Brian Owings
FileNet Development
Nov 2014

Product Implementation Training (PIT)

FileNet P8 v5.2.1

Transcription: Building a Transcription Handler

Course Objectives

This course is intended for:

- Those supporting business partners in the implementation of transcription handlers.
- Those diagnosing issues related to a vendor-supplied transcription handler.

Useful prerequisites:

- Detailed knowledge of CPE API's for building engine extensions.
- Use of the Admin Client for configuration.

After this course you should be able to:

- Understand the overall design of a transcription handler.
- Understand the interfaces used in building the handler.
- Understand how to install and configure a transcription handler.

Course Roadmap

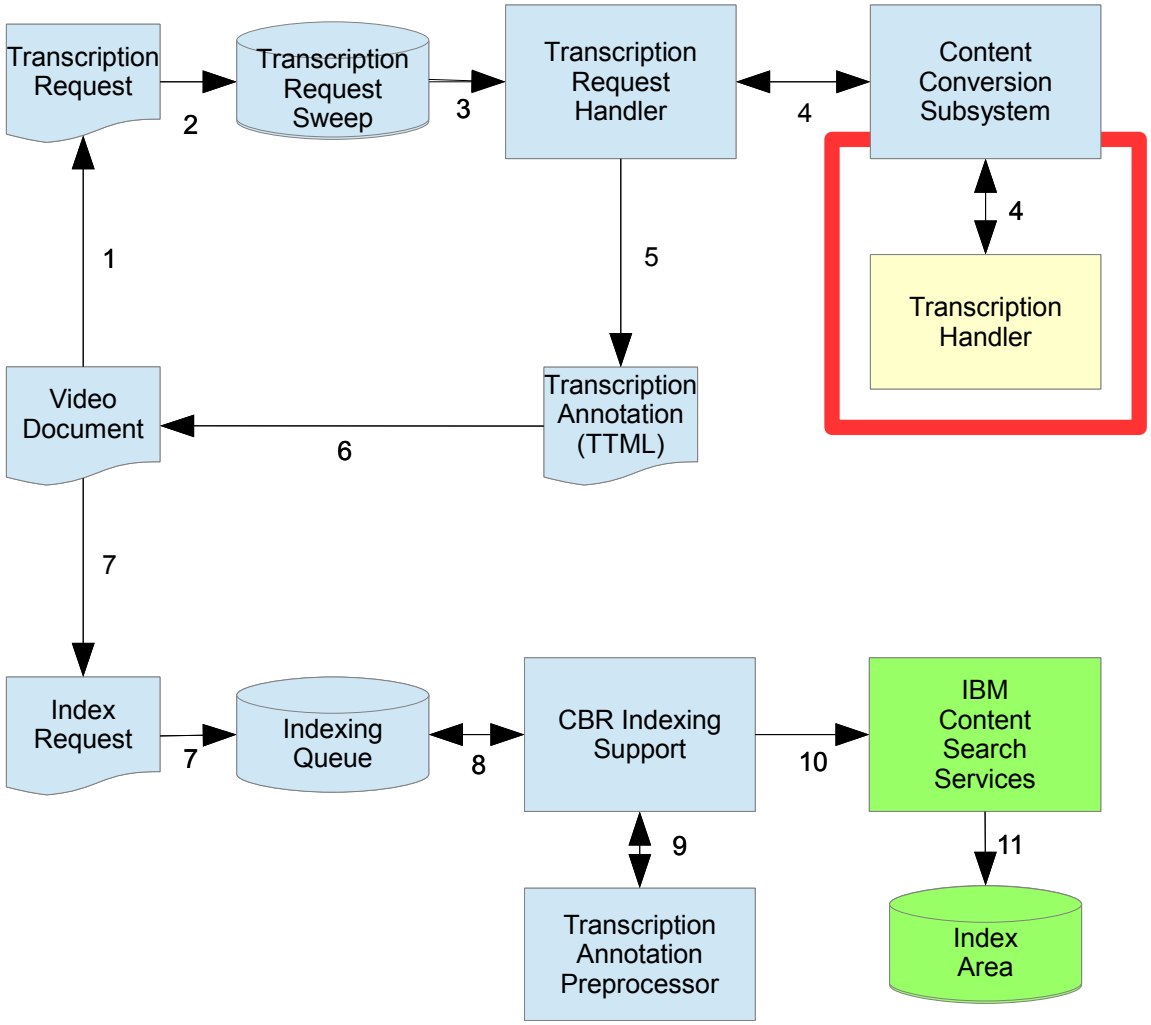
- Overview
- Implementing the Transcription Handler
    - Interfaces
    - Requirements
- Creating an Installer for the Transcription Handler
- Course Summary

Best practices mentioned throughout the course.

