1 Table of Contents

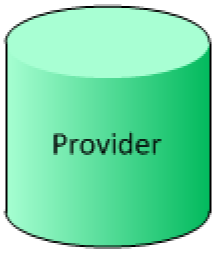
2 About Content Transformation Services

ECMG Content Transformation Services (CTS) is an easy-to-use .NET framework and application suite for document and content processing. CTS connects to many of the most popular Enterprise Content Management (ECM) repositories. CTS enables important tasks such as retrieve, export, import, migrate, checkout, checkin, cancel checkout, delete as well as many other vital operations.

CTS also has a unique ability to abstract document metadata and taxonomies.

3 Concepts

3.1 Provider



A Provider is to CTS what an ODBC, JDBC or OleDB driver is to a database. Providers are the actual connection point from CTS to a content repository. Instances of providers are configured into content sources.

A provider can support one or more interfaces. For more on interfaces please read the topic **provider interfaces (Section 3.1.1)**. Each provider has a unique set of values used to configure it for use in the repository. These values can be combined into a connection string.

3.1.1 Provider Interfaces

Providers support one of more CTS provider interfaces. The interface is a contract between CTS and the individual provider dll. Each interface provides a discrete capability.

Some of the available interfaces for CTS are…

- Exporter
- Importer
- Explorer
- Classification
- BasicContentServices
- UpdateProperties

The **Exporter** interface provides the capability to export content from a repository and translates the metadata into a cdf file.

The **Importer** interface provides the capability to import content into a repository from a cdf file.

The **Explorer** interface provides the information necessary for the ECMG Explorer application to browse and display information for a repository.

The **Classification** interface provides the capability to gather classification information from a repository. This is very helpful for creating repository profiles. The ECMG Explorer capability to create repository information files (*.rif) is directly built on this interface.

The **BasicContentServices** interface provides the capability to execute standardized operations that would be used by a typical end user ECM client application such as Checkin, Checkout etc.

The **UpdateProperties** interface provides the capabilities to perform property updates on repository documents.

3.1.2 Provider Properties

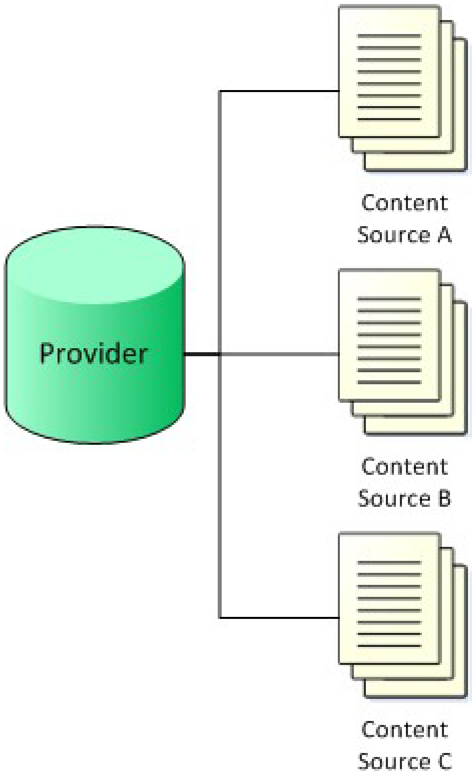
Provider Properties are the information used to configure a content source. The set of properties vary from provider to provider but generally speaking they can be placed into two categories, connection information and path information. The connection information is the information needed to connect to a particular type of repository. The path information is the information needed to perform the export and import operations.

Examples of connection properties are ServerName, ObjectStore, port number, UserName, & Password. Examples of path properties are ExportPath and ImportPath.

Some provider properties are required and some are not. When configuring a content source in ECMG Explorer the property configuration form indicated which properties are required and which are not.

3.2 Content Source

A content source is a configured instance of a **provider (Section 3.1)**.



Let's say that the provider shown above is for IBM/FileNet P8. Content Source A is a configured connection to repository A on Server A. Content Source B is a configured connection to repository B on Server A. Content Source C is a configured connection to repository C on Server B. All of the Content Sources above are pointing to P8 repositories, but each of them is a different instance with different configuration information.

The types of information you can configure in a content source are known as the **provider properties (Section 3.1.2)**. They determine where to connect to and who to connect as.

3.3 Process Overview

Processes in CTS are used to perform various content related tasks. The tasks may be as simple as exporting content from a repository to as complex as migrating content and creating renditions. A process is made of one or more operations. By chaining together different operations CTS can do a complete a wide variety of tasks. Since the software can't anticipate the specific needs of every project processes provide a way to configure CTS to do much more than it could otherwise. The process concept is not unique to CTS, it is a time tested methodology for many other software products that have some type of workflow. CTS borrows the process concept and makes it available for use with several CTS applications.

As you will see, there are many operations available for use. Each new release of CTS will likely contain additional new operations as well. In addition, if there is not an operation that covers a particular task as part of the standard CTS installation, new operations may be created by customers as operation extensions using the CTS extension APIs.

Each operation can be expressed as a node in the Process file. The process file is an XML file that contains all the operations that the process is to perform.

Operations will most typically be placed in the main Operations collection of a process as see in the example below.

Simple Process

```
<Process Name="Migration Process" Description="" LogResult="True" locale="en-US"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <Operations>
  <ExportOperation Name="Export" Description="" LogResult="True" Scope="Source">
   <Parameters>
    <Parameter Name="GenerateCDF" Type="ecmBoolean" Value="False" />
    <Parameter Name="GetPermissions" Type="ecmBoolean" Value="True" />
   </Parameters>
   <RunBeforeBegin />
   <RunAfterComplete />
   <RunOnFailure />
  </ExportOperation>
  <ImportOperation Name="Import" Description="" LogResult="True" Scope="Destination">
   <Parameters>
    <Parameter Name="DeletePropertiesWithoutValues" Type="ecmBoolean" Value="True" />
    <Parameter Name="DocumentFilingMode" Type="ecmString" Value="UnFiled" />
    <Parameter Name="BasePathLocation" Type="ecmString" Value="Front" />
    <Parameter Name="FolderDelimiter" Type="ecmString" Value="/" />
    <Parameter Name="LeadingFolderDelimiter" Type="ecmBoolean" Value="True" />
    <Parameter Name="EnforceClassificationCompliance" Type="ecmBoolean" Value="True" />
    <Parameter Name="SetPermissions" Type="ecmBoolean" Value="True" />
   </Parameters>
   <RunBeforeBegin />
   <RunAfterComplete />
   <RunOnFailure />
  </ImportOperation>
 </Operations>
 <RunBeforeBegin />
 <RunAfterComplete />
 <RunOnFailure />
 <RunBeforeParentBegin />
 <RunAfterParentComplete />
 <RunOnParentFailure />
```

```
</Process>
```

Process Sample

```
<?xml version="1.0" encoding="utf-16"?>
<Process Name="Find and Process Files Process" Description="" LogResult="True"
locale="en-US" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Operations />
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
  <RunBeforeParentBegin />
  <RunAfterParentComplete />
  <RunOnParentFailure />
  <RunBeforeJobBegin>
    <UpdateJobFromFileSearchOperation Name="UpdateJobFromFileSearch" Description="Name"
LogResult="True" Scope="Source">
    <Parameters>
    <Parameter Name="JobName" Type="ecmString" Value="Load Check Summaries" />
    <Parameter Name="SearchRoot" Type="ecmString" Value="C:\temp" />
    <Parameter Name="SearchPattern" Type="ecmString" Value="boaa*.zip" />
    </Parameters>
    <RunBeforeBegin />
    <RunAfterComplete>
      <RunJobOperation Name="RunJob" Description="Runs the job specified in the
'JobName' parameter." LogResult="True" Scope="Source">
      <Parameters>
        <Parameter Name="JobName" Type="ecmString" Value="Load Check Summaries" />
      </Parameters>
      <RunBeforeBegin />
      <RunAfterComplete />
      <RunOnFailure />
      </RunJobOperation>
    </RunAfterComplete>
    <RunOnFailure />
    </UpdateJobFromFileSearchOperation>
  </RunBeforeJobBegin>
</Process>
```

3.3.1 Operations

3.3.1.1 Action Operations

Action operations perform some type of action. While this may sound obvious, it is an important distinction since there are other types of actions as well, such as decision actions.

3.3.1.1.1 Cancel Checkout

The cancel checkout operation cancels the checkout of a document from the source repository.

CancelCheckoutOperation

```
<CancelCheckoutOperation Name="CancelCheckout" Description="" LogResult="True"
Scope="Source">
  <Parameters>
    <Parameter Name="FailIfNotCheckedOut" Type="ecmBoolean" Value="False" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</CancelCheckoutOperation>
```

3.3.1.1.2 Checkin Operation

The checkin operation checks a document into the source or destination repository as determined by the *Scope* attribute..

CheckinOperation

```
<CheckinOperation Name="Checkin" Description="" LogResult="True" Scope="Source">
  <Parameters>
    <Parameter Name="CheckinAsMajor" Type="ecmBoolean" Value="False" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</CheckinOperation>
```

3.3.1.1.3 Check In From Job Operation

The Check In From Job Operation is used in cases where you need to check a document into a repository based on a previous Checkout operation that was performed using a diffrent job. The JobName parameter specifies the originating Checkout job. The originating Checkout job has to be in the same project as the job using the Check In From Job operation. The CheckinAsMajor flag determines whether or not the document should be checked back in as a major version.

This operation will determine the original id of the document in the source repository from the originating Checkout job.

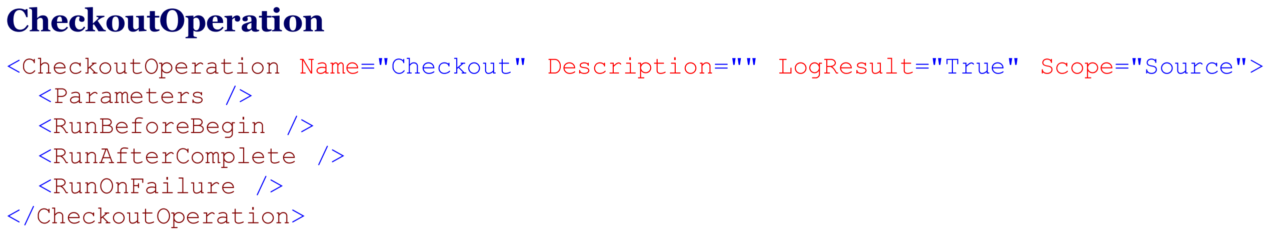
If the originating job does not have a corresponding work item for the item you are trying to check in it will fail.

CheckinFromJobOperation

```
<CheckinFromJobOperation Name="CheckinFromJob" Description="Name" LogResult="True"
Scope="Source">
 <Parameters>
  <Parameter Name="JobName" Type="ecmString" Value="CheckoutTest" />
  <Parameter Name="CheckinAsMajor" Type="ecmBoolean" Value="True" />
 </Parameters>
 <RunBeforeBegin />
 <RunAfterComplete />
 <RunOnFailure />
</CheckinFromJobOperation>
```

3.3.1.1.4 Checkout Operation

The checkout operation checks a document out of the repository.



