



Adding Intelligence to Media

INTRODUCTION TO ASSET RELATIONSHIPS

Copyright © 2014 Adobe Systems Incorporated. All rights reserved.

Introduction to Asset Relationships.

Adobe, the Adobe logo, Creative Cloud, Creative Suite, InDesign, InCopy, Illustrator, Photoshop, Premiere, and Prelude are either registered trademarks or trademarks of Adobe Systems Inc. in the United States and/or other countries. Microsoft and Windows are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple, Mac OS, and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. Java and Sun are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other trademarks are the property of their respective owners.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Inc. Adobe Systems Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

Adobe Systems Inc., 345 Park Avenue, San Jose, California 95110, USA.

1 Asset Relationships in XMP

This overview introduces the concepts you need to understand why preserving asset relationships is important in managing asset efficiently, and how such relationships can be managed using the Extensible Metadata Platform (XMP).

Preserving the asset relationships in documents that your application creates and edits opens doors for many important workflows, particularly those that involve digital asset management (DAM). In order to participate in these workflows, you need to understand the XMP asset relationship model, and leverage its power by using and updating the metadata values correctly in all documents that your applications create, edit, and export.

- ▶ This document will help you become familiar with the terms and concepts used in the XMP asset-relationship model.
- ▶ For more complete technical details on this subject, see the *Partners Guide to XMP for Dynamic Media*.
- ▶ Refer to the *XMP Specification* for detailed descriptions of the XMP data model, file type support, and schemas supported in Adobe® products.

All of these documents are available from <http://www.adobe.com/devnet/xmp.html>.

Overview

Modern documents are often composed by importing or including other documents, in whole or in part. For example, a text document can include other text documents or images; an image can be composed from other images, and can also include text. A video or audio file is often composed using clips from other video and audio files.

It is important to keep track of the assets used in a document's composition, so that you can implement various kinds of searches and other asset management workflows. This is particularly useful when you store the document in a digital asset management (DAM) system. Both the document itself and the assets used in its composition can change over time, and version control can become very complex.

Asset relationship workflows

Complex documents created with applications such as Photoshop, Illustrator, or InDesign include XMP metadata that identifies assets used in their composition. The XMP asset-management metadata model is intended to support workflows such as these:

Asset creation hierarchy

A very basic asset-relationship workflow is retrieving and displaying the hierarchy of assets used in the composition of a document. An application might, for example, show a graphical representation of the composition, or might allow a user to download included assets in order for the document to be viewed correctly.

Need for update

Access to source documents is particularly important if you want to update your document. You can monitor a document's included assets to see if any have been modified since it was included.

Sub-asset extraction	If the application that creates a document updates the XMP with information about assets that are used in it, a DAM system can extract the information, allowing it to manage assets more efficiently. If a new document is uploaded that uses assets that are already present, the DAM system can simply increase the reference count of those assets rather than uploading a new copy.
Asset usage analytics	A DAM system can use asset relationship data to track how assets (or even parts of assets) are being used.
Asset governance	A DAM system can use asset relationship data to control uploads in various ways. For example, you could establish a constraint that prevents upload of a document that uses any rights-protected asset, or that uses a specific asset such as a company logo.
Asset association search	<p>You can use asset relationship data to support searches of many types.</p> <p><i>Similarity search:</i> If you want to allow a user to select an image and search for similar images similar to it, you can implement this by finding documents that are in the derivation chain of a particular document.</p> <p><i>Focused search:</i> If you wish to search for something in a large movie file, for example, you can use the asset-relationship information to narrow the search to a smaller part of the composed file, and search only among assets that were used in that part.</p>
Manual association of assets	You can allow a user to manually associate an asset with the document they are creating. For example, a video editing application that allows the user to work with low-res proxies can then re-link to the real footage.

Preserving asset relationships in metadata

In order to participate in these workflows, an application is responsible for using and updating the metadata values correctly in all documents that application creates, edits, or exports. In order to make sure your documents use the XMP asset relationship model correctly, your applications must take these actions:

- ▶ When your application creates a new document it must add the necessary *document identifiers*. These identifiers form the basis of all future asset-relationship preservation. See [“Managed assets and document identifiers” on page 5](#).
- ▶ When your application converts a document from one format to another, it must add the **xmpMM:DerivedFrom** property in order to maintain the derivation history. See [“Derived assets” on page 7](#).
- ▶ For workflows where your application creates a composed document, it must incorporate information about the components using the **xmpMM:Ingredients** and **xmpMM:Pantry** entries in the final document. See [“Tracking composed asset relationships” on page 9](#).

Asset relationship terminology

We use the term **asset** to refer to any digital content, such as a digital document, image, movie, or song. Digital documents (such as PDF) are called **digital assets**, while images, movies, or songs and so on are called **media assets**.

When you create new digital content that does not reference any other asset (typically using an application's **File > New** command), this is called a **new asset** or **simple asset**. For example in Adobe Photoshop®, using the **File > New** command creates a new PSD file that has no references, and is thus a simple asset. Similarly, creating a PDF from a scanned document results in a simple asset.

When you create new file from an existing file, with Save As or Export, for example, the result is **derived asset**. The asset from which the new file was derived is the **parent asset**. A derived asset can have only one parent asset.

Composed assets

The complexities of asset management are largely concerned with **composed assets**; that is, content that is built from collections of other assets. Composed assets are created by including or linking one asset to other assets.

Static media (such as a PSD or PDF) can include images, for example. You might reference JPG files in a Photoshop PSD file. The assets used in the composition have a relatively simple relationship to the composed document, being entirely included or imported.

Dynamic media, such as audio and video files, are often built as a collection of several smaller clips, images and audio files. For example, in Premier® Pro you can add clips and images to a Sequence to create a new Movie. The asset relationships here are more complex, because the composed asset can use just **part** of another asset.

Managed assets and document identifiers

Complexities in asset management arise because the individual assets that make up a composed asset are independent documents or files that can undergo changes and versioning separately from the asset that includes or imports them. The XMP inclusion model is designed to establish and maintain asset relationships among a composed document and its included assets.

In order for your assets to participate in the asset-relationship model, it is very important that the components of a composed asset, as well as the composed asset itself, and its parent if it is a derived asset, all be identified with *globally unique identifiers*, or GUIDs.

The XMP Media Management Schema (see *XMP Specification Part 2, Standard Schemas*) defines these properties, which together uniquely identify any managed asset:

Identifier	Property	Description
<i>DocumentID</i>	xmpMM:DocumentID	<p>Uniquely identifies an asset.</p> <p>Different versions of the same asset have the same DocumentID. For example, if you edit a PSD file in Photoshop and save the changes, the DocumentID value remains the same.</p>
<i>InstanceID</i>	xmpMM:InstanceID	<p>Uniquely identifies a specific version of an asset.</p> <p>For example, if you edit a PSD file in Photoshop and save the changes, the DocumentID value remains the same, but the InstanceID changes, thus identifying a new version.</p>
<i>OriginalDocumentID</i>	xmpMM:OriginalDocumentID	<p>This value links an asset to its original source.</p> <p>For example, if you save the PSD as a JPEG, then convert the JPEG into a GIF, the immediate source of the GIF is the JPEG, but the <i>original</i> source is the PSD. Thus the OriginalDocumentID in the GIF file has the same value as the DocumentID of the PSD file, thus linking the GIF with its original source.</p> <p>For new assets, the OriginalDocumentID value must be the same as its DocumentID.</p>

An asset that has all three of these properties in its XMP Packet is a **managed asset**.

Any application that creates or operates on managed assets is responsible for adding, initializing, and updating these values correctly whenever it operates on an asset, so that the asset can participate in asset relationship workflows.

Creating and updating asset IDs

When an application creates a new asset, it must add all three document Identifiers in the XMP Packet, and initialize them correctly. For new assets, the **DocumentID** and **OriginalDocumentID** must be the same.