Course Roadmap

- Overview
- Implementing the Transcription Handler
  - Interfaces
  - Requirements
- Creating an Installer for the Transcription Handler
- Course Summary

Overview



Overview

The transcription handler:

- Plugs into the content conversion subsystem (acting as a conversion handler)
- Is packaged in a JAR, loaded into P8 as a Code Module.
- Is described within the model of the server by an instance of CmContentConversionAction server subclass.
- Implements the ContentConversionHandler interface:

```
interface ContentConversionHandler {

        boolean requiresFile(String outputConversionClassName);

        boolean isSynchronous(String outputConversionClassName);

        void convertStream(CmContentConversionAction action,
                InputStream source, String sourceContentType, String retrievalName,
                CmContentConversionSettings outputSettings,
                ContentConversionResult result);

        void convertFile(CmContentConversionAction action,
                String sourceFile, String sourceContentType, String retrievalName,
                CmContentConversionSettings outputSettings,
                ContentConversionResult result);

        void resumeConversion(CmContentConversionAction action,
                byte[] deferralData,
                CmContentConversionSettings outputSettings,
                ContentConversionResult result);

};
```

File or Stream?

A transcription handler can be passed content as a file or as a stream:

- File can be more straightforward
- File allows admin to configure the temp file storage
- Stream is more efficient – even if content is stored as files it is copied.
- File will decript the content and store it on the file system temp file area.

Recommendation: Use files *if* the transcription uses files directly for content, *otherwise*, use streams.

Synchronous or Asynchronous?

A transcription handler can be either synchronous or asynchronous:

- Synchronous converts in *one call* to the handler.
- Asynchronous requires *multiple calls*:
  - A call to start
  - Repeated calls to determine if conversion is done and return the results (polling).

Pro's and cons:

- Synchronous is simpler to implement
- Synchronous ties up a thread on the P8 server
- Use of synchronous is only for fast conversion
- Transcription conversion usually takes a long time, therefore:

Recommendation: Use asynchronous for transcription

Additional Servers

To implement asynchronous transcription, you are likely to use a separate server.

Why?

- P8 is normally deployed as a cluster, this means that a different P8 server in the cluster may ask for the results of an earlier asynchronous transcription request.
- P8 is scalable therefore transcription should also be scalable.
- Transcription is likely resource intensive, requiring its own server(s).

Course Roadmap

- Overview
- Implementing the Transcription Handler
  - Interfaces
  - Requirements
- Creating an Installer for the Transcription Handler
- Course Summary

Interfaces

These interfaces are used in building the transcription handler:

- ContentConversionHandler – the main interface implemented by the handler
- ContentConversionResult – used to return results from the handler
- CmContentConversionAction – defines the handler
- CmContentConversionSettings – describes the desired results of the handler
- HandlerCallContext – additional utilities available

All of these are available in Jace.jar.

ContentConversionHandler

The main interface implemented to provide a handler:

```
interface ContentConversionHandler {

        boolean requiresFile(String outputConversionClassName);

        boolean isSynchronous(String outputConversionClassName);

        void convertStream(CmContentConversionAction action,
                    InputStream source, String sourceContentType, String retrievalName,
                    CmContentConversionSettings outputSettings,
                    ContentConversionResult result);

        void convertFile(CmContentConversionAction action,
                    String sourceFile, String sourceContentType, String retrievalName,
                    CmContentConversionSettings outputSettings,
                    ContentConversionResult result);

        void resumeConversion(CmContentConversionAction action,
                    byte[] deferralData,
                    CmContentConversionSettings outputSettings,
                    ContentConversionResult result);
};
```

ContentConversionHandler requiresFile

```
boolean requiresFile(String outputConversionClassName);
```

- Informs conversion services to pass content using files
- Return *true* to use files, *false* to use streams
- Argument:

  `outputConversionClassName` – the name of the ContentConversionSettings class associated with the conversion.

