WHITEPAPER

Migrating from JCAPS to Anypoint Platform

A step-by-step guide

Summary

This document describes how JavaCAPS (JCAPS) users can migrate from JCAPS to MuleSoft's Anypoint Platform in phases, relieving the risks associated with doing a complete migration and replacement at once. Learn how to accomplish the full migration in this step-by-step guide.

Introduction

JCAPS is a legacy SOA product from Oracle. It was originally developed by SeeBeyond, which was later acquired by Sun Microsystems. JCAPS became Sun's flagship SOA product until Oracle acquired Sun, at which point JCAPS became a legacy product. Oracle required JCAPS customers to migrate to their own Oracle Fusion Middleware for continued support.

This is often the main reason why JCAPS customers are looking for a replacement integration platform, but there are other reasons driving this change:

- **Proliferation and adoption of SaaS systems**: JCAPS does not offer any SaaS connectors.
- **Need for a cloud integration platform**: JCAPS does not have an integration Platform as a Service (iPaaS) option.
- **API management platform**: JCAPS does not offer an API management platform or a connector that easily connects or exposes REST APIs. It also lacks a SOA governance solution.

MuleSoft's Anypoint Platform addresses these reasons with an enterprise-grade integration solution. Deploy Anypoint Platform on-premises with Mule ESB or in the cloud with CloudHub to experience these benefits:

Enterprise Capabilities

- Highly available clusters and an in-memory data grid
- Enterprise management dashboards including flow analyzer and business events

Productivity Improvements over JCAPS

- Full-featured ESB including Enterprise Integration Patterns (EIPs) and flow controls
- Graphical data mapper with lookup tables and Mule Expression Language
- Developer lifecycle integration (Maven, Git, CI, etc.)

Business Benefits

- Ability to migrate in phases rather than doing a complete rewrite
- Faster time-to-market with existing JCAPS code
- Unified, future-proof platform covering SaaS, SOA and API
- Award-winning customer support
- Open source platform with over 210,000 developers

This paper describes how to leverage these benefits through a phased migration to Mule ESB based on best practices. *The methods in this paper apply to JCAPS versions 5.x and 6.x SRE.*

Phases of Migration

The methodology described in this paper allows you to migrate from JCAPS in multiple phases. This is a key differentiator from other vendors and allows for faster time to market for new Mule ESB projects. There are three phases of this migration: co-exist, migrate, rewrite.

Phase 1: Co-exist strategy - Keep JCAPS runtime

The first phase of the migration involves integrating with Mule ESB without rewriting JCAPS. By leveraging endpoints such as JMS, SOAP, and File, you can add incremental features and new projects using Mule ESB, while keeping the existing JCAPS integrations intact that are already in production.

Phase	Tools			
	JCAPS IDE	JCAPS runtime	Anypoint Studio (MuleSoft's IDE)	Mule ESB
1) Co-exist	✔	✔	✔	✔

JCAPS to ESB communication options

JMS Connectivity

SeeBeyond JMS Server (also known as eGate IQ Manager or STCMS) is the standard JMS server most often used by JCAPS users. To connect to this JMS server, use MuleSoft's JMS Connector.

Instructions on connecting to SeeBeyond JMS server are available at:
http://docs.oracle.com/cd/E19336-01/819-6852/819-6852.pdf

1. Locate the JAR files required to connect to STCMS. These are usually available in your runtime (STCIS) location or available via the eGate JMS API Kit (for example, com.stc.jms.jmsis.jar, com.stc.jmsis.jar, com.stc.jms.stcjms.jar).
2. Add the MuleSoft JMS Connector to your flow.
3. Configure the JMS connectors *JNDI Connection* options so it can communicate with STCMS:

```
[Connection Factories]
JMSJCA.UnifiedCF=connectionfactories/xaconnectionfactory
JMSJCA.TopicCF=connectionfactories/xatopicconnectionfactory
JMSJCA.QueueCF=connectionfactories/xaqueueconnectionfactory

[Factory Properties]
java.naming.factory.initial=com.stc.jms.jndispi.InitialContextFactory
java.naming.provider.url=stcms://localhost:18007
java.naming.security.principal=Administrator
java.naming.security.credentials=STC
```

Web Service Connectivity

Use the standard SOAP/REST/HTTP connectivity options within Mule ESB to talk to SeeBeyond web services, especially to invoke eInsight/BPEL running on the SeeBeyond Integration Server. JCAPS can invoke HTTP, REST, or SOAP-based services hosted by Mule ESB. Similarly, Mule ESB can invoke HTTP, REST, or SOAP-based services on JCAPS.