The initial value of **InstanceID** can be the same as or different from the **DocumentID**. Any time an application modifies the asset however, it must also update the **InstanceID**.

In order to fulfil this requirement, you must identify all routes through which your users can create new assets or modify existing assets, and make sure that the ID values are properly updated in every case

Derived assets

If your application has any kind of save-as or export functionality, it is creating derived assets. Any application that can create derived assets must preserve the *derivation history*; that is, the relationship between a derived asset and the parent asset from which it was derived.

Because you can continue to convert or export a derived asset, creating new derived assets, you must keep track of both the *original* source and the *immediate* source. In order to identify the immediate source of a derived asset, the XMP Media Management Schema defines the property **xmpMM:DerivedFrom**, whose value is a structure that references the unique IDs of the immediate parent asset.

As long as the derivation history is maintained correctly, you can then track all of the sources back through a derivation chain back to the original asset.

In order to maintain these relationships correctly, every time your application creates a derived asset, you must:

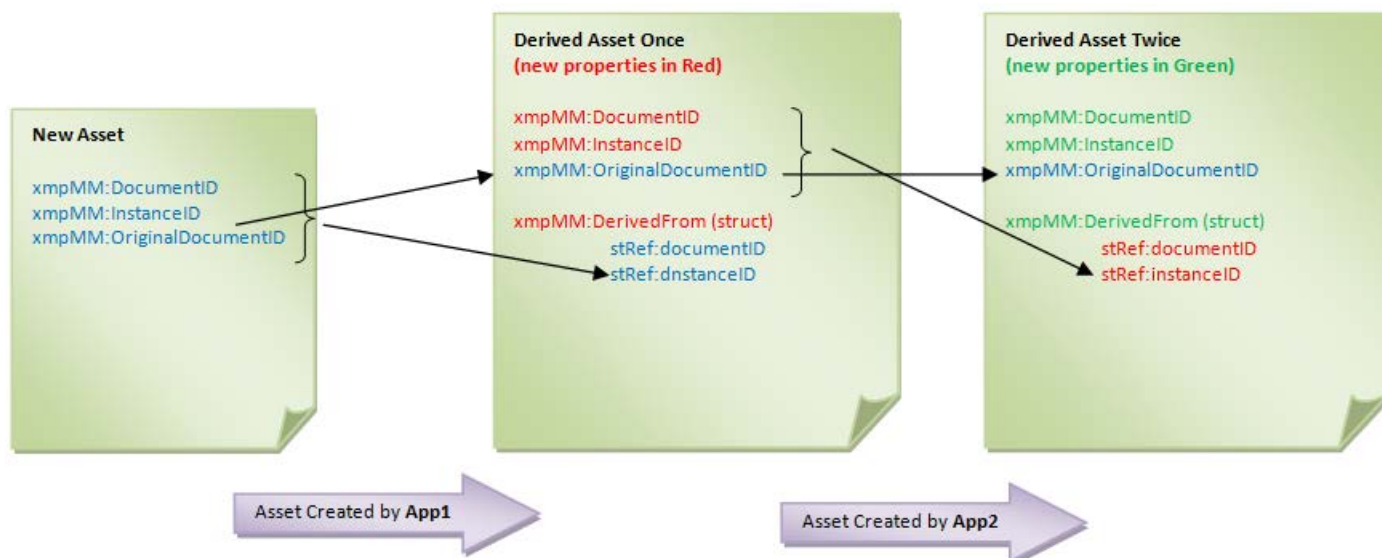
1. Create and initialize the **DocumentID** and **InstanceID** for the new derived asset;
2. Copy the **OriginalDocumentID** value from the immediate parent;
3. Create the **DerivedFrom** property, and initialize its values from the IDs of the immediate parent.

Derived asset example

We considered the case of converting a PSD file into a JPEG, then converting the JPEG into a GIF. Here, the *immediate* source of the GIF is the JPEG, while the *original* source is the PSD.

- In order to identify the original source, you simply copy the **OriginalDocumentID** from the immediate parent to the new derived asset. In our example, the GIF file must have the same **OriginalDocumentID** value as its immediate parent, the JPEG file. The value of **OriginalDocumentID** in both derived assets is the **DocumentID** of the PSD file, thus linking the GIF with its original source.
- In order to identify the immediate source, you must copy the **DocumentID** and **InstanceID** of the immediate parent into the **DerivedFrom** structure (creating this property if it does not yet exist). For the JPEG file, this is the same as the original source, and identifies the PSD file. For the GIF file, this identifies the JPEG file that is the immediate source, while the original source remains the same.

This illustration shows how the metadata tracks the derivation of an asset through two export or save-as operations:

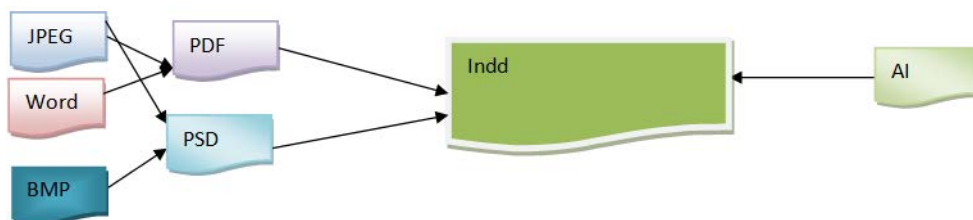


Composed asset relationships

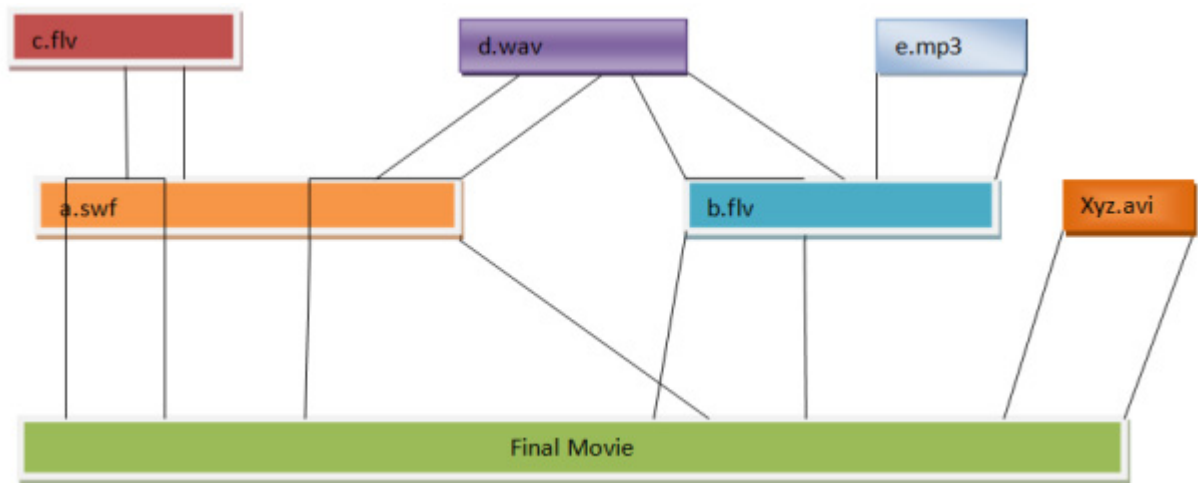
The asset relationships in a composed asset can become very complex, particularly when an included or imported asset is itself a composed asset. The metadata for a document must be able to track all of the included assets in such a way as to retain their own hierarchical structure, and make it accessible from the current document. Because of the potential complexity, this structural record must be optimized to prevent it from becoming overwhelmingly large.



A composed asset can be composed of simple assets or other composed assets.



Dynamic assets (such as videos) are more typically composed of both completely included assets and parts of other assets, as shown here.



Here, the movie includes specific sections of the `a.swf` and `b.flv` files, which in turn contain parts of other files. The inclusions belong to limited sections of the included files. Notice, for instance, that the section of `b.flv` that includes the file `e.mp3` is *not* included in the final movie. It should be possible to optimize the relationship map so that `e.mp3`, which is not actually used, is not included as a required asset in the final movie.