ContentConversionHandler isSynchronous

```
boolean isSynchronous(String outputConversionClassName);
```

- Informs conversion services that synchronous convert is desired
- Returns *true* for synchronous and *false* for asynchronous
- Argument:

  `outputConversionClassName` – the name of the ContentConversionSettings class associated with the conversion

ContentConversionHandler convertStream, convertFile

```
void convertStream(CmContentConversionAction action,
                   InputStream source, String sourceContentType, String retrievalName,
                   CmContentConversionSettings outputSettings,
                   ContentConversionResult result);

void convertFile(CmContentConversionAction action,
                   String sourceFile, String sourceContentType, String retrievalName,
                   CmContentConversionSettings outputSettings,
                   ContentConversionResult result);
```

- Called to perform the conversion (stream or file)
- For asynchronous, this only starts the conversion. For synchronous, the entire conversion is performed during the call.
- Arguments:
  - `action` – the action defining the conversion handler
  - `source / sourceFile` – the stream or file containing the media to convert
  - `sourceContentType` – the MIME content type of the source
  - `retrievalName` – the original file name used to import the content
  - `outputSettings` – describes the desired conversion
  - `result` – provided to return the results of the conversion

ContentConversionHandler resumeConversion

```
void resumeConversion(CmContentConversionAction action,
                      byte[] deferralData,
                      CmContentConversionSettings outputSettings,
                      ContentConversionResult result);
```

- Used for asynchronous conversions only.
- Allows for polling for the results of the conversion.
- Arguments:
  - `action` – the action defining the conversion handler
  - `deferralData` – data returned from the original convertStream or convertFile
  - `outputSettings` – describes the desired conversion
  - `result` – provided to return the results of the conversion

ContentConversionResult

- Passed to ContentConversionHandler to return transcription results.
- Only the *setter* methods (shown below) are used by the handler:

```
interface ContentConversionResult {

        void setStreamResult(InputStream s);

        void setDeferral(byte[] deferralData, int deferralSeconds);

        void setAbandonedResult();

        void setUnableResult();

        ..

};
```

ContentConversionResult Methods

```
void setStreamResult(InputStream s);
```

- Returns the conversion results in the form of a stream.
- Argument:
  `s` – a stream containing the results as TTML.

```
void setDeferral(byte[] deferralData, int deferralSeconds);
```

- Use on asynchronous only.
- Indicates the results are not yet available.
- Arguments:
  `deferralData` – a value used to correlate the conversion call with the resumeConversion call (like a “cookie”)
  `deferralSeconds` – number of seconds (minimum) before polling

```
void setAbandonedResult();
```

- Indicates the transcription couldn't be performed or had to be stopped
- The transcription might be retried by P8 (depending on transcription sweep queue configuration)

```
void setUnableResult();
```

- Indicates the transcription cannot be performed. (It would never succeed, even if retried.)
- If another transcription handler exists, P8 will try to use it.

CmContentConversionAction

- Created by the transcription handler installer (we'll cover that later)
- References the code module for the transcription handler
- Holds the configuration information for the transcription handler

```
public interface CmContentConversionAction
            extends com.filenet.api.core.RepositoryObject, com.filenet.api.events.Action
    {
            public Boolean get_IsEnabled();
            public void set_IsEnabled(Boolean value);

            public CmConversionSettingsClassDefinition get_CmOutputConversion();
            public void set_CmOutputConversion(CmConversionSettingsClassDefinition value);

            public StringList get_CmSupportedInputContentTypes();
            public void set_CmSupportedInputContentTypes(StringList value);

            public StringList get_CmUnsupportedInputContentTypes();
            public void set_CmUnsupportedInputContentTypes(StringList value);

            public Integer get_Priority();
            public void set_Priority(Integer value);
    }
```

CmContentConversionAction Methods

- Typically only getProperties is used in the handler:

```
public Properties getProperties();
```

This would be used to access custom properties defined for the handler. For example:

```
String transcriptionServiceURI = action.getProperties().getStringValue("TranscriptionServiceURI");
```

- Other methods will be used in creating an instance:

```
public void set_CmSupportedInputContentTypes(StringList value);
```

  - Sets the list of supported content types for the conversion.
  - For transcription of video, this will be video content types.

```
public void set_CmUnsupportedInputContentTypes(StringList value);
```

  - Sets the list of unsupported content types.
  - This will be a more qualified versions of the supported types:
    For example, if video/* is the supported, video/mpeg might be unsupported

CmContentConversionSettings

Describes the type of conversion desired.

```
public interface CmContentConversionSettings
                        extends     com.filenet.api.core.RepositoryObject,
                                    com.filenet.api.core.Subscribable,
                                    com.filenet.api.core.IndependentlyPersistableObject
    {

            public String get_CmOutputContentType();

            .. other methods ..

    }
```

CmContentConversionSettings Methods

```
public String get_CmOutputContentType();
```

- Returns the MIME content type of the desired result.
- For transcription this is always application/ttml+xml.

```
public Properties getProperties();
```

- For RmsTranscriptSettings, the following properties are defined:
  - TranscriptionLocale – the desired locale of the transcription.