CheckoutOperation

```
<CheckoutOperation Name="Checkout" Description="" LogResult="True" Scope="Source">
  <Parameters />
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</CheckoutOperation>
```

3.3.1.1.5 Delete Operation

The delete operation deletes a document from the repository.

DeleteOperation

```
<DeleteOperation Name="Delete" Description="" LogResult="True" Scope="Source">
  <Parameters />
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</DeleteOperation>
```

3.3.1.1.6 Export Operation

The export operation exports a document from a source repository. The exported document may optionally be written out to disk. If the exported document will be imported into a source repository in the same process it is not necessary to save it to disk.

ExportOperation

```
<ExportOperation Name="Export" Description="" LogResult="True" Scope="Source">
  <Parameters>
    <Parameter Name="GenerateCDF" Type="ecmBoolean" Value="True" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</ExportOperation>
```

3.3.1.1.7 Import Operation

The import operation imports a document into the repository.

ImportOperation

```
<ImportOperation Name="Import" Description="" LogResult="True" Scope="Destination">
  <Parameters>
    <Parameter Name="DeletePropertiesWithoutValues" Type="ecmBoolean" Value="True" />
    <Parameter Name="DocumentFilingMode" Type="ecmString" Value="UnFiled" />
    <Parameter Name="BasePathLocation" Type="ecmString" Value="Front" />
    <Parameter Name="FolderDelimiter" Type="ecmString" Value="/" />
    <Parameter Name="LeadingFolderDelimiter" Type="ecmBoolean" Value="True" />
    <Parameter Name="EnforceClassificationCompliance" Type="ecmBoolean" Value="True" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</ImportOperation>
```

3.3.1.1.8 Migrate Operation

The migrate operation is actually a process composed by default of three separate operations, (export, transform and import).

Migration Process

```xml
<?xml version="1.0" encoding="utf-16"?>
<Process Name="Migration Process" Description="" LogResult="True" locale="en-US"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Operations>
    <ExportOperation Name="Export" Description="" LogResult="True" Scope="Source">
      <Parameters>
        <Parameter Name="GenerateCDF" Type="ecmBoolean" Value="False" />
      </Parameters>
      <RunBeforeBegin />
      <RunAfterComplete />
      <RunOnFailure />
    </ExportOperation>
    <TransformOperation Name="Transform" Description="" LogResult="True"
Scope="Source">
      <Parameters>
        <Parameter Name="RootTransformation" Type="ecmString" Value="Import from File
System to Base Content Engine Web Services 3.5 Provider" />
      </Parameters>
      <RunBeforeBegin />
      <RunAfterComplete />
      <RunOnFailure />
    </TransformOperation>
    <ImportOperation Name="Import" Description="" LogResult="True" Scope="Destination">
      <Parameters>
        <Parameter Name="DeletePropertiesWithoutValues" Type="ecmBoolean" Value="True"
/>
        <Parameter Name="DocumentFilingMode" Type="ecmString" Value="UnFiled" />
        <Parameter Name="BasePathLocation" Type="ecmString" Value="Front" />
        <Parameter Name="FolderDelimiter" Type="ecmString" Value="/" />
        <Parameter Name="LeadingFolderDelimiter" Type="ecmBoolean" Value="True" />
        <Parameter Name="EnforceClassificationCompliance" Type="ecmBoolean"
Value="True" />
      </Parameters>
      <RunBeforeBegin />
      <RunAfterComplete />
      <RunOnFailure />
    </ImportOperation>
  </Operations>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
  <RunBeforeParentBegin />
  <RunAfterParentComplete />
  <RunOnParentFailure />
</Process>
```

3.3.1.1.9 Read Document Operation

The read document operation opens a document (*cpf* or *cdf*) file from a local or network drive. The primary usage of this operation is to initialize the document object used during a process for subsequent operations. The Scope parameter determines whether or not the document is read from the SourceDocId or the DestinationDocId.

Note that this operation is only applicable for cases where the relevant DocId points to a *cpf* or *cdf* file.

ReadDocumentOperation

```
<ReadDocumentOperation Name="ReadDocument" Description="" LogResult="True"
Scope="Source">
 <Parameters>
  <Parameter Name="Scope" Type="ecmString" Value="Source" />
 </Parameters>
 <RunBeforeBegin />
 <RunAfterComplete />
 <RunOnFailure />
</ReadDocumentOperation>
```

3.3.1.1.10 Replace Operation

ReplaceOperation

```
<ReplaceOperation Name="Replace" Description="" LogResult="True" Scope="Source">
  <Parameters />
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</ReplaceOperation>
```

3.3.1.1.11 RunJob Operation

The run job operation runs another job in the same project.

RunJobOperation

```xml
<RunJobOperation Name="RunJob" Description="Runs the job specified in the 'JobName'
parameter." LogResult="True" Scope="Source">
  <Parameters>
    <Parameter Name="JobName" Type="ecmString" Value="" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</RunJobOperation>
```

3.3.1.1.12 SaveToFile Operation

The save to file operation saves the current document to a file in the destination folder.

SaveToFileOperation

```
<SaveToFileOperation Name="SaveToFile" Description="" LogResult="True" Scope="Source">
  <Parameters>
    <Parameter Name="DestinationFolder" Type="ecmString" Value="%CtsDocsPath%\Exports"
/>
    <Parameter Name="SaveMode" Type="ecmString" Value="Archive" />
    <Parameter Name="CreateItemFolders" Type="ecmBoolean" Value="False" />
    <Parameter Name="ArchivePassword" Type="ecmString" Value="" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</SaveToFileOperation>
```

3.3.1.1.13 Transform Operation

The transform operation transforms the current document using the specified transformation.

TransformOperation

```
<TransformOperation Name="Transform" Description="" LogResult="True" Scope="Source">
  <Parameters>
    <Parameter Name="RootTransformation" Type="ecmString" Value="" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</TransformOperation>
```

3.3.1.1.14 Unfile Operation

The unfile operation unfiles the current document from all folders.

UnfileOperation

```
<UnfileOperation Name="Unfile" Description="" LogResult="True" Scope="Source">
  <Parameters />
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</UnfileOperation>
```

3.3.1.1.15 Update Operation

The update operation updates the contents of the current document.

UpdateOperation

```
<UpdateOperation Name="Update" Description="" LogResult="True" Scope="Source">
  <Parameters />
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</UpdateOperation>
```

3.3.1.1.16 UpdateJobFromFileSearch Operation

The update job from file search operation updates the target job with the results of a file search. If the found files are already added to the target job they are ignored.

UpdateJobFromFileSearchOperation

```
<UpdateJobFromFileSearchOperation Name="UpdateJobFromFileSearch" Description="Name"
LogResult="True" Scope="Source">
  <Parameters>
    <Parameter Name="JobName" Type="ecmString" Value="Job Name" />
    <Parameter Name="SearchRoot" Type="ecmString" Value="C:\Temp" />
    <Parameter Name="SearchPattern" Type="ecmString" Value="*.*" />
    <Parameter Name="TopDirectoryOnly" Type="ecmBoolean" Value="True" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</UpdateJobFromFileSearchOperation>
```

3.3.1.1.17 UpdatePermissions Operation

The update permissions operation updates the permissions on the current document.

⚠ This operation is only valid for providers that implement the `IUpdatePermissions` interface.

UpdatePermissionsOperation

```
<UpdatePermissionsOperation Name="UpdatePermissions" Description="" LogResult="True"
Scope="Source">
  <Parameters>
    <ObjectParameter Name="SecurityArguments" Type="ecmObject">
      <Value>
        <DocumentSecurityArgs>
          <ObjectID />
          <Permissions>
            <Permission>
              <PrincipalName>BIAdmin</PrincipalName>
              <Access xsi:type="AccessLevel">
                <ParentName>AccessRight</ParentName>
                <Name>FullControl</Name>
                <Value xsi:nil="true" />
                <PermissionList />
                <Level>FullControl</Level>
                <Rights />
              </Access>
              <AccessType>Allow</AccessType>
              <Source>Direct</Source>
              <PrincipalType>Group</PrincipalType>
            </Permission>
          </Permissions>
          <Mode>Append</Mode>
        </DocumentSecurityArgs>
      </Value>
    </ObjectParameter>
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
</UpdatePermissionsOperation>
```

3.3.1.2 Decision Operations

Decision operations allow a process to perform different operations depending on the result of an evaluation. Based on the result of the evaluation, a set of other operations may or may not be performed. All decision operations have two operation sub collections, TrueOperations and FalseOperations. If the evaluation results in a true result, the operations listed in the TrueOperations collection will be executed. Otherwise the operations in the FalseOperations collection will be executed. Note that it is not necessary to have operations in both collections.

3.3.1.2.1 ContentExtensionDecision Operation

ContentExtensionDecisionOperation

```
<ContentExtensionDecisionOperation Name="ContentExtensionDecision" Description=""
LogResult="True" Scope="Source">
  <Parameters>
    <Parameter Name="VersionScope" Type="ecmLong" Value="0" />
    <Parameter Name="ContentElementIndex" Type="ecmLong" Value="0" />
    <Parameter Name="Mode" Type="ecmString" Value="Valid" />
    <Parameter Name="Extensions" Type="ecmString" Cardinality="ecmMultiValued">
    <Values />
    </Parameter>
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
  <TrueOperations />
  <FalseOperations />
</ContentExtensionDecisionOperation>
```

3.3.1.2.2 ContentSizeDecision Operation

ContentSizeDecisionOperation

```xml
<ContentSizeDecisionOperation Name="ContentSizeDecision" Description=""
LogResult="True" Scope="Source">
  <Parameters>
    <Parameter Name="VersionScope" Type="ecmLong" Value="0" />
    <Parameter Name="ContentElementIndex" Type="ecmLong" Value="0" />
    <Parameter Name="MinimumContentSizeKB" Type="ecmLong" Value="0" />
    <Parameter Name="MaximumContentSizeKB" Type="ecmLong" Value="102400" />
  </Parameters>
  <RunBeforeBegin />
  <RunAfterComplete />
  <RunOnFailure />
  <TrueOperations />
  <FalseOperations />
</ContentSizeDecisionOperation>
```

3.3.1.3 Operation Parameters

Each operation may have one or more parameters. Operation parameters provide additional information needed for the operation to perform it's job. For example, the import operation has a parameter that specifies whether or not to suppress sending properties without values to the destination repository.

The whole idea of having a set of predefined operations is to be able to configure a process to do many different things. As you can imagine, for each kind of operation you want to do, there are different kinds of instructions that are needed. That is where the parameters come in. Without parameters, operations just would not be practical.

3.3.2 Extensions

3.3.2.1 Operation Extensions

Operation Extensions are used to create new operations that are not part of the core operations that are shipped with CTS. This enables customers and partners to create new operations specific to their needs.

All operation extensions must be registered in the extension catalog before they will be registered by CTS.

3.3.2.2 Extension Catalog

TODO: Write the content for this topic.

3.3.3 Process Events

Processes and operations have defined events that are fired at various times during the execution of the process. You can define your processes to execution operations at specific points during the process. The process events are listed below.