Tracking composed asset relationships

The XMP model for relationships among composed assets is based on a cooking metaphor; a composed dish contains *ingredients*, which are selected from a *pantry*. A recipe must have an ingredients list, which describes the ingredients used in its preparation. A pantry contains the actual physical ingredients. When you have identified what you need using the ingredients list, you go to the pantry to find them.

The XMP Media Management Schema defines the properties **xmpMM:Ingredients** and **xmpMM:Pantry** to track similar relationships in composed document whose specific components are selected from a set of available components. The ingredients list identifies the components, while the pantry provides more detailed information for each component.

- The **Ingredients** property contains an array of references to the component assets that are included in the composed asset. Each entry *identifies* a specific asset that was imported, using a reference to its **DocumentID** and **InstanceID**.
- The **Pantry** property, according to the metaphor, contains the actual physical ingredients that are listed in the recipe. In XMP, this property contains the *full description* of assets that are referenced in the **Ingredients** property. This value is an array in which each item is the extracted XMP Packet of each of the component assets.

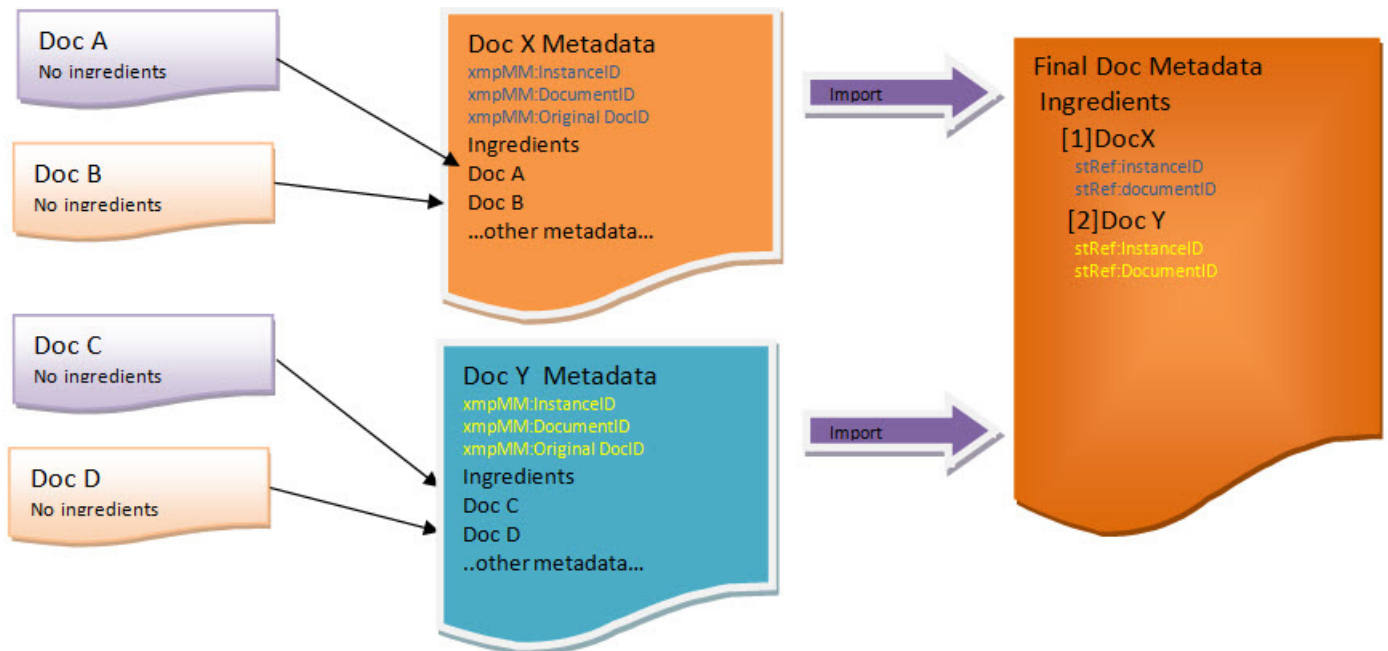
In the same way that a cook looks at the recipe to know what ingredients to find in the pantry, you look at the **Ingredients** property to determine what information to get out of the **Pantry** property. The next chapter describes in more detail exactly how this works.

2 Managing Relationships in Composed Assets

This chapter discusses techniques you can use to work with the asset relationships in composed assets, and provides some real-world case examples.

The composition tree in metadata

This shows how simple component assets are mapped into the **Ingredients** lists of two composed assets, which are then imported into a final composed asset.



Ingredients entries

Notice that the **Ingredients** property of a composed document lists only those assets that are *directly* used in its composition. Here, the Final Doc lists only Doc X and Doc Y as ingredients. Because Doc X and Doc Y are also composed documents, they have their own **Ingredients** lists.

All the properties values in an **Ingredients** list are *references*. Reference types are defined in the `ResourceRef` namespace; see details in the *XMP Specification Part 2, Standard Schemas*. In the example, you can see that the ID values are given as `stRef:instanceID` and `stRef:documentID` (note the initial lower-case in the property name). These two property references must be included to identify a component asset, but there can be additional property references in the **Ingredients** list, as needed.