Phase 2: Migrate existing investment in Java Collaboration Definitions - Deprecate JCAPS runtime and keep eDesigner

For phase 2, migrate the Java Collaboration Definitions (JCD) and transform logic, and host it as is in Mule ESB with the Java message processor. You can use Spring to wrap Object Type Definition (OTD) objects as beans or POJOs so that they can be used in the JCAPS native format from within the MuleSoft's graphical data mapper.

Use this phase to migrate from JCAPS' production and go live with Mule ESB as the runtime for legacy JCAPS integrations. You can use eDesigner during the migration effort but not during the runtime. Use Mule ESB or CloudHub as the runtime instead of the SeeBeyond Integration Server or GlassFish Application Server.

Any tweaks to the OTD used within Mule ESB require revisiting the OTD and refreshing it using eDesigner, so the JCAPS IDE may still be needed during this phase. The migration from the JCAPS runtime shields you from potential production downtime issues once Oracle discontinues support. Also, now that the production environment is running on Mule ESB runtime, you will be fully supported with MuleSoft.

Phase	Tools			
	JCAPS IDE	JCAPS runtime	Anypoint Studio (MuleSoft's IDE)	Mule ESB
1) Co-exist	✔	✔	✔	✔
2) Migrate	✔	✗	✔	✔

Phase 3: Rewrite in Mule ESB - Complete the JCAPS migration

Re-implement the Java Collaboration Definition (JCD) using Mule ESB for business logic, MuleSoft's connectors instead of JCAPS eWays, and MuleSoft's graphical data mapper instead of JCD's graphical mapper. Neither JCAPS runtime nor IDE are required from here onwards.

Phase	Tools			
	JCAPS IDE	JCAPS runtime	Anypoint Studio (MuleSoft's IDE)	Mule ESB
1) Co-exist	✔	✔	✔	✔
2) Migrate	✔	✗	✔	✔
3) Rewrite	✗	✗	✔	✔

Enhance JCAPS migrated code

The JCAPS code that was migrated can now be significantly enhanced to take advantage of the new capabilities available in Mule ESB such as using API Gateway to API-enable legacy JCAPS integrations, API Manager to govern and enforce runtime policies, and deploy legacy JCAPS applications into MuleSoft's iPaaS, CloudHub.

Migration Methodology

The components in JCAPS that can be migrated *mostly* as-is are:

1. Java Collaboration Definition (JCD)
2. Object Type Definitions (OTD)

For BPEL (eInsight), the best options to migrate are to use the MuleSoft jBPM/Activiti component for human workflow and regular Mule flows for orchestration. This means BPEL projects cannot be migrated as-is and require a rewrite, but 100% functional parity is available. Next we discuss how to repurpose existing JCD's and OTD's.