- **RunBeforeBegin (Section 3.3.3.1)**
- **RunAfterComplete (Section 3.3.3.2)**
- **RunOnFailure (Section 3.3.3.3)**
- **RunBeforeParentBegin (Section 3.3.3.4)**
- **RunAfterParentComplete (Section 3.3.3.5)**
- **RunOnParentFailure (Section 3.3.3.6)**
- **RunBeforeJobBegin (Section 3.3.3.7)**
- **RunAfterJobComplete (Section 3.3.3.8)**
- **RunOnJobFailure (Section 3.3.3.9)**

Operations have a subset of these operations that are fired during the execution of the operation. The operation events are listed below.

- **RunBeforeBegin (Section 3.3.3.1)**
- **RunAfterComplete (Section 3.3.3.2)**
- **RunOnFailure (Section 3.3.3.3)**

The example process below uses the RunBeforeJobBegin event to execute an UpdateJobFromFileSearchOperation

RunBeforeJobBeginSample

```
<?xml version="1.0" encoding="utf-16"?>
<Process Name="Get Latest Check Summaries" Description="" LogResult="True" locale="en-
US" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <Operations />
 <RunBeforeBegin />
 <RunAfterComplete />
 <RunOnFailure />
 <RunBeforeParentBegin />
 <RunAfterParentComplete />
 <RunOnParentFailure />
 <RunBeforeJobBegin>
  <UpdateJobFromFileSearchOperation Name="UpdateJobFromFileSearch" Description="Name"
LogResult="True" Scope="Source" >
   <Parameters>
    <Parameter Name="JobName" Type="ecmString" Value="2. Read Check Summaries" />
    <Parameter Name="SearchRoot" Type="ecmString"
Value="C:\Users\Public\Documents\Incoming" />
    <Parameter Name="SearchPattern" Type="ecmString" Value="boaa*.zip" />
    <Parameter Name="TopDirectoryOnly" Type="ecmBoolean" Value="True" />
   </Parameters>
   <RunBeforeBegin />
   <RunAfterComplete>
    <RunJobOperation Name="RunJob" Description="Runs the job specified in the 'JobName'
```

```
parameter." LogResult="True" Scope="Source">
     <Parameters>
      <Parameter Name="JobName" Type="ecmString" Value="2. Read Check Summaries" />
     </Parameters>
     <RunBeforeBegin />
     <RunAfterComplete />
     <RunOnFailure />
     </RunJobOperation>
    </RunAfterComplete>
   <RunOnFailure />
  </UpdateJobFromFileSearchOperation>
 </RunBeforeJobBegin>
</Process>
```

3.3.3.1 Run Before Begin

The RunBeforeBegin event is triggered before the respective process or operation is begun. You can add one or more operations to this event to execute before beginning.

3.3.3.2 Run After Complete

The RunAfterComplete event is triggered after the respective process has completed all the main operations or the operation itself has completed.   You can add one or more operations to this event to execute after completion.

3.3.3.3 Run On Failure

The RunOnFailure event is triggered when the respective process or operation has failed. You can add one or more operations to this event to execute on failure.

3.3.3.4 Run Before Parent Begin

The RunBeforeParentBegin event is fired right before a new batch starts to execute it's first item. You can add operations to this event that you want to run before a new batch is started. Note that this event is fired before each time you execute a batch, not just the first time you execute it.

3.3.3.5 Run After Parent Complete

The RunAfterParentComplete event is fired after a new batch completes the execution of it's last item. You can add operations to this event that you want to run when a batch has completed processing.

3.3.3.6 Run On Parent Failure

The RunOnParentFailure event is fired when a batch completes the with at least one item having failed. You can add operations to this event that you want to run when a batch has completed processing with failed items.

3.3.3.7 Run Before Job Begin

The RunBeforeParentBegin event is fired right before a job starts to execute an item. You can add operations to this event that you want to run before a job is started. Note that this event is fired before each time you execute a job, not just the first time you execute it. An example of using this event is shown below.

The example process below uses the RunBeforeJobBegin event to execute an UpdateJobFromFileSearchOperation

RunBeforeJobBeginSample

```
<?xml version="1.0" encoding="utf-16"?>
<Process Name="Get Latest Check Summaries" Description="" LogResult="True" locale="en-
US" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <Operations />
 <RunBeforeBegin />
 <RunAfterComplete />
 <RunOnFailure />
 <RunBeforeParentBegin />
 <RunAfterParentComplete />
 <RunOnParentFailure />
 <RunBeforeJobBegin>
  <UpdateJobFromFileSearchOperation Name="UpdateJobFromFileSearch" Description="Name"
LogResult="True" Scope="Source" >
   <Parameters>
    <Parameter Name="JobName" Type="ecmString" Value="2. Read Check Summaries" />
    <Parameter Name="SearchRoot" Type="ecmString"
Value="C:\Users\Public\Documents\Incoming" />
    <Parameter Name="SearchPattern" Type="ecmString" Value="boaa*.zip" />
    <Parameter Name="TopDirectoryOnly" Type="ecmBoolean" Value="True" />
   </Parameters>
   <RunBeforeBegin />
   <RunAfterComplete>
    <RunJobOperation Name="RunJob" Description="Runs the job specified in the 'JobName'
parameter." LogResult="True" Scope="Source">
     <Parameters>
      <Parameter Name="JobName" Type="ecmString" Value="2. Read Check Summaries" />
     </Parameters>
     <RunBeforeBegin />
     <RunAfterComplete />
     <RunOnFailure />
     </RunJobOperation>
    </RunAfterComplete>
   <RunOnFailure />
  </UpdateJobFromFileSearchOperation>
 </RunBeforeJobBegin>
</Process>
```

3.3.3.8 Run After Job Complete

The RunAfterJobComplete event is fired when a job completes it's last unprocessed item in the last batch. You can add operations to this event that you want to run when a job is completed.

3.3.3.9 Run On Job Failure

The RunOnJobFailure event is fired when a job completes the with at least one item having failed. You can add operations to this event that you want to run when a job has completed processing with failed items.

3.4 Transformation Overview

Content Transformation Files define the project specific business logic for translating repository meta data from a source repository to a destination repository. The logic is encapsulated in an xml based file hereafter referred to as a content transformation file (or ctf file).

A transformation specifies actions to be performed on a document in transition between a source and destination repository. These actions can be categorized in simple terms by their type. There are actions to create a new property, rename an existing property, delete an existing property, change a property value and to change the filename of a document content element.

A content transformation file is simply a serialized instance of a transformation object. When a transformation file path is specified in the configuration of a CTS based application, the configuration is merely pointing to a location where the transformation file can be found. When the application is executed the transformation is deserialized from xml and all of the transformation actions specified in the file become instructions in memory.

For detailed information on the structure and syntax of a content transformation file refer to the content transformation file topic.

A content transformation file (ctf) defines all of the transformation actions to be performed on the document. These actions are performed on the abstract document definition contained within the content definition file (cdf).

A transformation is a defined set of transformation actions. The most common types of transformation actions are defined below.

Action	Description
Create Property (Section 3.4.1.5)	Defines and creates a new property in the document.
Rename Property (Section 3.4.1.16)	Renames an existing property in the document.
Delete Property (Section 3.4.1.12)	Removes a property from the document.
Change Property Value (Section 3.4.1.6)	Changes the value of an existing property in the document.
Change Property Cardinality (Section 3.4.1.7)	Changes the cardinality of an existing property.
Change Content Retrieval Name (Section 3.4.1.8)	Changes the file name of a content element in the document.
Change Content Mime Type (Section 3.4.1.9)	Changes the mime type of a content element in the document.

3.4.1 Transformation Actions

3.4.1.1 Add Literal Folder Path

The AddLiteralFolderPath action adds a folder path to the current document.

AddLiteralFolderPath

```
<Action xsi:type="AddLiteralFolderPath" Name="" Description="" >
  <FolderPath>/Test Folder</FolderPath>
</Action>
```

3.4.1.2 Add Property Value

The AddPropertyValue action adds an additional value to a multi-valued property.

The are four basic ways to utilize the AddPropertyValue action. These are listed below.

- **Literal Value Substitution (Section 3.4.1.6.1)**
    - Apply a simple hard coded value to the property.
- **Data Parser (Section 3.4.1.8)**
    - Resolve the new value from some variation of an existing value of another property of the document.
- **Data Map (Section 3.4.1.6.3.2)**
    - Resolve the new value from an external data source, such as a database. The value can be filtered based on the value of one or more properties in the document.
- **Data List**
    - Resolves the new value from a simple list.

The structure and syntax of the AddPropertyValue part is identical to the structure and syntax of the **ChangePropertyValue (Section 3.4.1.6)** part. The primary difference is in the type attribute of the Action node. For AddPropertyValue it is ALWAYS AddPropertyValue . For ChangePropertyValue it is ALWAYS ChangePropertyValue.

The reason that the structure and behavior is so similar to the ChangePropertyValue part is that the AddPropertyValue transformation action is actually sub-classed from the ChangePropertyValue transformation action.

The key sections of the AddPropertyValue are listed below.

Section	Description
PropertyName	For ChangeContentRetrievalName the value should ALWAYS be ContentPath.
PropertyScope	For ChangeContentRetrievalName the value should ALWAYS be VersionProperty.
VersionIndex	The zero based index for the version to apply to. For all versions use the value -1.
SourceType	This will either be **Literal (Section 3.4.1.6.1)** or **DataParser (Section 3.4.1.8)**.

Usage:

The usage shown below is for the **(Section 3.4.1.8)Data Parser (Section 3.4.1.8)(Section 3.4.1.8)**.

AddPropertyValue

```
<Action xsi:type="AddPropertyValue" Name="Add TagNumber" Description="Add Tag Number
from lookup">
  <PropertyName>TagNumber</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <AllowDuplicates>false</AllowDuplicates>
  <DataLookup xsi:type="DataParser">
  <SourceProperty AutoCreate="false" Persistent="false">
    <PropertyName>EquipmentNumber</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
    <ValueIndex>First</ValueIndex>
  </SourceProperty>
  <DestinationProperty AutoCreate="false" Persistent="false">
    <PropertyName>TagNumber</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
    <ValueIndex>First</ValueIndex>
  </DestinationProperty>
  <Part>COMPLETE</Part>
  </DataLookup>
</Action>
```

3.4.1.3 Clear All Folder Paths

The ClearAllFolderPaths action clears all the folder paths for a document.

ClearAllFolderPaths

```
<Action xsi:type="ClearAllFolderPaths" />
```

3.4.1.4 Clear All Property Values

The ClearAllPropertyValues action clears all the values of an existing property, whether it is a single valued or a multivalued property.

The key sections of the ClearAllPropertyValues action are listed below.

Section	Description
PropertyName	The name of the property to change.
PropertyScope	The scope of the property to change. Possible values are **VersionProperty** and **DocumentProperty**.
VersionIndex	Specifies the specific version of the document to apply the change to. The index is zero based, meaning the first version is specified with a value of zero. The value -1 may also be specified to affect all versions.