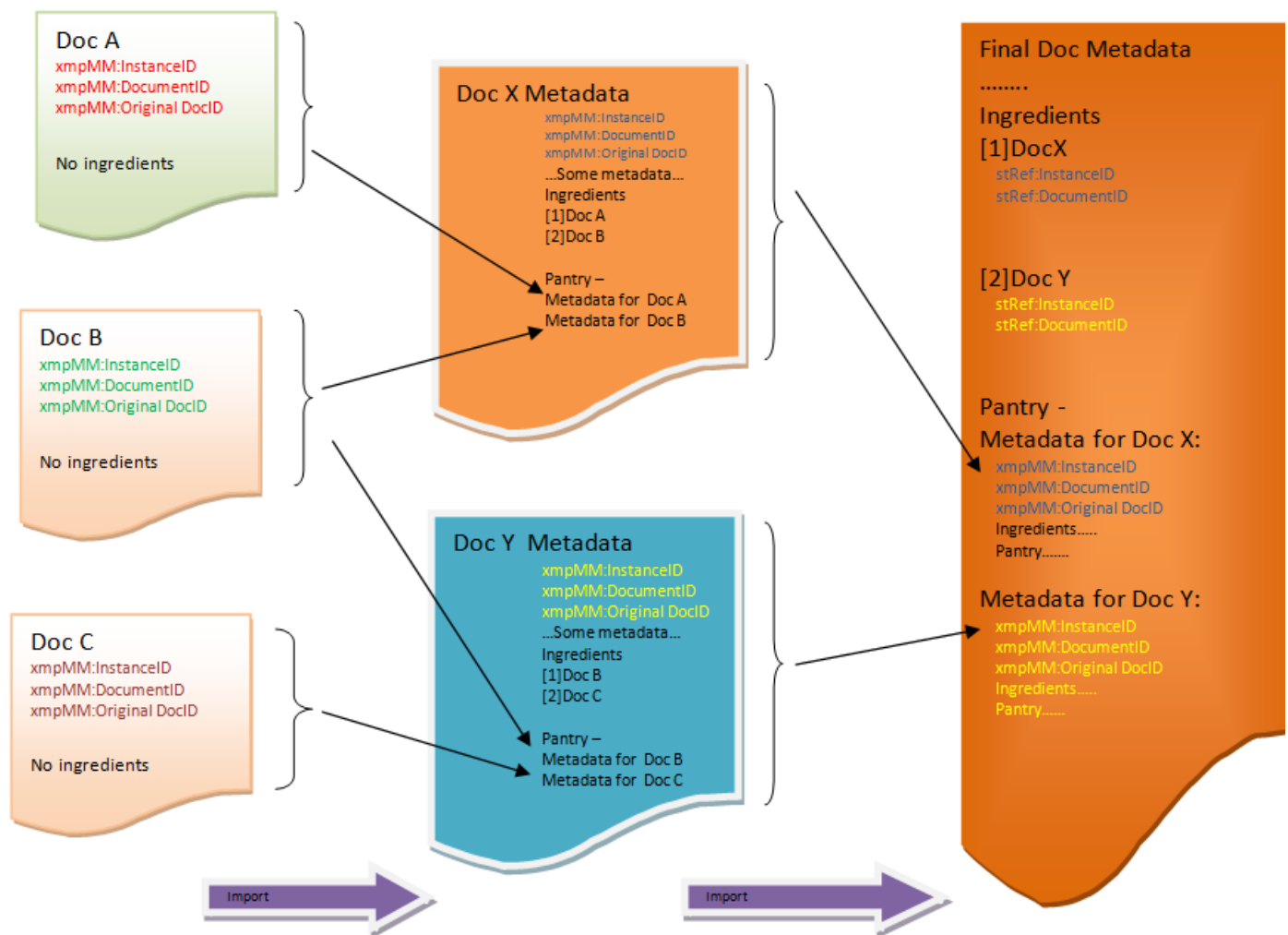
Pantry entries

In our food analogy, a pantry is the place in which ingredients can be found. The ingredients list simply identifies the ingredients, while the pantry actually provides them.

In a composed document, the **Pantry** property is intended to provide the complete information about the documents that are only identified in the **Ingredients** list. To accomplish this, each entry in the **Pantry** array contains the extracted XMP Packet from a document that is referenced in the **Ingredients** list.

Consider the case illustrated here, which shows what happens if the XMP Packet is copied directly from each component into the composed asset that imports it.

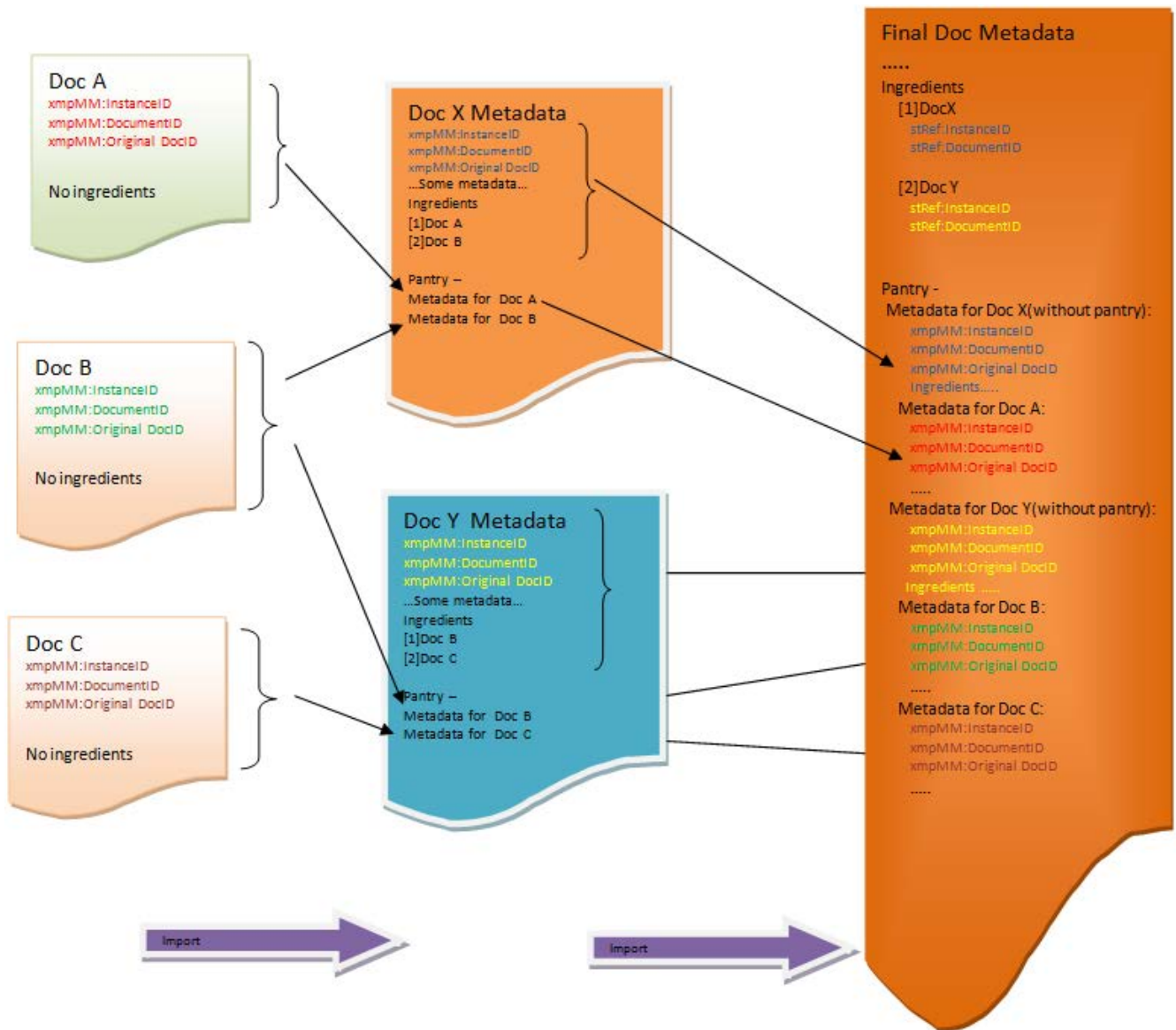
Notice that the Final Doc has the metadata from its components Doc X and Doc Y—but each of those is also a composed document, each of which imports Doc B. The metadata for Doc B appears in the **Pantry** in each of those documents. If we simply copy the **Pantry** contents from each component without optimization, the Final Doc ends up with metadata from Doc B appearing twice in its **Pantry** list.



There is an obvious problem of duplication here; the metadata of a composed asset contains the metadata of its components, which in turn contain the metadata of their components, and so on. If you don't optimize the process, there is potential for duplication of information that can get out of hand very quickly.

To avoid this kind of duplication, you must remove the **Pantry** property from each component's XMP Packet before copying the metadata into the composed document, then add the elements of the component's **Pantry** back into the **Pantry** of the composed document at the top level. This allows you to check for duplication.

The result of this technique for the example is shown in the following diagram.



Moving pantry entries from the metadata of each component into top-level entries in the **Pantry** property of the composed document introduces some further complications:

- The hierarchy of the inclusion tree is not explicitly maintained in the metadata structure, so you must iterate through the metadata of contained assets to find other assets that are more deeply nested. For an example of how to do this, see [‘Finding metadata of ingredients’ on page 13](#).
- Because *parts* of dynamic media assets can be used as components, excluding duplication becomes more complex than simply looking for duplicate asset IDs. For a discussion of how part relationships are represented, see [‘Excluding metadata from unused parts’ on page 14](#).

Finding metadata of ingredients

The **Pantry** entry lists the XMP metadata of all the assets included directly or indirectly in a composed document. The **Ingredients** list in the top-level metadata, however, only contains the references to those documents that were directly imported into the outermost document.

Although the hierarchy of the inclusion tree is not maintained in the way the metadata is stored, you can traverse the metadata structure in such a way as to find the information about any directly or indirectly included asset. In order to find the metadata of any asset that is nested more deeply in the component tree, you must recurse through **Ingredients** lists of component assets, and you must know the ID values for the document you are searching for.

Using the example in our illustration, suppose you want to see how Doc A is included in the Final Doc. You can traverse the metadata of Final Doc:

1. Starting with the **Ingredients** list of Final Doc, look in the first ingredient, Doc X, to find the **InstanceID** and **DocumentID** of the asset it points to.
2. Search the **Pantry** of Final Doc for an entry that matches this **InstanceID** and **DocumentID**.