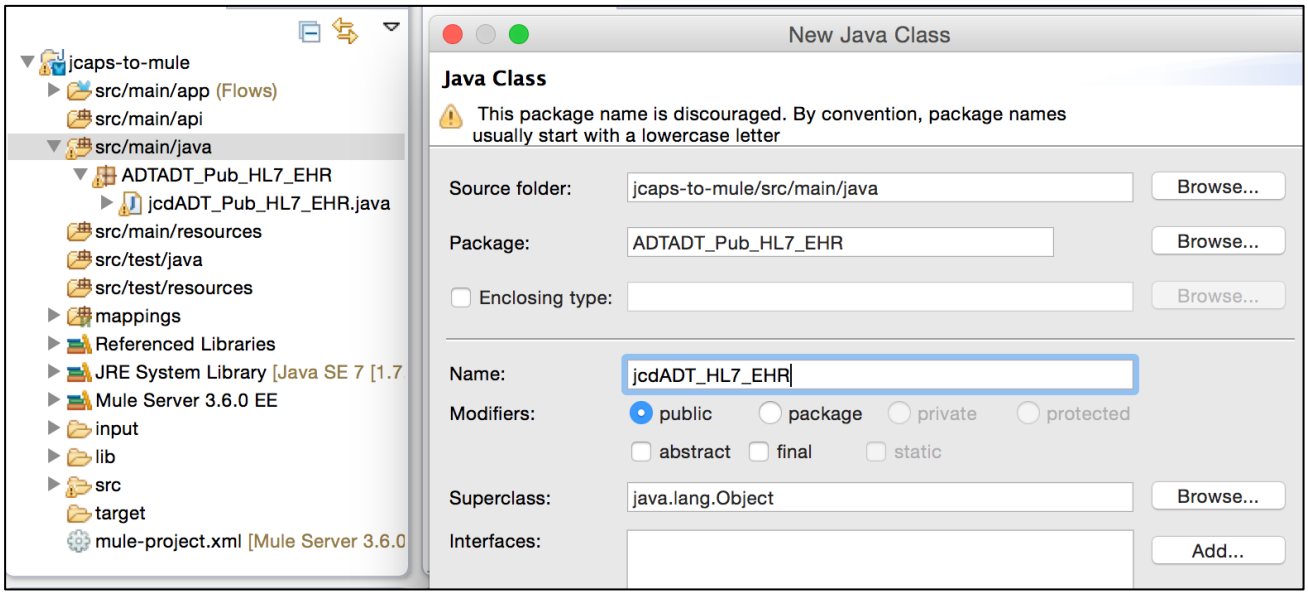
High-level guidelines

- Re-use OTD's and related transformations in JCD
    - Import all *com.stc.** base JARs and OTD JARs from the EAR file for the project (build the EAR from eDesigner)
    - Note: Large number of JARs not an issue – Mule only loads classes from the JARs it actually uses, which is a very small subset
- Remove eWay connector-specific operations from JCD. For instance, if JMS is being used:
    - Remove the jmsOTD.send() operation from JCD
    - Keep the get/set operations on jmsOTD since they hold data
    - Add a MuleSoft JMS connector to your Mule flow
    - Pass output data from the JCD transformer to the MuleSoft JMS connector (for example, jmsOTD.marshalToString() becomes the payload for the JMS connector)
- Remove business logic from the JCD and make the logic part of the Mule flow:
    - For example, bubble up routing and orchestration such as: *if X then file.write() else jms.send()*
    - Leverage EIP's (Enterprise Integration Patterns) and flow controls as first-class objects in the Mule flow and leave only transforms in the JCD
    - Note: This gives more readable integrations versus business flow logic being embedded in one big JCD class

Detailed steps

Mule Project

1. In Anypoint Studio, click **File** > **New** > **Mule Project**
2. Create a new Java Class in the Mule Project, under /src/main/java
3. Use the existing JCD's package name and JCD class name for this new class and leave everything else as default values

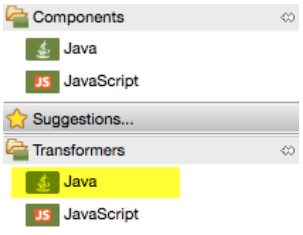


Java Transformer

1. Copy and paste your existing JCD's Java source code and replace this new empty class code
2. Add 'extends org.mule.transformer.AbstractTransformer' to your JCD class, so that it can be invoked from the Mule Java Transformer:

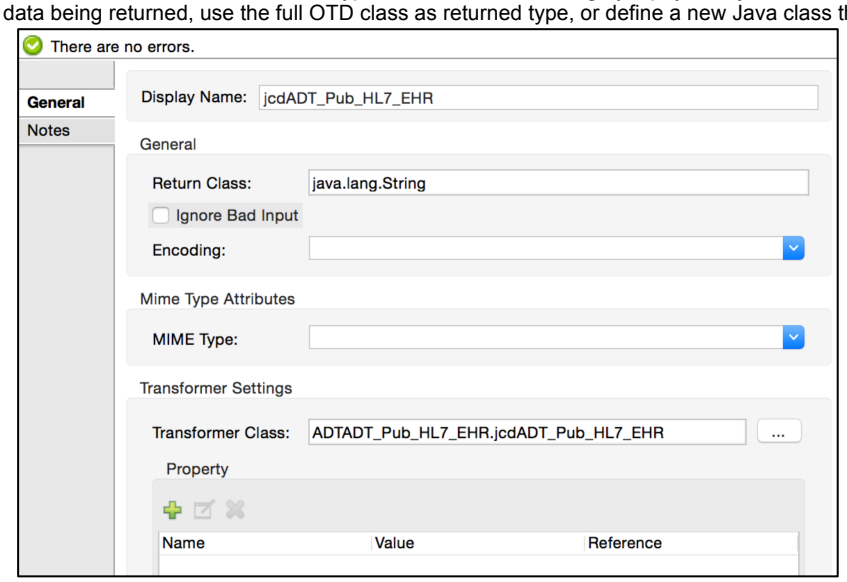
```
import java.text.*;

// Make JCD class extend AbstractTransformer to
// implement the Mule Java Transformer
public class jcdADT_Pub_HL7_EHR extends AbstractTransformer
{
      // Program Header
      ...
}
```

3. Drag and drop a **Java Transformer** into your flow from the palette. This hosts and invokes the existing JCD class and returns the output of the existing JCD transform. (Note: We use the Java *Transformer*, not Java *Component*.)



4. Point to the newly defined JCD class as the "Transformer Class"

5. Define the return class. For File and JMS-type use cases with a string/byte payload, you can use java.lang.String or Byte as the return type. For more complex data being returned, use the full OTD class as returned type, or define a new Java class that wraps multiple OTD classes as private variables.



JCD Logger

1. From your JCD, remove `public com.stc.codegen.logger.Logger logger`
2. You automatically have access to the Mule log4j logger, so log statements in your JCD continue to work as-is

doTransform() method

1. To invoke the JCD from the Mule Java Transformer, add a doTransform() method in the JCD
2. Add this new method to your JCD. This is the entry point into your JCD and from here you can invoke the existing JCD receive() method.

```
public Object doTransform(Object src, String encoding) throws TransformerException {
        logger.info("Inside JCD");
        String outputJMSMessage = "";
        logger.info(outputJMSMessage);
        return outputJMSMessage;
  }
```

Receive() method

- The main method in the JCD is the receive() method. In the JCAPS runtime, the JCD framework executes your JCD by invoking this method
- We invoke this receive() method in the JCD as well, but from our doTransform() method
- But first, remove all JCAPS connectors (eWays) used in the receive() method arguments to use MuleSoft's connectors outside the JCD:

```
// Comment out JCAPS eWays to use MuleSoft connectors
public xsd.ADT378381852.Hl7 receive( com.stc.connectors.jms.Message input,
       xsd.ADT378381852.Hl7 ADT_hl7_1, udl.HL7_23_ADT_A0839804825.ADT_AA08 HL7_23_ADT_A08_1)
       //, com.stc.connectors.jms.JMS JMS_1,
       // com.stc.connectors.jms.JMS JMS_Rollback )
     throws Throwable
{
     xsd.ADT378381852.Hl7 ret - null;
     logger.info(input.toString());
     System.out.println(input.toString());

     try {
          // ...
}
```