Other properties can be defined as needed. For example:

- Settings to tune speed vs accuracy in transcription
- Information about to the number of speakers in the audio

Note: Any property values on the ContentConversionSettings object will be propagated to like-properties defined on the transcript annotation.

HandlerCallContext

- Contains utility methods useful in handlers.
- Accessible from a transcription handler using:

```
HandlerCallContext.getInstance()
```

Some useful methods:

```
public void logWarning(Object obj)
public void logError(Object obj)
public void traceSummary / traceModerate / traceDetailed(Object obj)
public boolean isDetailTraceEnabled / isModerateTraceEnabled / isSummaryTraceEnabled()
```

- to implement logging and tracing
- integrated into the Filenet P8 server logs.

```
public String getTemporaryFilesDirectory()
```

- the directory to write temporary files.

```
public boolean isShuttingDown()
```

- to indicate P8 server is shutting down.

Requirements on Transcription Handler

Some requirements on all conversion handlers:

- Handler can only use the stream or file passed to it during the convert method call
- Handler must delete all temporary files it creates.
    - CE server recycle will delete all temp files.
- `convertFile` must treat source file as read-only
- Result stream might be read by a different thread – cannot rely on thread-local storage
- Handler should not use large memory areas:
    - avoid memory use for each pending transcription
    - avoid in-memory streams

Additional requirements for async handlers:

- `ContentConversionResult.setUnableResult` can only be used on convert methods, not from resumeConversion
- `resumeConversion` may never be called (due to admin deleting the request), therefore the handler should auto-abandon after some length of time

Course Roadmap

- Overview
- Implementing the Transcription Handler
    - Interfaces
    - Requirements
- → Creating an Installer for the Transcription Handler
- Course Summary

Creating an Installer

Steps to install a transcription handler:

1. Create a code module containing the transcription handler's JAR
2. Define a subclass of CmContentConversionAction
   - Add any needed configuration properties to this subclass
3. Create an instance of the subclass and associate it with the code module
4. (Optional) Define a subclass and instance of RmsTranscriptionSettings
   - Add any per conversion-related options to this subclass

1. Installing a Code Module

```
// Create CodeModule object
CodeModule newCM = Factory.CodeModule.createInstance(objectStore, "CodeModule");
newCM.getProperties().putValue("DocumentTitle", "TranscriptionHandler Module”);

// Create ContentTransfer object from the JAR content
ContentTransfer ctNew = Factory.ContentTransfer.createInstance();
ctNew.setCaptureSource(fileIS);
ctNew.set_ContentType("application/java-archive");

// Add JAR content element to the CodeModule object
ContentElementList contentList = Factory.ContentElement.createList();
contentList.add(ctNew);
newCM.set_ContentElements(contentList);

// Check in CodeModule object
newCM.checkin(AutoClassify.DO_NOT_AUTO_CLASSIFY, CheckinType.MAJOR_VERSION);
newCM.save(RefreshMode.REFRESH);

// Add the code module to the /CodeModules folder
Folder folder = Factory.Folder.fetchInstance(objectStore, "/CodeModules", null);
DynamicReferentialContainmentRelationship drcr = (DynamicReferentialContainmentRelationship) folder.file(newCM,
      AutoUniqueName.AUTO_UNIQUE, "ContentConversionHandlerJar_" + vendorName,
DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE);
drcr.save(RefreshMode.NO_REFRESH);
```

