West Street Consulting

AXCM Plugin Tutorial (XPath Schemes)

This tutorial is designed to give you a quick hands-on introduction to the AXCM plugin. It covers some of the basic functionality and should familiarize you with the core features that the software provides.

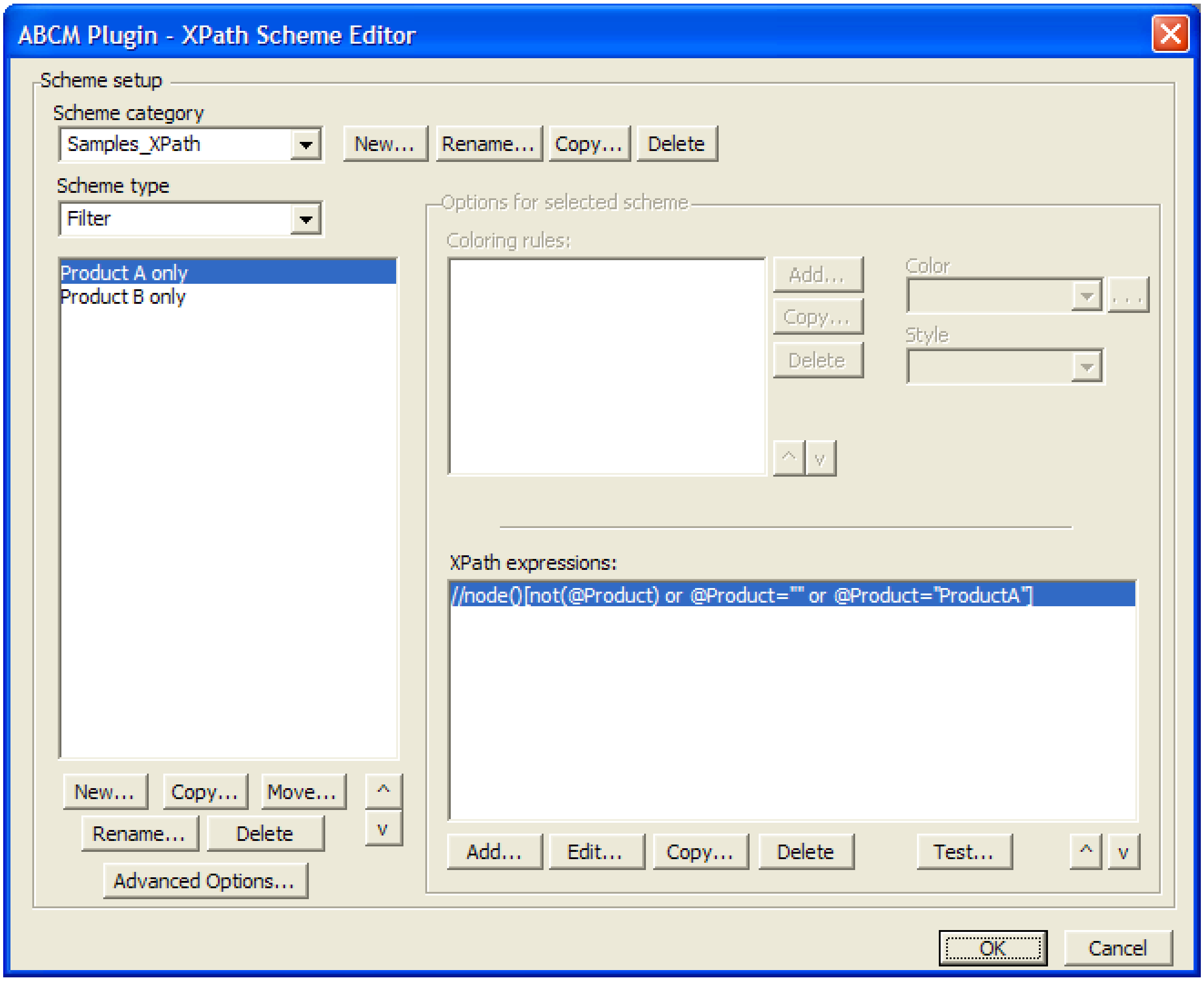
Before using this tutorial, note the following:

- This tutorial explores the XPath scheme functionality introduced in version 2.0 of the software. There is a different tutorial that explores the “classic” scheme functionality that has existed since its inception. **IMPORTANT**: You should have an understanding of the distinction between the two, especially if you intend to play a leading role in the implementation of AXCM at your organization. See the user guide for more information.
- This tutorial does not intend to serve as an XPath primer. For more information on XPath, see the AXCM and FrameSLT user guides.
- The tutorial is designed for use with the sample files that install with AXCM. The instructions in this tutorial assume that the files have not been altered since the installation. However, you will be altering them during the course of the tutorial. If you or someone else will need the original files afterwards, consider making copies of the sample files and performing the tutorial on the copies. The files do not need to be in any particular location for this tutorial to work.
- The tutorial is designed to work with the sample schemes that installed with AXCM. These schemes are contained in the sample main settings file that is provided with AXCM. If your preferences are pointing to some main settings file that does not contain this set of sample schemes, the tutorial cannot be completed. In this case, please see the AXCM documentation or contact someone with AXCM experience for help. If you have just installed AXCM and have not altered your preferences, you should be fine.
- Some of the screen shots in this tutorial still display the previous name of the plugin, "ABCM." The screenshots should otherwise be accurate.

Part 1 – Setting up your “scratch” category

The main settings file that installs with AXCM includes a “Samples_XPath” category that contains scheme data for use with this tutorial. In the following steps, you will make a copy of this category which will be used for the remainder of the tutorial. By making a copy, you will be able to edit schemes without affecting the original scheme data that came with the plugin.

1. Open the XPath scheme editor (**AXCM > Main Settings > XPath Schemes**).


ABCM Plugin - XPath Scheme Editor

Scheme setup

Scheme category: Samples_XPath

Scheme type: Filter

Product A only
Product B only

Options for selected scheme

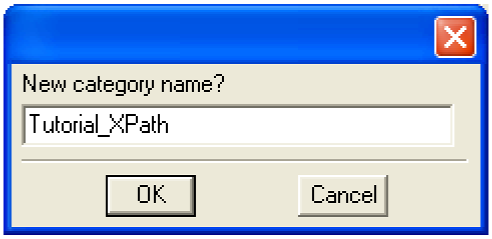
Coloring rules:

XPath expressions:

//node()[not(@Product) or @Product="" or @Product="ProductA"]

A majority of the work you do in this tutorial will focus on this dialog box. In the following steps, you will make one small change to prepare for the tutorial. Later, you will return to this dialog box to explore it in more detail.

2. Under **Scheme category**, select **Samples_XPath**.
3. To the right of the scheme category drop-down, click **Copy**.
4. In the new category prompt, type the name **Tutorial_XPath**, and click **OK**.


New category name?

Tutorial_XPath

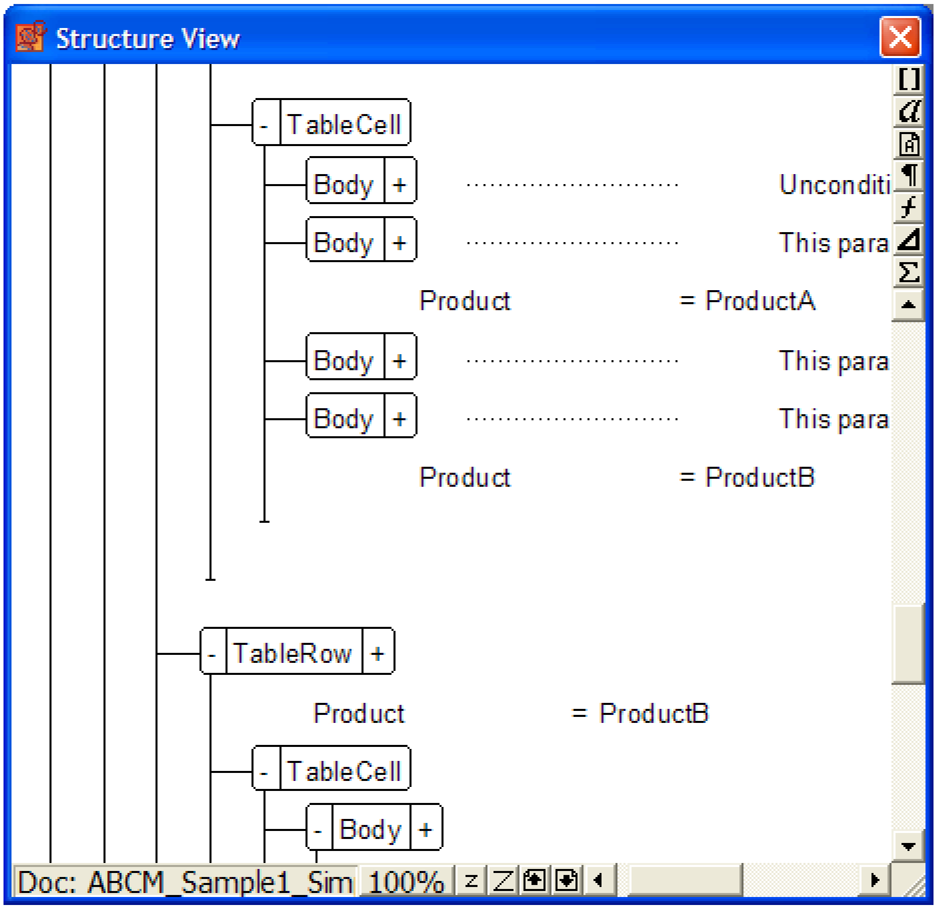
5. Note that the **Scheme category** list now contains **Samples_XPath** and **Tutorial_XPath**, which are currently exact duplicates.
6. Click **OK** to close the scheme editor.