Initialize OTD's

- In the JCAPS runtime, the framework auto-instantiates all OTD's in the receive() method input arguments, but in this case, you have to instantiate them yourself
- Do this from the newly created doTransform() method

```
public Object doTransform(Object src, String encoding) throws TransformerException {
    logger.info("Inside JCD");
    String outputJMSMessage = "";
    xsd.ADT378381852.Hl7 HL7 = new xsd.ADT378381852.Hl7();
    ud1.HL7_23_ADT_A0839804825.ADT_A08 ADT = new ud1.HL7_23_ADT_A0839804825.ADT_A08();
    com.stc.connectors.jms.impl.MessageImpl inputJMSMessage = new com.stc.connectors.jms.impl.MessageImpl();
    inputJMSMessage.setBytesMessage(((String) src).getBytes());
    try{
        xsd.ADT378381852.Hl7 ADTOutput = receive (inputJMSMessage, HL7, ADT);
        outputJMSMessage = ADTOutput.marshalToString();
    } catch (Throwable ex){
        ex.printStackTrace();
    }
    logger.info(outputJMSMessage);
    return outputJMSMessage;
}
```

Remove all eWay operations

1. Comment out any eWay operations like send() or execute()
   (Actual operation names depends on the JCAPS eWay used)
   In this example, the eWay used is JMS, so remove the JMS send() operation

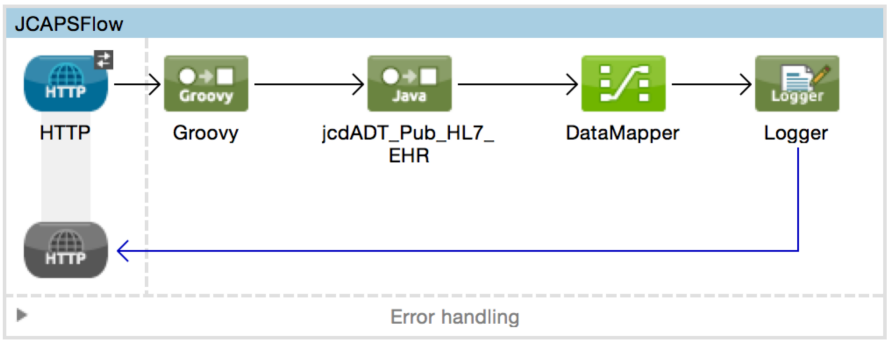
```
   if (HL7_23_ADT_A08_1.getGroupBatch( m1 ).getZpt( i2 ).hasZpt5HomeUnitIndicator()) {
      ADT_hl7_1.getDva().setHomeUnitIndicator( HL7_23_ADT_A08_1.getGroupBatch( m1 ).getZpt( i2
).getZpt5HomeUnitIndicator() );
   }
   log( "log-" + sProjectName + ": ReferringPhysicianID[" + ADT_hl7_1.getPv1().getReferringPhysicianID() + "]", 0,
runPubMsg );
   // SEND TO JMS
   //sendToJMS( ADT_hl7_1, JMS_1, sRecordKey, ep, runPubMsg, sRunId, sMsgRunID, sMsgType );
   iProcessed++;
   postRunRecordLevel( ep, "PUB_REC_PROCESSED", ADT_hl7_1.marshalToString(), runPubMsg );
} catch ( Exception e ) {
   e.printStackTrace();
   log( "log-" + sProjectName + ": Mapping failed " + e.toString(), 0, runPubMsg );
```

2. Return from the JCD the message that was originally sent using the JMS eWay. After this message returns to the Mule flow, use the MuleSoft JMS connector to send the message instead.
3. Do this using the marshal methods that are available on most eWays, for example, return payload as string or bytes using the OTD's marshalToString() or marshalToBytes() method.
4. Route this message to the MuleSoft JMS connector instead, by returning this string/byte from the doTransform() method

```
// doTransform() method which we invoke by Mule Java Transformer
public Object doTransform(Object src, String encoding) throws TransformerException {
    logger.info("Inside JCD");
    String outputJMSMessage = "";
    xsd.ADT378381852.Hl7 HL7 = new xsd.ADT378381852.Hl7();
    ud1.HL7_23_ADT_A0839804825.ADT_A08 ADT = new ud1.HL7_23_ADT_A0839804825.ADT_A08();
    com.stc.connectors.jms.impl.MessageImpl inputJMSMessage = new com.stc.connectors.jms.impl.MessageImpl
    inputJMSMessage.setBytesMessage(((String) src).getBytes());
    try{
        xsd.ADT378381852.Hl7 ADTOutput = receive (inputJMSMessage, HL7, ADT);
        outputJMSMessage = ADTOutput.marshalToString();
    } catch (Throwable ex){
        ex.printStackTrace();
    }
    logger.info(outputJMSMessage);
    return outputJMSMessage;
}
```

