapping. Typically this is done by defining an owning `Module` and then including that module in the application's `AppManifest`.

# Known issues / Limitations

- ¥ This implementation only supports one blob/clob property per domain type.
- ¥ It also doesn't distinguish between different domain types which may require archiving. This makes it the responsibility of the SPI to "assemble" the lists of all domain entities which may require archival (eg `Documents`, `Commands`, `PublishedEvents`) rather than each being archived separately.
- ¥ The library also doesn't provide any support for the consuming application to retrieving blobs from Minio. This is accomplished easily enough though, eg:

```
final String miniUrl = ...

final URL url = new URL(miniUrl);

final HttpURLConnection httpConn = openConnection(url);
if (httpConn == null) {
    return null;
}

final String contentType = httpConn.getContentType();

final MimeType mimeType = determineMimeType(contentType);
if (mimeType == null) {
    return null;
}

httpConn.disconnect();

final ByteSource byteSource = Resources.asByteSource(url);
final byte[] bytes = byteSource.read();

return new Blob(document.getName(), mimeType.getBaseType(), bytes);
```

# Dependencies

For the Apache Isis webapp, this library has no dependencies.