ClearAllPropertyValues

```
<Action xsi:type="ClearAllPropertyValues" Name="" Description="">
  <PropertyName>TagNumber</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>-1</VersionIndex>
</Action>
```

3.4.1.5 Create Property Action

The CreatePropertyAction defines and creates a new property in the document.

The key sections of the CreatePropertyAction are listed below.

Section	**Description**
Name	The name of the action
Persistent	Specifies whether or not the property should be deleted when the transformation completes. Possible values are *true* and *false*. If set to *false* the property will be deleted automatically when the transformation is complete.
PropertyName	The name of the new property to be created.
PropertyScope	Specifies the scope of the new property. Possible values are **VersionProperty** and **DocumentProperty**.
PropertyValue	Optional, if supplied, specifies the value to be applied to the new property.
PropertyType	The data type to be stored in the new property. Corresponds to the data type defined for the property in the underlying repository. Possible values are **ecmBinary**, **ecmBoolean**, **ecmDate**, **ecmDouble**, **ecmGuid**, **ecmLong**, **ecmObject** and **ecmString**.
FirstVersionOnly	Deprecated, Please use VersionScope. Specifies whether or not the value is to be applied to the first version only or all versions of the document. This is important for properties which are only settable on document creation and are read only thereafter.
VersionScope	Specifies which versions of the document the property is to be added to. Possible values are **AllVersions**, **FirstVersionOnly** and **LastVersionOnly**.

CreatePropertyAction

```
<Action xsi:type="CreatePropertyAction" Name="CreateCreateLifecycle"
Description="CreateCreateLifecycle" Persistent="true">
  <PropertyName>CreateLifecycle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <PropertyValue xsi:type="xsd:string">Draft</PropertyValue>
  <PropertyType>ecmString</PropertyType>
  <Cardinality>ecmSingleValued</Cardinality>
  <FirstVersionOnly>false</FirstVersionOnly>
  <VersionScope>AllVersions</VersionScope>
</Action>
```

3.4.1.6 Change Property Value

The ChangePropertyValue changes the value of an existing property in the document.

The are four basic ways to utilize the ChangePropertyValue action. These are listed below.

- **Literal Value Substitution (Section 3.4.1.6.1)**
  - Apply a simple hard coded value to the property.
- **Data Parser (Section 3.4.1.6.3)**
  - Resolve the new value from some variation of an existing value of another property of the document.
- **Data Map (Section 3.4.1.6.3.2)**
  - Resolve the new value from an external data source, such as a database. The value can be filtered based on the value of one or more properties in the document.
- **Data List**
  - Resolves the new value from a simple list.

3.4.1.6.1 Literal Value Substitution

The Literal Value Substitution sets a value to a hard coded literal value.

The key sections are listed below.

Section	Description
PropertyName	The name of the property in which the value is to be changed.
PropertyScope	Specifies the scope of the property. Possible values are **VersionProperty** and **DocumentProperty**.
PropertyValue	Specifies the value to be applied to the new property.
VersionIndex	The zero based index for the version to apply to. For all versions use the value -1.
SourceType	Specifies whether or not the value is to be applied to the first version only or all versions of the document. This is important for properties which are only settable on document creation and are read only thereafter.Specifies the general manner in which the source value is resolved. Possible values are **Literal** and **DataLookup**. *Note that for a* **Literal Value Substitution** *the only valid value in this field is* **Literal**.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue" Name="Set FileNetClass to Document"
Description="">
  <PropertyName>FileNetClass</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <PropertyValue xsi:type="xsd:string">Document</PropertyValue>
  <VersionIndex>-1</VersionIndex>
  <SourceType>Literal</SourceType>
</Action>
```

3.4.1.6.2 Dynamic Dates and Times

Dates and times may be dynamically set when the current date and or time is needed in a transformation. There are a series of keywords that can be used.

The are four basic ways to utilize the ChangePropertyValue action. These are listed below.

- **%now%**
    - Returns a standard date/time value for use with properties defined as date/time.
- **%now.touniversaltime%, %now.toutc%, %now.utc%**
    - Returns a date/time value converted to universal time for use with properties defined as date/time.
- **%now.date%**
    - Returns a standard date only value for use with properties defined as date/time.
- **%today%, %now.today%**
    - Returns a standard date only value for use with properties defined as date/time.
- **%now.touniversaltime.date%, %now.toutc.date%, %now.utc.date%**
    - Returns a date only value converted to universal time for use with properties defined as date/time.
- **%now.day%**
    - Returns the current day as an integer
- **%now.month%**
    - Returns the current month as an integer
- **%now.year%**
    - Returns the current year as an integer
- **%now.hour%**
    - Returns the current hour as an integer
- **%now.minute%**
    - Returns the current minute as an integer
- **%now.second%**
    - Returns the current second as an integer
- **%now.millisecond%**
    - Returns the current millisecond as an integer
- **%now.dayofyear%**
    - Returns the current day of the year as an integer
- **%now.dayofweek%**
    - Returns the current day of the week as an integer where Sunday starts with zero.
- **%now.dayofweek.tostring%**
    - Returns the current day of the week as the English word (i.e. Tuesday)
- **%now.ticks%**
    - Returns the number of ticks for the current date/time as an integer
- **%now.tolongdatestring%**
    - Returns the current date as a long date string using the current culture. In US English the return value would look something like *Wednesday, May 16, 2001*.
- **%now.tolongtimestring%**
    - Returns the current date as a long time string using the current culture. In US English the return value would look something like *3:02:15 AM*.
- **%now.toshortdatestring%**
    - Returns the current date as a shortdate string using the current culture. In US English the return value would look something like *5/16/2001*.

- **%now.toshorttimestring%**
    - Returns the current date as a short time string using the current culture. In US English the return value would look something like *3:02 AM*.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue"
  <PropertyName>ImportDate</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <PropertyValue xsi:type="xsd:string">%now%</PropertyValue>
  <VersionIndex>-1</VersionIndex>
  <SourceType>Literal</SourceType>
</Action>
```

3.4.1.6.3 Data Parser

A Data Parser is used to resolve the new value from another property of the document. It can get the value exactly as is or it can get the result of parsing operations as defined in the **Part (Section 3.4.1.6.3.1)** section.

The key sections of the ChangePropertyValue via Data Map are listed below.

Section	Description
PropertyName	The name of the property whose value is to be changed.
PropertyScope	Specifies the scope of the property to change. Possible values are **VersionProperty** and **DocumentProperty**.
VersionIndex	Specifies the specific version of the document to apply the change to. The index is zero based, meaning the first version is specified with a value of zero. The value -1 may also be specified to affect all versions.
SourceType	Specifies the general manner in which the source value is resolved. Possible values are Literal and **DataLookup**. *Note that for a* **DataParser** *the only valid value in this field is* **DataLookup**.
SourceProperty	Detailed information on the document property to obtain the value from.
DestinationProperty	Detailed information on the document property to apply the value to.
AutoCreate	Specifies whether or not to create the property if it does not exist.
Persistent	Used in conjunction with AutoCreate. If AutoCreate is set to true, Persistent determines whether or not the new property is automatically deleted at the end of the transformation operation. If set to false then the property will be automatically deleted, if set to true then the property will be persisted. This is primarily used for automatically deleting 'temporary' properties used during complex transformation operations.
Part (Section 3.4.1.6.3.1)	Specifies how to parse the source property.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue" Name="Change DocumentTitle using TrueTitle"
Description="">
  <PropertyName>DocumentTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>-1</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <AllowDuplicates>false</AllowDuplicates>
  <DataLookup xsi:type="DataParser">
  <SourceProperty>
    <PropertyName>TrueTitle</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
    <ValueIndex>First</ValueIndex>
  </SourceProperty>
  <DestinationProperty AutoCreate="false" Persistent="false">
    <PropertyName>Title</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
    <ValueIndex>First</ValueIndex>
  </DestinationProperty>
  <Part>COMPLETE</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1 Data Parser Parts

This topic focuses exclusively on the ***Part*** tag in a *DataParser* specific *DataLookup* node. The Part tag is where you can specify exactly how you want the DataParser to parse the information in the source property.

Some part tags only take the name, such as COMPLETE, others have one or more required arguments, such as LEFT. Arguments are always specified with a colon delimiter. For example the LEFT part is always expressed with one argument specifying the number of characters wanted; LEFT:3 means that we want the first three characters. Some part tags require multiple arguments and a few have optional arguments as well. For more details on the syntax of a particular part tag see the topic for that tag as linked below.

The available options for the Part tag are listed below.

Section	Description	Syntax
COMPLETE	Get the source value unchanged from its original state.	COMPLETE
CONCATENATE	Concatenate the source property to the destination property. Use the specified delimiter if provided.	CONCATENATE
CONDITIONAL CONCATENATE	Concatenate the source property to the destination property only if the source property currently has a certain value. Use the specified delimiter if provided.	CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value *Optional* CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value:= *Optional* CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value:NOT EQUALS *Optional* CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value:STARTS WITH *Optional* CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value:ENDS WITH *Optional* CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value:CONTAINS *Optional* CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value:NOT EMPTY *Optional* CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value:DESTINATION EMPTY
CONDITIONAL CONCATENATE LITERAL	Concatenate the literal value to the destination property only if the source property currently has a certain value. Use the specified	CONDITIONAL CONCATENATE:Delmiter:Conditional Source Value:Literal Value

Section	Description	Syntax
	delimiter if provided.	
CONDITIONAL	Get the source property only if the destination property currently has a certain value.	
CONDITIONAL LITERAL	Apply a literal value only if the destination property currently has a certain value.	CONDITIONAL LITERAL:General:Document (
LEFT	Get just the part we want from the left of the value.	LEFT:12 (Get the first 12 characters)
RIGHT	Get just the part we want from the right of the value.	RIGHT:8 (Get the last 8 characters)
MID or SUBSTRING	Get just the part we want from the middle of the value.	
REMOVE LEFT	Remove the number of characters we don't want from the left of the value.	
REMOVE RIGHT	Remove the number of characters we don't want from the right of the value.	
TRIM	Remove the parts we don't want from the left and right of the value.	
TRIM LEFT	Remove the part we don't want from the left of the value.	
TRIM RIGHT	Remove the part we don't want from the right of the value.	
REMOVE AFTER LAST INSTANCE OF CHARACTER	Remove everything after the last instance of the specified character. Good for removing file extensions.	
REPLACE	Replace specified characters with supplied characters	
UPPER	Make all characters upper case. Otherwise it is the same as COMPLETE.	
LOWER	Make all characters lower case. Otherwise it is the same as COMPLETE.	

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue">
  <PropertyName>MclTitle</PropertyName>
<PropertyScope>VersionProperty</PropertyScope>
<VersionIndex>0</VersionIndex>
<SourceType>DataLookup</SourceType>
<DataLookup xsi:type="DataParser">
<SourceProperty>
```

```
    <PropertyName>TrueMclTitle</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
  </SourceProperty>
  <DestinationProperty>
    <PropertyName>MclTitle</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
  </DestinationProperty>
  <Part>COMPLETE</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.1 Complete

The COMPLETE Part takes the value of the source document property exactly as it exists in the current document and copies it into the destination property.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>TrueMclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>COMPLETE</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.2 Concatenate

The CONCATENATE Part takes the value of the source document property in the current document and concatenates it with the destination property. A delimiter may be specified to place between the two values if desired.