This entry contains the metadata for Doc X, which includes that document's **Ingredients** list (but does not include the **Pantry** for that document).

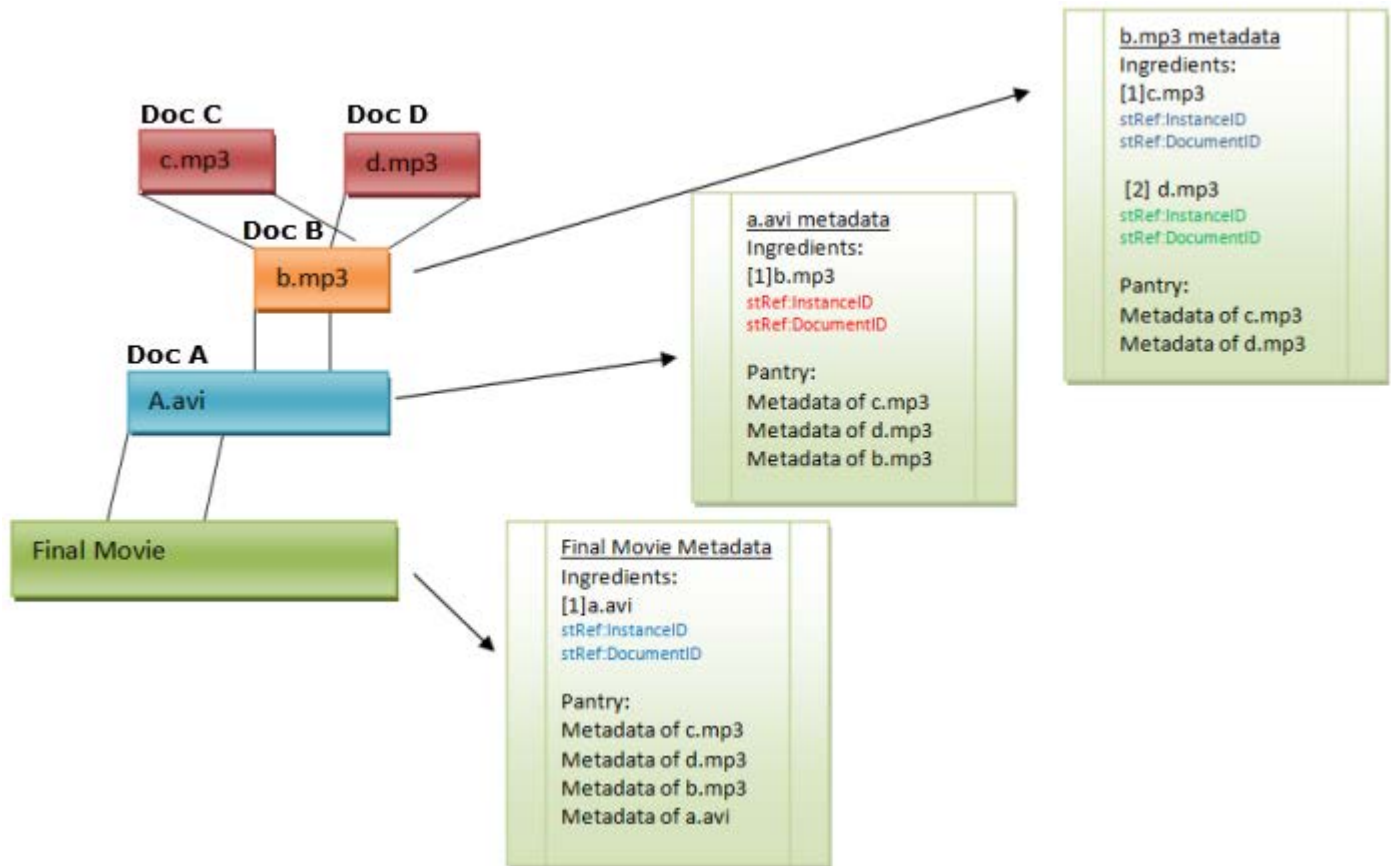
3. Check the **Ingredients** list of Doc X to find a reference to Doc A.
 - ▷ If Doc X does not contain the asset you are looking for, go on to the next **Ingredients** entry in Final Doc.
 - ▷ In this case, Doc X does contain Doc A as an ingredient, so get the **InstanceID** and **DocumentID** of the asset the **Ingredients** entry points to.
4. Search the **Pantry** of Final Doc for an entry that matches this **InstanceID** and **DocumentID**.

This is the XMP Packet for Doc A (copied from Doc X, with its original **Pantry** property removed).

You can use a similar approach to list the entire derivation chain of a document, by working back through the components of each component in the included metadata.

Excluding metadata from unused parts

Consider this example, which illustrates a typical case of a composed Dynamic Media asset.



This example shows the “part-of” relationship: Final Movie uses a part of document A (A.avi) which is itself composed from document B (b.mp3). Document B is in turn composed of documents C and D (c.mp3 and d.mp3).

Copying all of the **Ingredients** and **Pantry** information from A into the metadata of Final Movie results in the inclusion of metadata from all of the components of A. However, the part of document A that is actually included in Final Movie is not the part that is derived from document B, so that document and its components (C and D) are not, in fact, used at all in Final Movie.

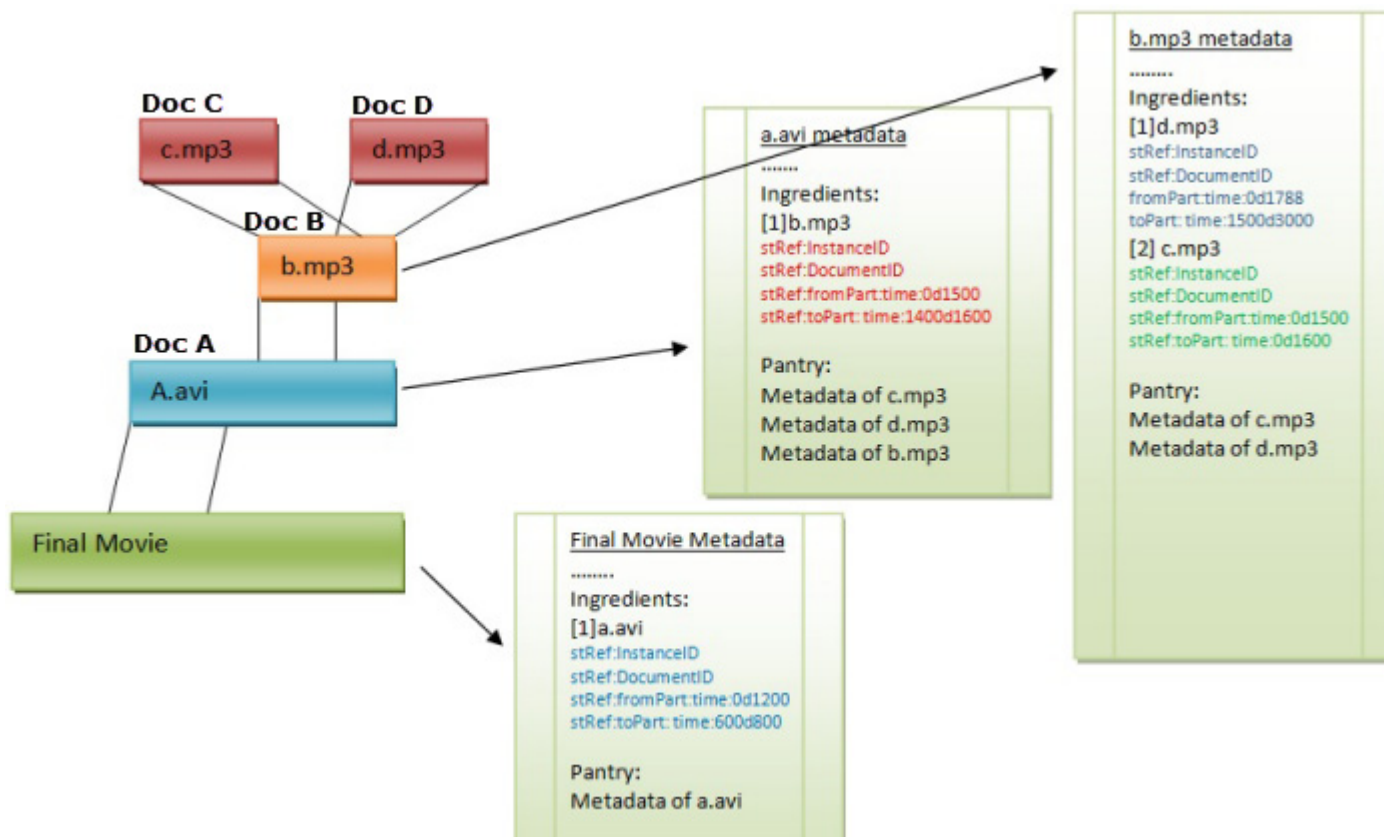
Clearly, it is not necessary to include metadata for assets that are not actually used—and doing so would quickly cause the size of the **Pantry** property to get out of hand. It is not only unnecessary, it is also undesirable, as it provides incorrect information about the actual composition of the composed asset.