2. Creating a subclass of ContentConversionAction

```
// Create a subclass of CmContentConversionAction
ClassDefinition cdCCA = Factory.ClassDefinition.fetchInstance(objectStore, ClassNames.CM_CONTENT_CONVERSION_ACTION, null);
String className = "MyTranscriptionConversionAction”;
Id classId = Id.createId();
ClassDefinition cdSubCCA = superClass.createSubclass(classId);
LocalizedStringList lsl = Factory.LocalizedString.createList();
lsl.add(getLocalizedStringEntry(className));
cdSubCCA.set_DisplayNames(lsl);
cdSubCCA.getProperties().putValue(PropertyNames.SYMBOLIC_NAME, className);
cdSubCCA.save(RefreshMode.REFRESH);

// Add custom configuration properties to the subclass
PropertyTemplateString ptStringURI = createPropertyTemplateString("TranscriptionServiceURI", Cardinality.SINGLE, new Integer(256), null, false);
PropertyTemplateString ptStringCredentials = createPropertyTemplateString("TranscriptionServiceCredentials", Cardinality.SINGLE, new Integer(32),
      null, false);
addPropertyDefinitionString(cdSubCCA, ptStringURI, null, null);
addPropertyDefinitionString(cdSubCCA, ptStringCredentials, null, null);
cdSubCCA.save(RefreshMode.REFRESH);
```

3. Creating an Instance of the Action

```
// Create the instance
CmContentConversionAction cca = Factory.CmContentConversionAction.createInstance(objectStore, actionClass);
cca.set_DisplayName(name);
cca.set_IsEnabled(Boolean.TRUE);

// Link to the code module
cca.set_ProgId(progId);
cca.set_CodeModule(cm);

// Set the supported and unsupported types
StringList supportedTypes = Factory.StringList.createList();
supportedTypes.add("video/mp4");
supportedTypes.add("audio/wav");
cca.set_CmSupportedInputContentTypes(supportedTypes);
StringList unsupportedTypes = Factory.StringList.createList();
unsupportedTypes.add("audio/wma");
unsupportedTypes.add("audio/alac");
cca.set_CmUnsupportedInputContentTypes(unSupportedTypes);

// Set initial values for configuration properties
cca.getProperties().putValue("TranscriptionServiceURI", initialURI);
cca.getProperties().putValue("TranscriptionServiceCredentials", initialCredentials);

// Save the instance of the action on the server
cca.save(RefreshMode.REFRESH);
```

4. Creating a subclass and instance of ContentConversionSettings

```
// Create subclass of RmsTranscriptionSettings
ClassDefinition cdCCA = Factory.ClassDefinition.fetchInstance(objectStore, ”RmsTrancriptionSettings”, null);
String className = "MyTranscriptionSettings”;
Id classId = Id.createId();
ClassDefinition settingsClass = superClass.createSubclass(classId);
LocalizedStringList lsl = Factory.LocalizedString.createList();
lsl.add(getLocalizedStringEntry(className));
settingsClass.set_DisplayNames(lsl);
settingsClass.getProperties().putValue(PropertyNames.SYMBOLIC_NAME, className);
settingsClass.save(RefreshMode.REFRESH);

// Add a custom property to the subclass
PropertyTemplateString ptStringURI = createPropertyTemplateString("TranscriptionQuality", Cardinality.SINGLE, new Integer(8), null, false);
addPropertyDefinitionString(settingsClass, ptStringURI, null, null);
settingsClass.save(RefreshMode.REFRESH);

// Create an instance of the subclass
CmContentConversionSettings ccs = Factory.CmContentConversionSettings.createInstance(objectStore, settingsClass);
ccs.set_DisplayName(“HighQualityTranscriptionSettings”);
ccs.getProperties().putValue("TranscriptionQuality”, “HIGH”);
cca.save(RefreshMode.REFRESH);

// Note: this instance (and others) can be selected when creating subscriptions for transcription
```

Demo

- Look at the test transcription handler and installer.
- Run the installer and verify it installed with ACCE.

Note: The test transcription handler was implemented for internal test purposes and is not available externally.

Course Roadmap

- Overview
- Implementing the Transcription Handler
    - Interfaces
    - Requirements
- Creating an Installer for the Transcription Handler
- Course Summary

Course Summary

What you've learned:

- Understand the overall design of a transcription handler.
- Understand the interfaces used in building a transcription handler.
- Install and configuration for the handler

Where to go for more info:

- For transcription specifics: TechDoc on the Transcription technology preview
  http://www.ibm.com/support/docview.wss?uid=swg27043790
- For everything else: InfoCenter for FileNet P8 v5.2.1

Thanks!

Contacts:

- Product Marketing Manager: Stephen Hussey
- Architect: Michael Seaman
- Development Manager: Grace Smith
- Subject Matter Experts (the development team):
  Shawn Waters, Antonio Montanana, Himanshu Shah, Darren Kennedy, Brian Owings