A common use of the CONCATENATE Part is to build folder paths from document properties. In fact, one of the most useful aspects of transformation actions is that they can be combined to implement very powerful and complex business rules. The CONCATENATE part may be repeated several times with different source properties and the same destination property to build complex values such as folder paths and document numbers.

The syntax is as follows CONCATENATE:Delimiter

Parameter	Description
Delimiter	Optional. The Delimiter to place between the source value and the original destination value.

The example below adds the value of CompanyCode to the Folder Path with a delimiter of forward slash. In this case if the folder path was originally '/Engineering' and the CompanyCode was IBM the folder path would be '/Engineering/IBM' after applying the transformation action.

ChangePropertyValue

```xml
<Action xsi:type="ChangePropertyValue">
  <PropertyName>Folder  Path</PropertyName>
  <PropertyScope>DocumentProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>CompanyCode</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>Folder  Path</PropertyName>
      <PropertyScope>DocumentProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>CONCATENATE:/</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.3 Conditional

The CONDITIONAL Part gets the value of the source property only if the destination property currently has a certain value and copies it into the destination property.

The syntax is as follows CONDITIONAL:Conditional Source Value:Case Sensitive

Parameter	Description
Literal Value	The new value to set.
Conditional Source Value	The conditional value to evaluate the property against.
Case Sensitive	Optional. Specifies whether the comparison should be case sensitive.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>TrueMclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>CONDITIONAL:Condition:false</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.4 Conditional Concatenate

The CONDITIONAL CONCATENATE Part concatenate the source property to the destination property only if the source property currently has a certain value. It uses the specified delimiter if provided.

The syntax is as follows CONDITIONAL CONCATENATE:Delimiter:Conditional Source Value:Operator

Parameter	Description
Delimiter	Optional. The Delimiter to place between the original value and the concatenated literal
Conditional Source Value	The conditional value to evaluate the property against.
Operator	The manner in which the condition is to be evaluated. Possible values are =, <>, **EQUALS**, **NOT EQUALS**, **STARTS WITH**, **ENDS WITH**, **CONTAINS**, **NOT EMPTY** and **DESTINATION EMPTY**.

The example below evaluates the value of CompanyName. If it has any value at all, it is concatenated to the end of the DocumentNumber property.

If the source value was 'ECMG' and the destination value was 'JOB' then the destination property would have been changed to 'JOB-ECMG'. In the case below the part is set to '*CONDITIONAL CONCATENATE:-::NOT EMPTY*'.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue">
  <PropertyName>DocumentNumber</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>CompanyName</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>DocumentNumber</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>CONDITIONAL CONCATENATE:-::NOT EMPTY</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.5 Conditional Concatenate Literal

The CONDITIONAL CONCATENATE LITERAL Part concatenates the literal value to the destination property only if the source property currently has a certain value. It uses the specified delimiter if provided.

The syntax is as follows CONDITIONAL CONCATENATE LITERAL:Conditional Source Value:Literal Value:Delimiter

Parameter	Description
Literal Value	The new value to set.
Conditional Source Value	The conditional value to evaluate the property against.
Delimiter	Optional. The Delimiter to place between the original value and the concatenated literal

The example below evaluates the value of CompanyName. If the value is 'ECMG' then it is changed to 'ECMG, LLC.'. In the case below the part is set to '*CONDITIONAL CONCATENATE LITERAL:ECMG:, LLC.*' This could have also been accomplished with '*CONDITIONAL CONCATENATE LITERAL:ECMG:LLC.:,* ' since we can conditionally provide a delimiter, which also is a literal.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue">
  <PropertyName>CompanyName</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>CompanyName</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>CompanyName</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>CONDITIONAL CONCATENATE LITERAL:ECMG:, LLC.</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.6 Lower

The LOWER Part takes the value of the source document property, makes all of the characters lower case and copies the result into the destination property. A common use for this is to enforce that the value for a property is all lower case. If the SourceProperty and the DestinationProperty are the same then the result will be that the value of the source property is made lower case.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>LOWER</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.7 Upper

The UPPER Part takes the value of the source document property, makes all of the characters upper case and copies the result into the destination property. A common use for this is to enforce that the value for a property is all upper case. If the SourceProperty and the DestinationProperty are the same then the result will be that the value of the source property is made upper case.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>UPPER</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.8 Left

The LEFT Part takes the specified number of characters from the left of the value of the source document property in the current document and copies it into the destination property.

The example below takes the first seven characters of the value of TrueMclTitle and places them into MclTitle. If the value of TrueMclTitle were 'Content Transformation Services' then the value of MclTitle would become 'Content'.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>LEFT:7</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.9 Right

The RIGHT Part takes the specified number of characters from the right of the value of the source document property in the current document and copies it into the destination property.

The example below takes the first seven characters of the value of TrueMclTitle and places them into MclTitle. If the value of TrueMclTitle were 'Content Transformation Services' then the value of MclTitle would become 'Services'.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>RIGHT:8</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.10 Mid or Substring

The Parts MID and SUBSTRING are equivalent and may be used interchangeably. In the information below we will use MID, but the exact same syntax and results would apply if we were to use SUBSTRING.

The syntax is as follows MID:Start:Length

Parameter	Description
Start	Required. Starting position of the characters to return. If Start is greater than the number of characters in the source property, the Mid function returns a zero-length string (""). Start is one based.
Length	Optional. Number of characters to return. If omitted or if there are fewer than Length characters in the text (including the character at position Start), all characters from the start position to the end of the string are returned.

The MID Part takes the characters starting from the from the right of the value of the source document property in the current document and copies it into the destination property.

The example below takes the first seven characters of the value of TrueMclTitle and places them into MclTitle. If the value of TrueMclTitle were 'Content Transformation Services' then the value of MclTitle would become 'Services'.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>TrueMclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>MID:9:14</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.11 Remove Left

The REMOVE LEFT Part removes the specified number of characters from the value of the source document property in the current document and copies the result into the destination property.

The syntax is as follows REMOVE LEFT:Count

Parameter	Description
Count	The number of characters to remove.

The example below modifies the original value of Document Class and places the modified value into Document Type. If the value of Document Class were 'EQG_General' the new value of Document Type would be 'General'.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue">
  <PropertyName>Document Type</PropertyName>
  <PropertyScope>DocumentProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>Document Class</PropertyName>
      <PropertyScope>DocumentProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>Document Type</PropertyName>
      <PropertyScope>DocumentProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>REMOVE LEFT:4</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.12 Remove Right

The REMOVE RIGHT Part removes the specified number of characters from the value of the source document property in the current document and copies the result into the destination property.

The syntax is as follows REMOVE RIGHT:Count

Parameter	Description
Count	The number of characters to remove.

The example below modifies the original value of Document Class and places the modified value into Document Type. If the value of Document Class were 'EQG_General' the new value of Document Type would be 'EQG'.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue">
  <PropertyName>Document Type</PropertyName>
  <PropertyScope>DocumentProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>Document Class</PropertyName>
      <PropertyScope>DocumentProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>Document Type</PropertyName>
      <PropertyScope>DocumentProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>REMOVE RIGHT:8</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.13 Remove After Last Instance Of Character

The REMOVE AFTER LAST INSTANCE OF CHARACTER Part takes the value of the source document property up to the last instance of the specified character in the current document and copies it into the destination property.

One case where this is especially useful is when you want to take the file name of a file without the extension and use that in another property.

The syntax is as follows REMOVE AFTER LAST INSTANCE OF CHARACTER:searchCharacter

Parameter	Description
searchCharacter	The character that marks the end of the usable value.

The example below modifies the original value of FileName and places the modified value into Title. If the value of FileName were 'Master File Index.xls' the new value of Title would be 'Master File Index'.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue">
  <PropertyName xsi:type="xsd:string">Title</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>FileName</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>Title</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>REMOVE AFTER LAST INSTANCE OF CHARACTER:.</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.14 Replace

The REPLACE Part takes the value of the source document property exactly as it exists in the current document and copies it into the destination property.

The syntax is as follows REPLACE:oldValue:newValue

Parameter	Description
oldValue	Value to be replaced.
newValue	Value to replace all occurrences of oldValue.

The REPLACE Part replaces all instances of oldValue with the value of newValue from the value of the source document property in the current document and copies it into the destination property.

The example below modifies the original value of TrueMclTitle and places the modified value into MclTitle. If the value of TrueMclTitle were 'Metadata Transformation Services' the new value of MclTitle would be 'Content Transformation Services'.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>TrueMclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>REPLACE:Metadata:Content</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.15 Index

The INDEX Part takes one value from multi-valued source property and copies it into the destination property.

There are occasions when one specific value is desired from a multi-valued property for use in a single-valued property. By using the INDEX part you can specify which value to copy.

If the source property had two properties, ('A345', 'A594'), an INDEX of 1 would return 'A345', while 2 would return 'A594'. If the value of index is greater than then number of values, an empty string will be returned.

Parameter	Description
Index	Determines which value to return based on the current order. Index is one based.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>Tag Number</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>Equipment ID</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>INDEX:1</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.16 Trim

The TRIM Part takes the value of the source document property in the current document, removes any leading or trailing spaces and copies the result into the destination property.

You can provide an optional trim character(s) to remove something other than spaces from the front or back of the value. For example TRIM:~ would remove any leading or trailing instance of the character ~ from the value. TRIM:Mcl would remove any leading or trailing instances of Mcl from the value.

Usage:

Trim any leading or trailing spaces from the source value before passing to the destination value.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>TrueMclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>TRIM</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.17 Trim Left

The TRIM LEFT Part takes the value of the source document property in the current document, removes any leading spaces and copies the result into the destination property.

You can provide an optional trim character(s) to remove something other than spaces from the front of the value. For example TRIM:~ would remove any leading instance of the character ~ from the value. TRIM:Mcl would remove any leading instances of Mcl from the value.

Usage:

Trim any leading spaces from the source value before passing to the destination value.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>TrueMclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>TRIM LEFT</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.18 Trim Right

The TRIM RIGHT Part takes the value of the source document property in the current document, removes any trailing spaces and copies the result into the destination property.

You can provide an optional trim character(s) to remove something other than spaces from the front or back of the value. For example TRIM:~ would remove any trailing instance of the character ~ from the value. TRIM:Mcl would remove any trailing instances of Mcl from the value.

Usage:

Trim any trailing spaces from the source value before passing to the destination value.

ChangePropertyValue