In order to make it possible to exclude metadata for unused parts of component assets, we have to keep track of the parts that are used. In order to do this, XMP defines the **Part** data type to identify a portion of a document. (For details of the definition, see the *XMP Specification Part 2, Standard Schemas*).

There are two properties in XMP **ResourceRef** schema that are references to the Part data type.

- **stRef:fromPart** : Identifies the part of a component asset (ingredient) used in a composed document.
- **stRef:toPart** : Identifies the part of the composed document that incorporates the asset.

You must include these properties in the **Ingredients** list in order to keep track of what portion of a component is used in the final composed document. When we do this for the example, the **Ingredients** and **Pantry** lists in the Final Movie document can properly represent that document's composition.



In the Final Movie metadata, the **Ingredients** entry for A (A.avi) has these two additional entries:

<code>stRef:fromPart : time:0d1200</code>	The portion of A spanning from 0 to the 1200 th frame is included in Final Movie.
---	--

<code>stRef:toPart: time:600d800</code>	A part of A is mapped into the 600 th to 800 th frame of the Final Movie.
---	---

When your application imports an asset, you must use this information to decide whether you need to include the metadata of components of the imported asset into the **Pantry** of your new composed asset.

In the example, the 0-1200th frames of A need to be included into 600th-800th frames of Final Movie. To maintain the asset relationships properly:

1. First look into the **Ingredients** list of A or all the components whose **toPart** overlaps with frames 0-1200.
2. For each such ingredient, get the **InstanceID** and search in the **Pantry** of A for XMP data with this **InstanceID**.
3. Copy any such data into a top-level **Pantry** entry of Final Movie.

In this example, none of the component of A are used in the part that is used in Final Movie, so the **Pantry** of Final Movie has only one entry, the metadata for A itself. The metadata for the unused components (B, C, and D) is successfully excluded.

3 Composed Asset Use Cases

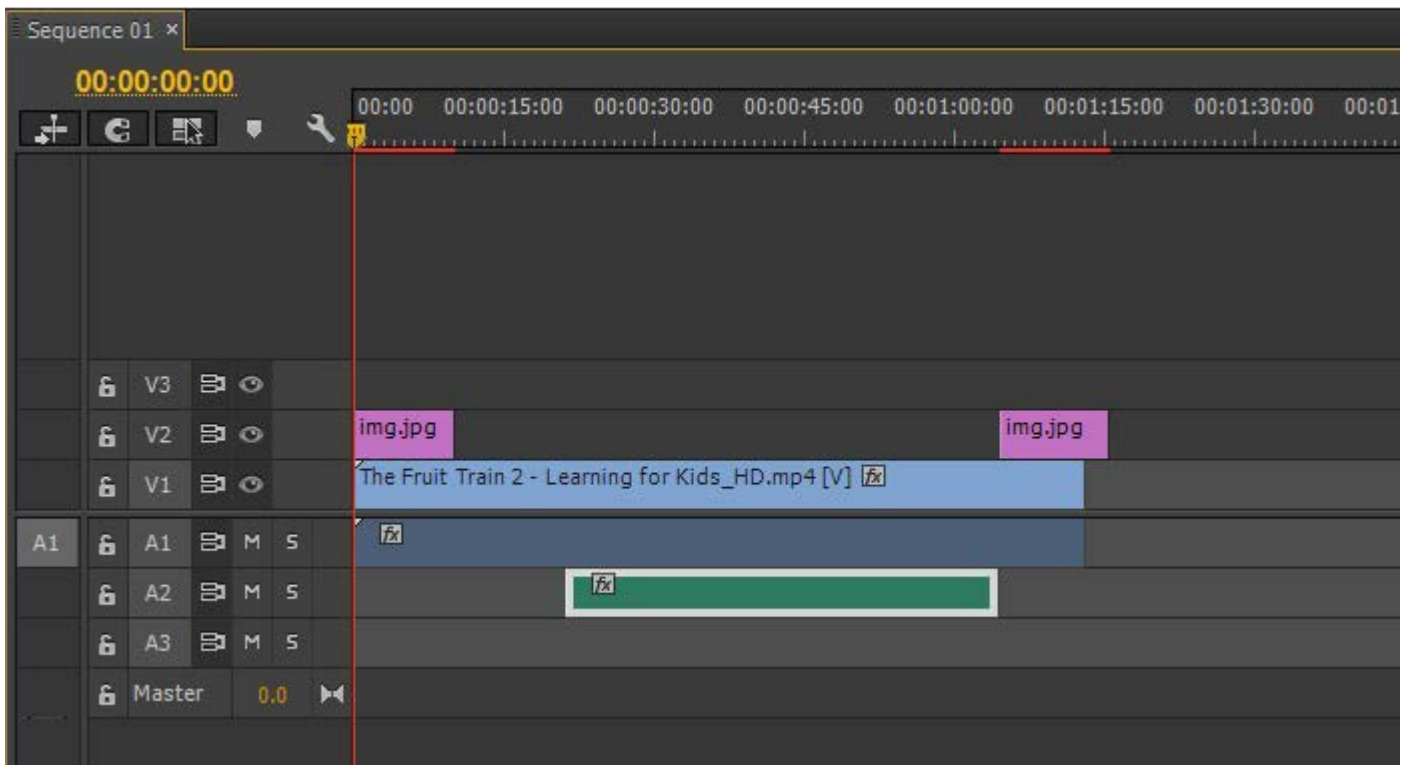
This chapter provides some real-world case examples of composed documents, showing how the metadata tracks the composition history. These examples show video files for movies created with Adobe Premiere Pro CC, which are composed using clips from other video and audio files, as well as static images.

The first example is a movie that is composed, but whose components are all whole files. The second example shows how the metadata handles the complexity introduced when a portion of the first composed file is imported into another composition.