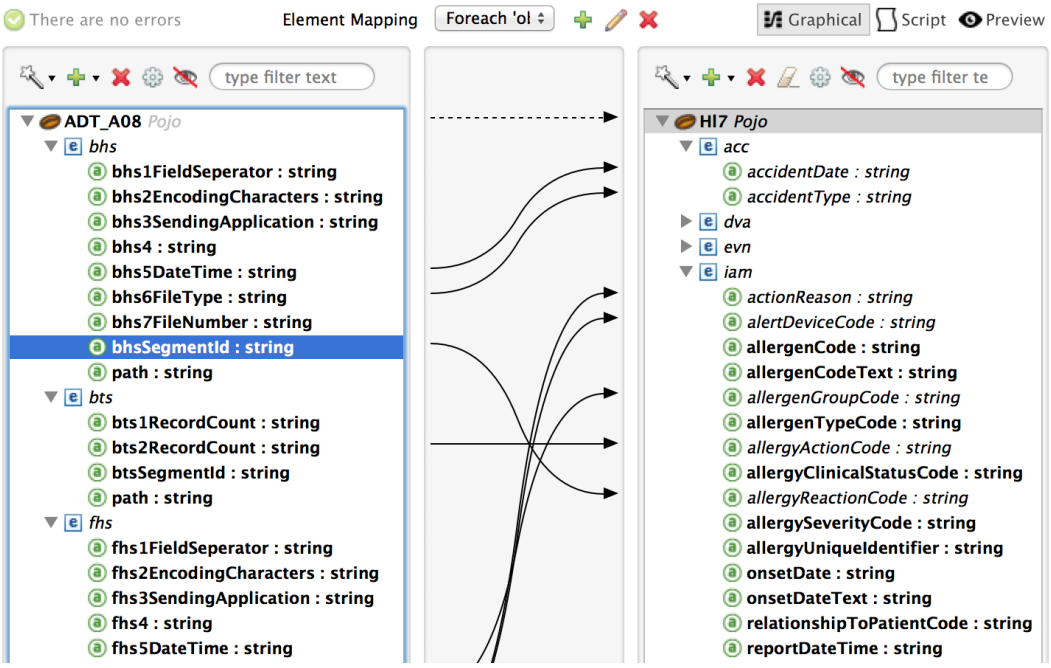
Design a Mule flow

1. In Studio, drag and drop and use all the Mule components you need to build your flow.
2. In this example, you can re-use the transformations within the JCD without tight coupling to JMS.
3. Now reuse the JCD transformations to build, for example, a REST API using the HTTP connector instead (or with JMS in and out, File in and File out, etc.)
4. The key is that the connectors can be swapped out but the transformation remains the same as with the original JCAPS JCD. Note how this is a vast improvement from a modular/reuse perspective compared to JCAPS.



Reuse OTDs, if needed

- OTD's are simply JCAPS-generated Java POJO's with getProperty() and setProperty() methods
- We can use these OTD's as Spring beans (POJO's) in the MuleSoft's graphical mapper as seen below



- This concludes the migration. You can now run and deploy your Mule application and execute your JCD code.

This paper described how to migrate in three phases from JCAPS to Mule ESB. Once migrated to Mule ESB, JCAPS users can take full advantage of the broad capabilities available through Anypoint Platform. Learn more about MuleSoft's Anypoint Platform by contacting us at info@mulesoft.com or get started with Mule ESB today.

MuleSoft

MuleSoft's mission is to connect the world's applications, data and devices. MuleSoft makes connecting anything easy with Anypoint Platform™, the only complete integration platform for SaaS, SOA and APIs. Thousands of organizations in 60 countries, from emerging brands to Global 500 enterprises, use MuleSoft to innovate faster and gain competitive advantage.