```
<Action xsi:type="ChangePropertyValue">
<PropertyName>MclTitle</PropertyName>
<PropertyScope>VersionProperty</PropertyScope>
<VersionIndex>0</VersionIndex>
<SourceType>DataLookup</SourceType>
<DataLookup xsi:type="DataParser">
  <SourceProperty>
    <PropertyName>TrueMclTitle</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
  </SourceProperty>
  <DestinationProperty>
    <PropertyName>MclTitle</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
  </DestinationProperty>
  <Part>TRIM</Part>
  </DataLookup>
</Action>
```

3.4.1.6.3.1.19 Regex

The REGEX Part evaluates the value of the source document property as it exists in the current document against the regular expression and copies the matches into the destination property. If the destination is a multi-valued property then each match will be copied into a separate value.

While regular expressions can be quite challenging to compose and test, that is a topic that is out of the scope of this documentation. A very good regular expression editor can be downloaded for free at http://www.ultrapico.com/Expresso.htm.

The syntax is as follows REGEX:BEGIN*regular expression*END:optional BEGIN_REPLACE*Replace Expression*END_REPLACE:optional first return index

Parameter	Description
Regular Expression	Required. The regular expression to be evaluated. It must be preceded by 'BEGIN*' and followed by '*END'. This ensures that the parser will know where to start and stop the evaluation of the regular expression itself. In addition, if the regular expression contains any characters that are reserved in XML (such as '<' or '>') those characters must be escaped using standard XML escape syntax. The example regular expression below uses escaped greater than and less than characters for named groups.
Replace Expression	Optional. The replacement expression to be used to create the replacement value(s). It must be preceded by 'BEGIN_REPLACE*' and followed by *END_REPLACE'. This ensures that the parser will know where to start and stop the evaluation of the replacement expression itself. Note that if a replacement expression is defined, the values returned will be partially determined by the replacement expression.
First Return Index	Optional. If there is more than one match found, the First Return Index is the first one to be given back. Index is one based.
Last Return Index	Optional. If there is more than one match found, the Last Return Index is the last one to be given back. Index is one based.

ChangePropertyValue

```
  <Action xsi:type="ChangePropertyValue">
    <PropertyName>MclTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>TrueMclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>MclTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>REGEX:BEGIN*(?&lt;ProceedingType&gt;[a-zA-Z]{2})-
(?&lt;Year&gt;(?:\d{4}|\d{2}))-(?&lt;Sequence&gt;(?:\d{4}|\d{2}))*END:optional
BEGIN_REPLACE*Replace Expression*END_REPLACE:optional first return index:optional last
return index</Part>
  </DataLookup>
```

```
</Action>
```

3.4.1.6.3.2 Data Map

A Data Map is used to resolve the new value from an external data source, such as a database. In the case where multiple values are returned from the search, the first value is the only value used. The value can be filtered based on the value of one or more properties in the document.

The key sections of the ChangePropertyValue via Data Map are listed below.

Section	Description
PropertyName	The name of the property whose value is to be changed..
PropertyScope	Specifies the scope of the property to change. Possible values are **VersionProperty** and **DocumentProperty**.
VersionIndex	Specifies the specific version of the document to apply the change to. The index is zero based, meaning the first version is specified with a value of zero. The value -1 may also be specified to affect all versions.
SourceType	Specifies the general manner in which the source value is resolved. Possible values are **Literal** and **DataLookup**. *Note that for a* **DataMap** *the only valid value in this field is* **DataLookup**.
ConnectionString	OLE DB compliant connection string for connecting to an external data source. Any valid OLE DB connection string may be used.
QueryTarget	The name of the table, query or view to resolve the value from.
SourceColumn	The column the value to be applied to the document.
Criteria	The collection of search criteria used to filter the results. For more details see the Search Criteria topic.

ChangePropertyValue via DataMap Sample

```
<Action xsi:type="ChangePropertyValue" Name="FileNetClass" Description="FileNetClass">
  <PropertyName>FileNetClass</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <AllowDuplicates>false</AllowDuplicates>
  <DataLookup xsi:type="DataMap">
     <SearchType>PropertySearch</SearchType>
     <OrderBy/>
     <LimitResults>0</LimitResults>
     <DistinctQuery>false</DistinctQuery>
     <ConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program
Files\ECMG\Content Transformation Services\Config\AppSettings.mdb;Persist Security
Info=False</ConnectionString>
     <QueryTarget>OpsFacDiscCat</QueryTarget>
     <SourceColumn>DisciplineCode</SourceColumn>
     <ResultColumns>
     <string>DisciplineCode</string>
     </ResultColumns>
     <Clauses>
       <Clause>
         <Criteria>
           <Criterion>
             <Name>CategoryCode</Name>
             <PropertyName>Category1</PropertyName>
             <PropertyScope>VersionProperty</PropertyScope>
             <Operator>opEquals</Operator>
             <SetEvaluation>seAnd</SetEvaluation>
             <DataType>ecmString</DataType>
             <Values/>
             <Cardinality>ecmSingleValued</Cardinality>
           </Criterion>
           <Criterion>
             <Name>FacilityCode</Name>
             <PropertyName>MyFacilityCode</PropertyName>
             <PropertyScope>VersionProperty</PropertyScope>
             <Operator>opEquals</Operator>
             <SetEvaluation>seAnd</SetEvaluation>
             <DataType>ecmString</DataType>
             <Values/>
             <Cardinality>ecmMultiValued</Cardinality>
           </Criterion>
         </Criteria>
         <SetEvaluation>seAnd</SetEvaluation>
       </Clause>
     </Clauses>
   </DataLookup>
</Action>
```

ChangePropertyValue for Multi-Valued Destination Property via DataMapCOPYCODE

```
<Action xsi:type="ChangePropertyValue" Name="Change BusinessUnits" Description="">
  <PropertyName>BusinessUnits</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <AllowDuplicates>false</AllowDuplicates>
  <DataLookup xsi:type="DataMap">
    <SearchType>PropertySearch</SearchType>
    <OrderBy />
    <LimitResults>0</LimitResults>
    <DistinctQuery>false</DistinctQuery>
    <ConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Documents\BusinessUnitLookups.xls;Extended Properties=Excel 8.0;Persist
Security Info=False</ConnectionString>
    <QueryTarget>Sheet1$</QueryTarget>
    <SourceColumn>BusinessUnits_mv</SourceColumn>
    <Delimiter>|</Delimiter>
    <ResultColumns>
    <string>BusinessUnits_mv</string>
    </ResultColumns>
    <Clauses>
      <Clause>
        <Criteria>
          <Criterion>
            <Name>StudyNumber</Name>
            <PropertyName>Study</PropertyName>
            <PropertyScope>VersionProperty</PropertyScope>
            <Operator>opEquals</Operator>
            <SetEvaluation>seAnd</SetEvaluation>
            <DataType>ecmString</DataType>
            <Cardinality>ecmSingleValued</Cardinality>
          </Criterion>
        </Criteria>
        <SetEvaluation>seAnd</SetEvaluation>
      </Clause>
    </Clauses>
  </DataLookup>
</Action>
```

3.4.1.6.3.2.1 Search Criteria

The search criteria section is evaluated at runtime relative to the specific repository type to search. The provider will translate the criteria into a repository specific search.

The key sections of the Criteria collection are listed below.

NOTE: The Criteria section is now nested within a Clause section of a Clauses section. This is different from previous versions of CTS.

Section	Description
Criteria	This section contains the search criteria for a search against the a content repository. One or more Criterion sections may be included.
Criterion	Each criterion section defines an individual where clause to be included in the search.
Name	The descriptive name for the criterion. The value is optional.
PropertyName	The property name as defined in the repository.
PropertyScope	The scope of the property to include in the search criteria. Possible values are **VersionProperty** and **DocumentProperty**.
Operator	The operator to evaluate the property against. Possible values are **opEquals**, **opLessThan**, **opGreaterThan**, **opLessThanOrEqualTo**, **opGreaterThanOrEqualTo**, **opLike**, **opNotLike**, **opIn**.
SetEvaluation	Determines how two or more Criterion objects will relate to one another in a search. Possible values are **seAnd** and **seOr**.
Value	Single-valued property value. *Not used when searching based on document values.*
Values	Collection of multi-valued property values. *Not used when searching based on document values.*
Cardinality	Specifies whether or not the specified property is a single valued or multi valued field. Possible values are **ecmSingleValued** and **ecmMultiValued**.

Usage:

The example below is for a search against an external Microsoft Access database using filtering on the column DocumentCategoryCode with the document value of the property HESSCategory1 and the column FacilityCode with the document value of the property HESSFacilityCode.

Search Criteria

```
<Clauses>
  <Clause>
    <Criteria>
      <Criterion>
```

```
        <Name>DocumentCategoryCode</Name>
        <PropertyName>HESSCategory1</PropertyName>
        <PropertyScope>VersionProperty</PropertyScope>
        <Operator>opEquals</Operator>
        <SetEvaluation>seAnd</SetEvaluation>
        <Value />
        <Values />
        <Cardinality>ecmSingleValued</Cardinality>
      </Criterion>
      <Criterion>
        <Name>FacilityCode</Name>
        <PropertyName>HESSFacilityCode</PropertyName>
        <PropertyScope>VersionProperty</PropertyScope>
        <Operator>opEquals</Operator>
        <SetEvaluation>seAnd</SetEvaluation>
        <Value />
        <Values />
        <Cardinality>ecmSingleValued</Cardinality>
      </Criterion>
    </Criteria>
  </Clause>
</Clauses>
```

3.4.1.6.3.2.2 OLE DB Connections

A Data Map gets its information via an OLE DB connection. Any available OLE DB provider may be used.

Special Note for connecting to an Excel spreadsheet.

Example:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\Index\DataLoad.xls;Extended
Properties=Excel 8.0;Persist Security Info=False
```

Jet Provider Using the Data Link Properties Dialog Box

From Microsoft KB 257819

If you use the ADO Data Control or the Data Environment in your application, then the **Data Link Properties** dialog box is displayed to gather the necessary connection settings.

1. On the Provider tab, select the Jet 4.0 Provider; the Jet 3.51 Provider does not support the Jet ISAM drivers. If you specify the Jet 3.51 Provider, at run time you receive the following error message:
   Couldn't find installable ISAM.
2. On the Connection tab, browse to your workbook file. Ignore the "User ID" and "Password" entries, because these do not apply to an Excel connection. (You cannot open a password-protected Excel file as a data source.
3. On the All tab, select Extended Properties in the list, and then click Edit Value. Enter Excel 8.0; separating it from any other existing entries with a semicolon (;). If you omit this step, you receive an error message when you test your connection, because the Jet Provider expects a Microsoft Access database unless you specify otherwise.
4. Return to the Connection tab and click Test Connection. Note that a message box appears informing you that the process has succeeded.

3.4.1.7 Change Property Cardinality

The ChangePropertyCardinality action changes the cardinality of an existing property.

When migrating from one repository to another there are occasionally cases where a property is defined as a single valued attribute in one system and a multi valued attribute in another system. This action enables the change from one cardinality to another in a transformation.

The key sections of the ChangePropertyCardinality action are listed below.

Section	Description
PropertyName	The name of the property to change.
PropertyScope	The scope of the property to change. Possible values are **VersionProperty** and **DocumentProperty**.
Cardinality	The new cardinality desired. If the existing cardinality is the same as the new cardinality, no change will occur and the ActionResult will return a Success flag of false.
MultiValuedPropertyDelimiter	The character sequence used to delimit individual values in a multi valued property.