Case 1: A movie with simple components

The first example is a movie (doc1.mp4) composed from these simple assets, shown below in the Premiere Timeline view:

- ▶ An MP4 clip, shown in blue
- ▶ Two static JPG files, shown in pink and grey
- ▶ An entire MP3 file, shown in green

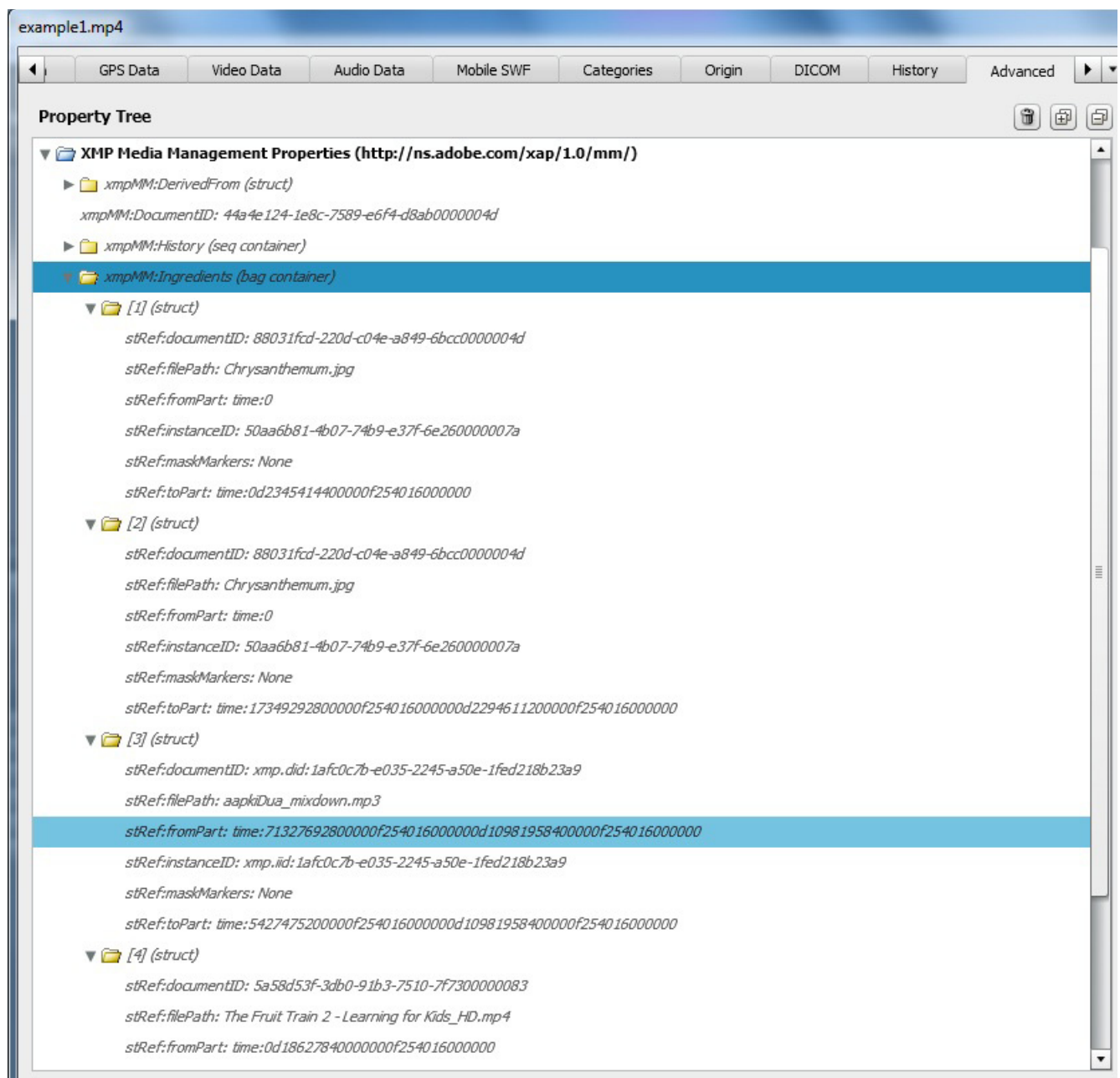


We will take a close look at how the metadata for the movie file reflects this composition.

Case 1 Ingredients

The **Ingredients** list for the movie file is shown in the flowing figure below. There are total of 5 ingredients:

- ▶ The video channel of the MP4 file
- ▶ The audio channel of MP4 file (this item is closed in the figure below)
- ▶ The JPG image included at the start (shown in pink in the timeline)
- ▶ The JPG image included at the end (shown in grey in the timeline)
- ▶ The MP3 file



The entry for each ingredient includes a reference to the **DocumentID** and **InstanceID** of the component asset; these are always required.

These entries also contain **fromPart** and **toPath** information, which is very important in Dynamic Media assets. Notice that the single JPG image was included twice in the composition, so it is treated as two different ingredients. Both these ingredient entries refer to same **DocumentID**, but their **toPart** values are different because they have been imported into different parts of the composition.


When you build the ingredients list for a composed document that your application creates, you can include any of the properties listed in the XMP **ResourceRef** schema, as needed for your particular workflows. In this example, there are some additional properties, such as **stRef:FilePath**, which identifies the file name of the original asset.

Case 1 Pantry



The Pantry property of the movie file, in contrast, has only three entries, one for each of the source documents. The first entry is for the MP4 file, the second for the JPG file (although it is used twice), and the third for the MP3 file.



If we open the first entry, we see the metadata for the MP4 file, which has been copied here.




 **XMP Media Management Properties (<http://ns.adobe.com/xap/1.0/mm/>)**

xmpMM:DocumentID: 000b081a-02ac-1bb6-1ff4-df930000003b

- ▶  *xmpMM:History (seq container)*
- ▶  *xmpMM:Ingredients (bag container)*

xmpMM:InstanceID: xmp.iid:29269108-e565-d14a-a659-b09c732da50f


xmpMM:OriginalDocumentID: xmp.did:8513545b-7303-a845-a7fe-b57099cda4d9

- ▼  *xmpMM:Pantry (bag container)*
 - ▼  *[1] (struct)*
 - ▶  *dc:contributor (bag container)*


xmp:CreateDate: 2013-04-11T16:26:02Z

xmp:MetadataDate: 2013-09-25T11:20:53+05:30

xmp:ModifyDate: 2013-09-25T05:50:21Z



- ▶  *xmpDM:duration (struct)*

xmpMM:DocumentID: 5a58d53f-3db0-91b3-7510-7f7300000083

- ▶  *xmpMM:History (seq container)*

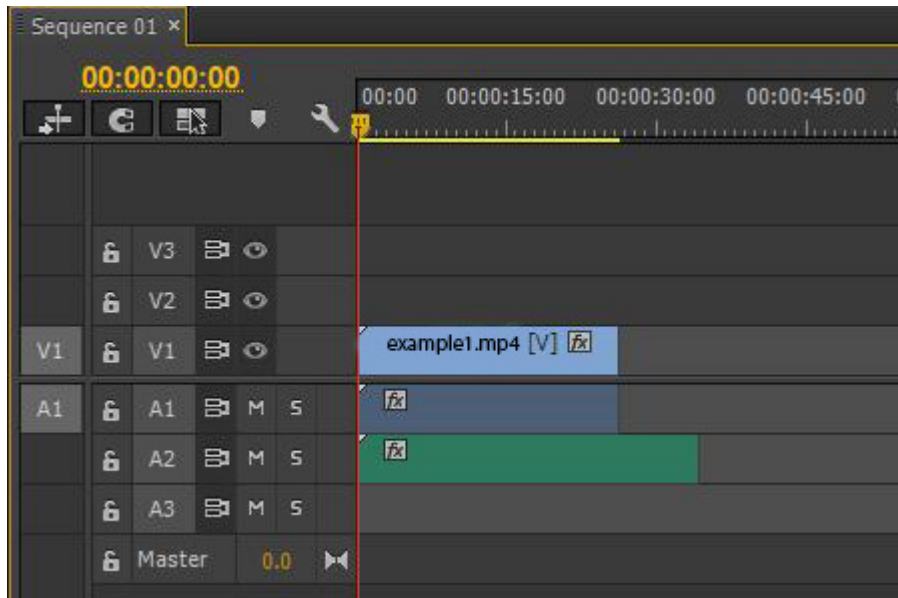
xmpMM:InstanceID: xmp.iid:23d9de10-8401-5c4d-af29-dc1e622b38f4

xmpMM:OriginalDocumentID: xmp.did:d0daa8d5-1f61-e245-a148-f680ab1e5c1a

- ▶  *[2] (struct)*
- ▶  *[3] (struct)*

Case 2: Nested composed assets

The second movie (`example2.mp4`) is composed using *part* of the composed file we used in the first example, with the addition of a new MP3 file. It is shown here in the Premier Sequence view. The two channels of the MP4 file are in blue, and the new MP3 file is in green.