ChangePropertyCardinality

```
<Action xsi:type="ChangePropertyCardinality" Name="" Description="">
  <PropertyName>TagNumber</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <Cardinality>ecmMultiValued</Cardinality>
  <MultiValuedPropertyDelimiter>|</MultiValuedPropertyDelimiter>
</Action>
```

3.4.1.8 Change Content Retrieval Name

The ChangeContentRetrievalName action changes the file name of a content element in the document.

The are four basic ways to utilize the ChangeContentRetrievalName action. These are listed below.

- **Literal Value Substitution (Section 3.4.1.6.1)**
  - Apply a simple hard coded value to the property.
- **Data Parser**
  - Resolve the new value from some variation of an existing value of another property of the document.
- **Data Map (Section 3.4.1.6.3.2)**
  - Resolve the new value from an external data source, such as a database. The value can be filtered based on the value of one or more properties in the document.
- **Data List**
  - Resolves the new value from a simple list.

The structure and syntax of the ChangeContentRetrievalName part is identical to the structure and syntax of the **ChangePropertyValue (Section 3.4.1.6)** part. The primary difference is in the type attribute of the Action node. For ChangeContentRetrievalName it is ALWAYS ChangeContentRetrievalName. For ChangePropertyValue it is ALWAYS ChangePropertyValue. The only other important difference is that for the ChangeContentRetrievalName part the PropertyName and DestinationProperty values NEVER change except for the possible exception of the VersionIndex.

The reason that the structure and behavior is so similar to the ChangePropertyValue part is that the ChangeContentRetrievalName transformation action is actually sub-classed from the ChangePropertyValue transformation action.

The key sections of the ChangeContentRetrievalName are listed below.

Section	Description
PropertyName	For ChangeContentRetrievalName the value should ALWAYS be ContentPath.
PropertyScope	For ChangeContentRetrievalName the value should ALWAYS be VersionProperty.
VersionIndex	The zero based index for the version to apply to. For all versions use the value -1.
SourceType	This will either be **Literal (Section 3.4.1.6.1)** or **DataParser**.

ChangeContentRetrievalName

```
<Action xsi:type="ChangeContentRetrievalName" Name="ContentPath"
Description="ContentPath">
  <PropertyName>ContentPath</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>0</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty AutoCreate="false" Persistent="false">
      <PropertyName>DocumentTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
      <ValueIndex>First</ValueIndex>
    </SourceProperty>
    <DestinationProperty AutoCreate="false" Persistent="false">
      <PropertyName>ContentPath</PropertyName>
      <PropertyScope>DocumentProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
      <ValueIndex>First</ValueIndex>
    </DestinationProperty>
    <Part>COMPLETE</Part>
  </DataLookup>
  <ContentElementIndex>0</ContentElementIndex>
</Action>
```

3.4.1.9 Change Content Mime Type

The ChangeContentMimeType action changes the mime type of a content element in the document.

The are four basic ways to utilize the ChangeContentMimeType action. These are listed below.

- **Literal Value Substitution (Section 3.4.1.6.1)**
  - Apply a simple hard coded value to the property.
- **Data Parser (Section 3.4.1.6.3)**
  - Resolve the new value from some variation of an existing value of another property of the document.
- **Data Map (Section 3.4.1.6.3.2)**
  - Resolve the new value from an external data source, such as a database. The value can be filtered based on the value of one or more properties in the document.
- **Data List**
  - Resolves the new value from a simple list.

The structure and syntax of ChangeContentMimeType part is identical to the structure and syntax of the **ChangePropertyValue (Section 3.4.1.6)** part. The primary difference is in the type attribute of the Action node. For ChangeContentRetrievalName it is ALWAYS ChangeContentRetrievalName. For ChangePropertyValue it is ALWAYS ChangePropertyValue. The only other important difference is that for ChangeContentMimeType part the PropertyName and DestinationProperty values NEVER change except for the possible exception of the VersionIndex.

The reason that the structure and behavior is so similar to the ChangePropertyValue part is that ChangeContentMimeType transformation action is actually sub-classed from the ChangePropertyValue transformation action.

The key sections of ChangeContentMimeType are listed below.

Section	Description
PropertyName	For ChangeContentMimeType the value should ALWAYS be ContentMimeType.
PropertyScope	For ChangeContentMimeType the value should ALWAYS be VersionProperty.
VersionIndex	The zero based index for the version to apply to. For all versions use the value -1.
SourceType	This will either be **Literal (Section 3.4.1.6.1)** or DataLookup.
ContentElementIndex	The zero based index for the content element to apply to.

ChangeContentMimeType via Literal

```
<Action xsi:type="ChangeContentMimeType">
  <PropertyName>ContentMimeType</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <PropertyValue xsi:type="xsd:string">image/jpg</PropertyValue>
  <VersionIndex>-1</VersionIndex>
  <SourceType>Literal</SourceType>
  <ContentElementIndex>0</ContentElementIndex>
</Action>
```

ChangeContentMimeType via DataList

```
<Action xsi:type="ChangeContentMimeType">
  <PropertyName>ContentMimeType</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <PropertyValue xsi:type="xsd:string">image/jpg</PropertyValue>
  <VersionIndex>-1</VersionIndex>
  <!--<SourceType>Literal</SourceType>-->
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataList" CaseSensitive="false">
    <SourceProperty>
      <PropertyName>FileExtension</PropertyName>
      <PropertyScope>ContentProperty</PropertyScope>
      <VersionIndex>-1</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>ContentMimeType</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>-1</VersionIndex>
    </DestinationProperty>
    <MapList>
      <!-- Microsoft Word Files-->
      <ValueMap Original="doc" Replacement="application/msword"/>
      <ValueMap Original="docx" Replacement="application/vnd.openxmlformats-
officedocument.wordprocessingml.document"/>
      <ValueMap Original="dot" Replacement="application/msword"/>
      <ValueMap Original="dotx" Replacement="application/vnd.openxmlformats-
officedocument.wordprocessingml.template"/>
      <!-- Microsoft Excel Files-->
      <ValueMap Original="xls" Replacement="application/vnd.ms-excel"/>
      <ValueMap Original="xlsx" Replacement="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet"/>
      <ValueMap Original="xlt" Replacement="application/vnd.ms-excel"/>
      <ValueMap Original="xltx" Replacement="application/vnd.openxmlformats-
officedocument.spreadsheetml.template"/>
      <!-- Microsoft PowerPoint Files-->
      <ValueMap Original="ppt" Replacement="application/vnd.ms-powerpoint"/>
      <ValueMap Original="pptx" Replacement="application/vnd.openxmlformats-
officedocument.presentationml.presentation"/>
      <ValueMap Original="pps" Replacement="application/vnd.ms-powerpoint"/>
      <ValueMap Original="ppsx" Replacement="application/vnd.openxmlformats-
officedocument.presentationml.slideshow"/>
      <ValueMap Original="pot" Replacement="application/vnd.ms-powerpoint"/>
      <ValueMap Original="potx" Replacement="application/vnd.openxmlformats-
officedocument.presentationml.template"/>
      <!-- Microsoft Outlook Files-->
      <ValueMap Original="msg" Replacement="application/x-filenet-filetype-msg"/>
      <!-- Adobe Pdf Files-->
      <ValueMap Original="pdf" Replacement="application/pdf"/>
    </MapList>
  </DataLookup>
  <ContentElementIndex>0</ContentElementIndex>
</Action>
```

3.4.1.10 Change All Times To UTC

This action is used to update all the date/time values document to the current UTC equivalent.

The key sections of the ChangeAllTimesToUTC are listed below.

Section	Description
PropertyScope	Specifies the scope of the property to change. Possible values are **VersionProperty** and **DocumentProperty**.

ChangeAllTimesToUTC

```
<Action xsi:type="ChangeAllTimesToUTC" Name="" Description="">
  <PropertyScope>VersionProperty</PropertyScope>
</Action>
```

3.4.1.11 Change Date Time Property Value To UTC

This action is used to update the value of a date property to the current UTC equivalent.

The key sections of the ChangeDateTimePropertyValueToUTC are listed below.

Section	Description
PropertyName	The name of the property whose value is to be changed..
PropertyScope	Specifies the scope of the property to change. Possible values are **VersionProperty** and **DocumentProperty**.
VersionIndex	Specifies the specific version of the document to apply the change to. The index is zero based, meaning the first version is specified with a value of zero. The value -1 may also be specified to affect all versions.
SourceType	Specifies the general manner in which the source value is resolved. Possible values are Literal and **DataLookup**. *Note that for a* **DataParser** *the only valid value in this field is* **DataLookup**.

ChangeDateTimePropertyValueToUTC

```
<Action xsi:type="ChangeDateTimePropertyValueToUTC" Name="TestDate"
Description="TestDate">
  <PropertyName>TestDate</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>-1</VersionIndex>
  <SourceType>Literal</SourceType>
</Action>
```

3.4.1.12 Delete Property Action

The DeletePropertyAction removes a property from the document.

The key sections of the CreatePropertyAction are listed below.

Section	Description
PropertyName	The name of the property to be deleted.
PropertyScope	Specifies the scope of the property to be deleted. Possible values are **VersionProperty** and **DocumentProperty**.

DeletePropertyAction

```
<Action xsi:type="DeletePropertyAction" Name="Delete CombinedTitle" Description="">
  <PropertyName>CombinedTitle</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
</Action>
```

3.4.1.13 Decision Action

The DecisionAction evaluates a test value using information in the document. If the evaluation returns true then the actions specified in the TrueActions section are executed. If the evaluation returns false then the actions specified in the FalseActions section are executed.

The DecisionAction is based on the **ChangePropertyValue (Section 3.4.1.6)** action. It may use either a **DataParser (Section 3.4.1.8)** or a **DataMap (Section 3.4.1.6.3.2)**.

The key sections of the DecisionAction are listed below.

Section	Description
Name	The name of the action.
Operator	The operator to be used in the evaluation. Possible values are **opEquals**, **opLessThan**, **opGreaterThan**, **opLessThanOrEqualTo**, **opGreaterThanOrEqualTo**, **opLike**, **opNotLike**, **opIn.** Note that **opLike**, **opNotLike**, **opIn** may only be used with a TestValueType of **ecmString**.
CaseSensitive	Determines whether the evaluation should be case sensitive.
PropertyName	The name of the property to evaluate against the TestValue
PropertyScope	The scope Specifies the scope of the property to change. Possible values are **VersionProperty** and **DocumentProperty**.
VersionIndex	Specifies the specific version of the document to apply the change to. The index is zero based, meaning the first version is specified with a value of zero. The value -1 may also be specified to affect all versions.
SourceType	Specifies the general manner in which the source value is resolved. The only supported value is **DataLookup**.
DataLookup	The specification for the value to be looked up. For more details see **Data Parser (Section 3.4.1.8)**.
TestValue	The value to be evaluated against.
TestValueType	The data type of the value to be evaluated. Possible values are **ecmString**, **ecmLong**, **ecmDouble**, **ecmDate** and **ecmBoolean**.
TrueActions	The set of actions to be executed if the evaluation returns True.
FalseActions	The set of actions to be executed if the evaluation returns False.

DecisionAction

```
<Action xsi:type="DecisionAction" Name="UBR Calc Decision" Operator="opEquals"
```

```
CaseSensitive="true">
  <PropertyName>ReportID</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <VersionIndex>-1</VersionIndex>
  <SourceType>DataLookup</SourceType>
  <DataLookup xsi:type="DataParser">
    <SourceProperty>
      <PropertyName>ReportID</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </SourceProperty>
    <DestinationProperty>
      <PropertyName>ReportID</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
    </DestinationProperty>
    <Part>COMPLETE</Part>
  </DataLookup>
  <TestValue>UBRCALC</TestValue>
  <TestValueType>ecmString</TestValueType>
  <TrueActions>
    <Action xsi:type="CreatePropertyAction" Name="Create Report Name">
      <PropertyName>ReportName</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <PropertyValue xsi:type="xsd:string">UBR Calculation</PropertyValue>
      <PropertyType>ecmString</PropertyType>
      <FirstVersionOnly>false</FirstVersionOnly>
      <VersionScope>AllVersions</VersionScope>
    </Action>
  </TrueActions>
  <FalseActions>
    <Action xsi:type="DeletePropertyAction" Name="Delete Report ID">
      <PropertyName>ReportID</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
    </Action>
    <Action xsi:type="ExitTransformationAction" Name="Exit Transform" />
  </FalseActions>
</Action>
```

3.4.1.14 ContentFileHasExtensionTest Action

The ContentFileHasExtensionTestAction checks to see if the specified version and content file has a file extension. If a file extension is present then the actions specified in the TrueActions section are executed, otherwise the actions specified in the FalseActions section are executed.

Section	Description
VersionIndex	Specifies the specific version of the document to test. The index is zero based, meaning the first version is specified with a value of zero. The value -1 may also be specified to test the latest version.
ContentIndex	Specifies the specific content element of the document to test. The index is zero based, meaning the first content element is specified with a value of zero.

In the example below, we are working with a file exported using the File System Provider. It has a FileName property at the version level. If the first content file of the latest version has an extension, we will create a property called **DocumentTitle** and get everything before the last period in the file name. If the content file has no extension then we will use the entire file name.

ContentFileHasExtensionTestAction

```
<Action xsi:type="ContentFileHasExtensionTestAction">
  <VersionIndex>-1</VersionIndex>
  <ContentIndex>0</ContentIndex>
  <TrueActions>
    <Action xsi:type="CreatePropertyAction" Name="Create Document Title">
      <PropertyName>DocumentTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <PropertyValue xsi:type="xsd:string"></PropertyValue>
      <PropertyType>ecmString</PropertyType>
      <FirstVersionOnly>false</FirstVersionOnly>
      <VersionScope>AllVersions</VersionScope>
    </Action>
    <Action xsi:type="ChangePropertyValue">
      <PropertyName>DocumentTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
      <SourceType>DataLookup</SourceType>
      <AllowDuplicates>false</AllowDuplicates>
      <DataLookup xsi:type="DataParser">
        <SourceProperty AutoCreate="false" Persistent="false">
          <PropertyName>FileName</PropertyName>
          <PropertyScope>VersionProperty</PropertyScope>
          <Type>ecmString</Type>
          <VersionIndex>0</VersionIndex>
          <ValueIndex>First</ValueIndex>
        </SourceProperty>
        <DestinationProperty AutoCreate="false" Persistent="false">
          <PropertyName>DocumentTitle</PropertyName>
          <PropertyScope>VersionProperty</PropertyScope>
          <Type>ecmString</Type>
          <VersionIndex>0</VersionIndex>
          <ValueIndex>First</ValueIndex>
        </DestinationProperty>
        <Part>REMOVE AFTER LAST INSTANCE OF CHARACTER:.</Part>
      </DataLookup>
    </Action>
  </TrueActions>
  <FalseActions>
    <Action xsi:type="CreatePropertyAction" Name="Create Document Title">
      <PropertyName>DocumentTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <PropertyValue xsi:type="xsd:string"></PropertyValue>
      <PropertyType>ecmString</PropertyType>
      <FirstVersionOnly>false</FirstVersionOnly>
      <VersionScope>AllVersions</VersionScope>
    </Action>
    <Action xsi:type="ChangePropertyValue">
      <PropertyName>DocumentTitle</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>0</VersionIndex>
      <SourceType>DataLookup</SourceType>
      <AllowDuplicates>false</AllowDuplicates>
      <DataLookup xsi:type="DataParser">
        <SourceProperty AutoCreate="false" Persistent="false">
          <PropertyName>FileName</PropertyName>
          <PropertyScope>VersionProperty</PropertyScope>
          <Type>ecmString</Type>
          <VersionIndex>0</VersionIndex>
          <ValueIndex>First</ValueIndex>
        </SourceProperty>
        <DestinationProperty AutoCreate="false" Persistent="false">
          <PropertyName>DocumentTitle</PropertyName>
          <PropertyScope>VersionProperty</PropertyScope>
          <Type>ecmString</Type>
          <VersionIndex>0</VersionIndex>
          <ValueIndex>First</ValueIndex>
        </DestinationProperty>
        <Part>COMPLETE</Part>
      </DataLookup>
    </Action>
  </FalseActions>
</Action>
```

3.4.1.15 Exit Transformation

The ExitTransformationAction stops the transformation process at the current location. It is typically used in conjunction with a DecisionAction.

The key sections of the ExitTransformationAction are listed below.

ExitTransformationAction

```
<Action xsi:type="ExitTransformationAction" Name="Exit Transform" />
```

3.4.1.16 Rename Property Action

The RenamePropertyAction renames an existing property in the document.

This is used in cases where a property will be used as from the source repository in the target repository with renaming the property the only change needed.

The key sections of the RenamePropertyAction are listed below.

Section	Description
PropertyName	The current name of the property.
PropertyScope	Specifies the scope of the property. Possible values are **VersionProperty** and **DocumentProperty**.
NewName	The new name to be given to the property.

RenamePropertyAction

```
<Action xsi:type="RenamePropertyAction">
  <PropertyName>HESSSystem</PropertyName>
  <PropertyScope>VersionProperty</PropertyScope>
  <NewName>HESSSystemRpt</NewName>
</Action>
```

3.4.1.17 Run Transformation Action

The RunTransformationAction executes another transformation against the document.

The key sections of the RunTransformationAction are listed below.

RunTransformationAction

```
<Action xsi:type="RunTransformationAction" Name="Run Base Transform" Description="">
  <TransformationPath>C:\Transformations\Import from File System to Base P8 Document
Class.ctf</TransformationPath>
</Action>
```

3.4.1.18 Split Property Value

The SplitPropertyValue action splits multiple values from a single property and allocates each individual value to different destination properties. The classic scenario for this action is to take a structured folder path and assign each level to a specified property.

The are four basic ways to utilize the SplitPropertyValue action. These are listed below.

- **Literal Value Substitution (Section 3.4.1.6.1)**
  - Apply a simple hard coded value to the property.
- **Data Parser (Section 3.4.1.8)**
  - Resolve the new value from some variation of an existing value of another property of the document.
- **Data Map (Section 3.4.1.6.3.2)**
  - Resolve the new value from an external data source, such as a database. The value can be filtered based on the value of one or more properties in the document.
- **Data List**
  - Resolves the new value from a simple list.

The structure and syntax of the SplitPropertyValue part is identical to the structure and syntax of the **ChangePropertyValue (Section 3.4.1.6)** part. The primary difference is in the type attribute of the Action node. For SplitPropertyValue it is ALWAYS SplitPropertyValue . For ChangePropertyValue it is ALWAYS ChangePropertyValue.

The reason that the structure and behavior is so similar to the ChangePropertyValue part is that the SplitPropertyValue transformation action is actually sub-classed from the ChangePropertyValue transformation action.

The key sections of the SplitPropertyValue are listed below.

Section	Description
PropertyName	For ChangeContentRetrievalName the value should ALWAYS be ContentPath.
PropertyScope	For ChangeContentRetrievalName the value should ALWAYS be VersionProperty.
VersionIndex	The zero based index for the version to apply to. For all versions use the value -1.
SourceType	This will either be **Literal (Section 3.4.1.6.1)** or **DataParser (Section 3.4.1.8)**.

SplitPropertyAction

```
<Action xsi:type="SplitPropertyAction" Name="Split Folder Path into multiple
properties" Description="">
  <PropertyName>Folder Path</PropertyName>
  <PropertyScope>DocumentProperty</PropertyScope>
  <Delimiter>/</Delimiter>
  <TrimPrefix>/</TrimPrefix>
  <StartPosition>1</StartPosition>
  <SourceProperty AutoCreate="false" Persistent="false">
    <PropertyName>Folders Path</PropertyName>
    <PropertyScope>DocumentProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
    <ValueIndex>First</ValueIndex>
```

```xml
  </SourceProperty>
  <DestinationProperties>
    <LookupProperty AutoCreate="true" Persistent="true">
      <PropertyName>Document Type</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>-1</VersionIndex>
      <ValueIndex>First</ValueIndex>
    </LookupProperty>
    <LookupProperty AutoCreate="true" Persistent="true">
      <PropertyName>Discipline</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>-1</VersionIndex>
      <ValueIndex>First</ValueIndex>
    </LookupProperty>
    <LookupProperty AutoCreate="true" Persistent="true">
      <PropertyName>Business Unit</PropertyName>
      <PropertyScope>VersionProperty</PropertyScope>
      <VersionIndex>-1</VersionIndex>
      <ValueIndex>First</ValueIndex>
    </LookupProperty>
  </DestinationProperties>
</Action>
```

4 Files

4.1 Content Definition File

A content definition file (cdf) is an abstract representation of a single repository document. It contains all of the available meta data for a single document stack.

5 Sample Files

5.1 Set File Name to Document Title

This sample only has one action. It takes the value of DocumentTitle and uses it to change the file name for the content associated with the document.

Complete Transformation File

```
<?xml version="1.0" encoding="utf-8"?>
<Transformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" Name="Set File Name to Document Title">
  <ExclusionPath>\</ExclusionPath>
  <Exclusions />
  <Actions>
    <Action xsi:type="ChangeContentRetrievalName" Name="Set File Name to Document
Title" Description="">
    <PropertyName>ContentPath</PropertyName>
    <PropertyScope>VersionProperty</PropertyScope>
    <VersionIndex>0</VersionIndex>
    <SourceType>DataLookup</SourceType>
    <DataLookup xsi:type="DataParser">
      <SourceProperty>
        <PropertyName>DocumentTitle</PropertyName>
        <PropertyScope>VersionProperty</PropertyScope>
        <VersionIndex>0</VersionIndex>
      </SourceProperty>
      <DestinationProperty>
        <PropertyName>ContentPath</PropertyName>
        <PropertyScope>DocumentProperty</PropertyScope>
        <VersionIndex>0</VersionIndex>
      </DestinationProperty>
      <Part>COMPLETE</Part>
    </DataLookup>
    </Action>
  </Actions>
</Transformation>
```

6 Index

7 Developer Overview

The Content Transformation Services framework contains a powerful API for working with documents and repositories. All of the CTS applications are built on this API.