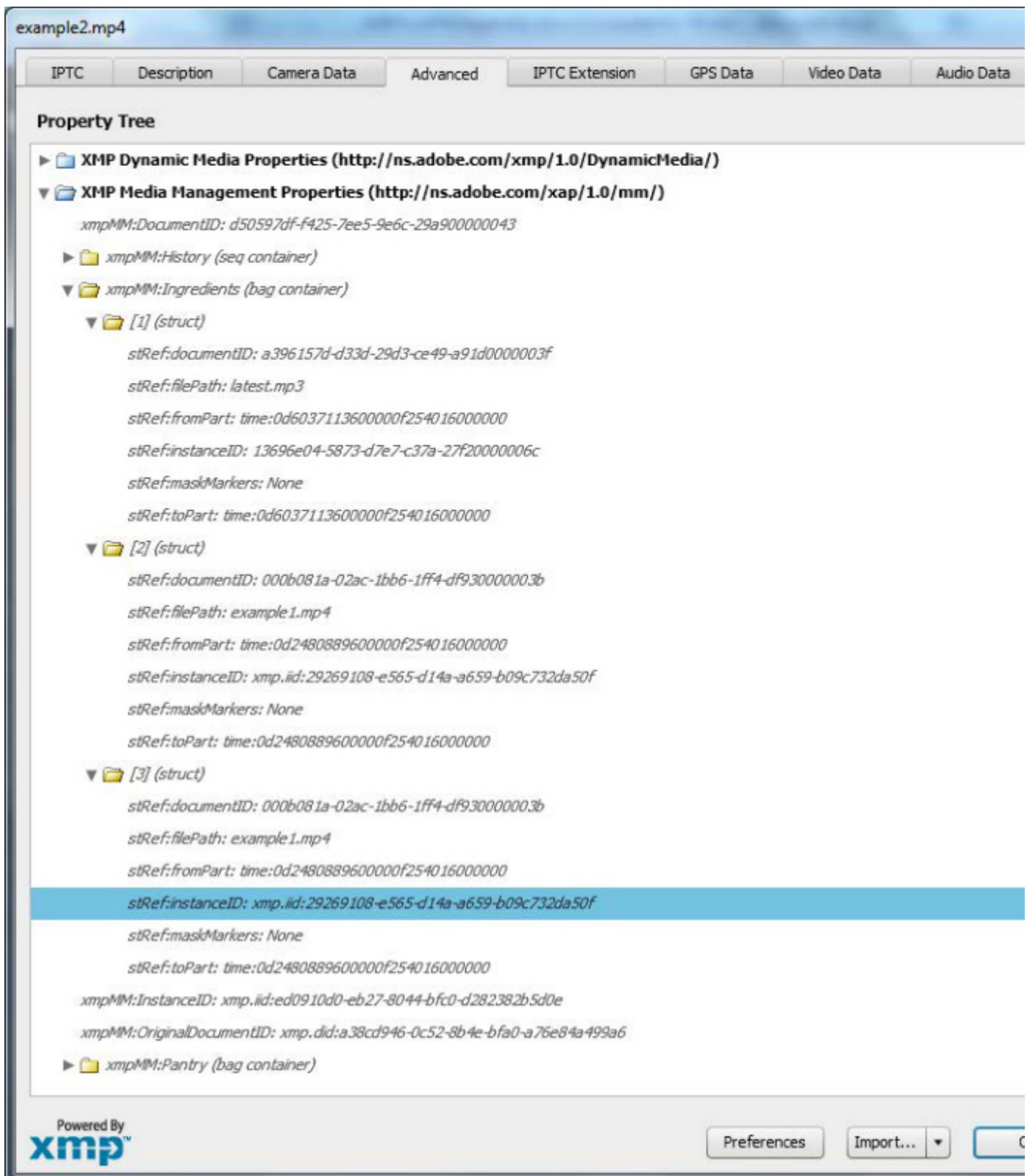


Case 2 Ingredients

There are three entries in the **Ingredients** list for this composed file:

- ▶ The video channel of the MP4 file (Example 1)
- ▶ The audio channel of the MP4 file (Example 1)
- ▶ The MP3 file

In the following figure, the **instanceID** reference for the MP4 file from the first example is highlighted.



Case 2 Pantry

The **Pantry** property for this file is a little more complicated, since one of the ingredients has its own ingredients that must be taken into account.

There are four entries:

- ▼ *xmpMM:Pantry (bag container)*
 - ▶ *[1] (struct)*
 - ▶ *[2] (struct)*
 - ▶ *[3] (struct)*
 - ▶ *[4] (struct)*

The first entry contains the metadata for the new MP3 file, which is straightforward.

- ▼ *xmpMM:Pantry (bag container)*
 - ▼ *[1] (struct)*
 - xmp:MetadataDate: 2013-09-25T12:42:05:30*
 - xmp:ModifyDate: 2013-09-25T12:42:05:30*
 - xmpDM:genre: Other*
 - xmpMM:DocumentID: a396157d-d33d-29d3-ce49-a91d0000003f*
 - ▶ *xmpMM:History (seq container)*
 - xmpMM:InstanceID: 13696e04-5873-d7e7-c37a-27f20000006c*
 - xmpMM:OriginalDocumentID: xmp.did:80d60fd3-9d7f-7643-914e-7400d3c100b5*
 - ▶ *[2] (struct)*
 - ▶ *[3] (struct)*
 - ▶ *[4] (struct)*

The second entry corresponds to the MP4 clip that was one of the component assets of Example 1. The metadata for this asset has its own ingredients list, but no pantry list. The pantry items have been moved out to the top level of the **Pantry** list for this new composed asset.

- ▼ *xmpMM:Pantry (bag container)*
 - ▶ *[1] (struct)*
 - ▼ *[2] (struct)*
 - ▶ *dc:contributor (bag container)*
 - xmp:CreateDate: 2013-04-11T16:26:02Z*
 - xmp:MetadataDate: 2013-09-25T11:20:53:05:30*
 - xmp:ModifyDate: 2013-09-25T05:50:21Z*
 - ▶ *xmpDM:duration (struct)*
 - xmpMM:DocumentID: 5a58d53f-3db0-91b3-7510-7f7300000083*
 - ▶ *xmpMM:History (seq container)*
 - xmpMM:InstanceID: xmp.iid:23d9de10-8401-5c4d-af29-dc1e622b38f4*

The third entry is for the clip of the composed video from Example 1. Again, this has its own ingredients list (along with lots of other information), but the pantry items have been moved out to the top level of the new asset.



Notice that the **Ingredients** list of this clip, however, has only three entries, not the five entries that were present in the **Ingredients** list for the entire movie. The entries are for:

- ▶ The video channel of the MP4 clip used in Example 1
- ▶ The audio channel of the MP4 clip used in Example 1
- ▶ The JPG file for the first image used in Example 1

The ingredients entries for the MP3 file and second instance of the JPG file are *not* included. When the Example 2 movie was composed, it included only a portion of Example 1, not the whole file. The portion that was included happens to be a stretch that does not incorporate any of the material imported from these two files. Any changes in those two files can never affect this new composition, so the metadata from them will never be needed in the composition history.

When Adobe Premiere saved this file, it parsed the metadata to determine that these sources were not used, and removed them from the nested **Ingredients** list, in order to reduce the amount of metadata recursion in the **Pantry** list for the new composition. This is an example of the optimization discussed in [‘Excluding metadata from unused parts’ on page 14](#).

Finally, the fourth entry in the Pantry list for Example 2 contains the metadata from the JPG file used in Example 1. Although it was not imported directly into Example 2, it is part of the clip that was imported, and changes in that source file can therefore affect Example 2.