Part 2 - Simple conditions – Coloring and filtering

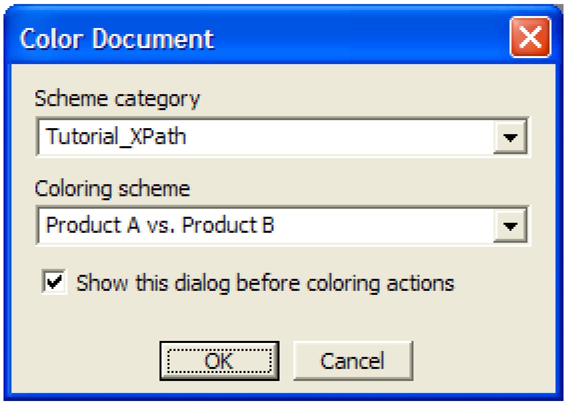
Assume that you have a FrameMaker book that you use to generate a manual for both Product A and Product B. You want to use conditional text to tag information specific to one product or the other, and you've chosen to use structured FrameMaker and AXCM for this. A conditional setup such as this is the most simple, using a single attribute and two separate values.

AXCM includes a sample file with this type of setup, using the `Product` attribute to denote the product condition with the values "ProductA" and "ProductB". Note that this is an example only, and that you may use any attributes and values you want to denote conditions.

1. Open the sample file `AXCM_Sample1_TwoSimpleConditions.fm` and close all other open files.
2. Open the **Structure View** window.
3. Note the use of the Product attribute to denote conditions, and how the text in the document is written to help you see where these conditions are.



4. Select **AXCM > Coloring > Color Document**.
5. Select the **Tutorial_XPath** category, the **Product A vs. Product B** scheme, and click **OK**.

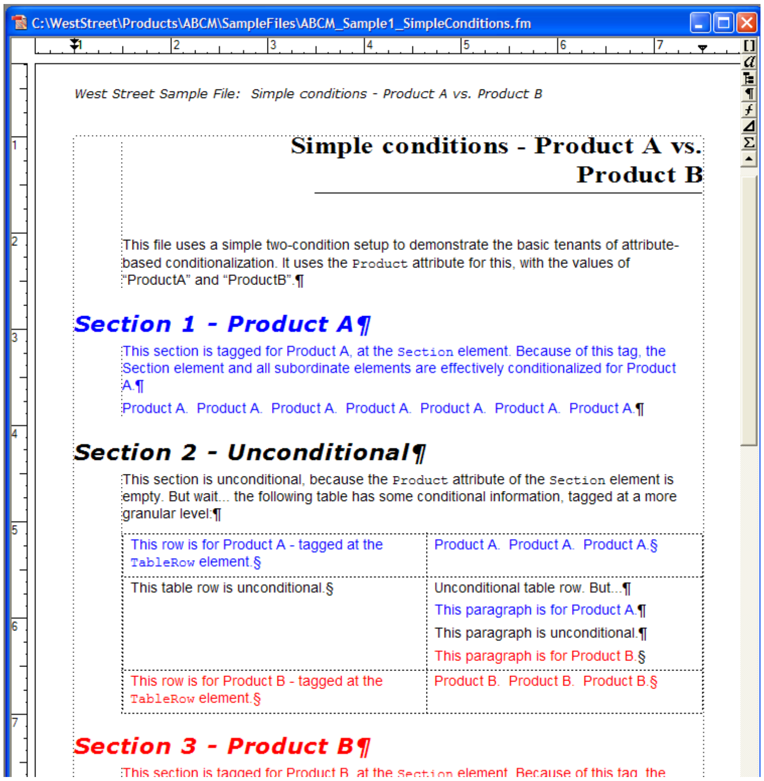


Scheme category: Tutorial_XPath

Coloring scheme: Product A vs. Product B

☑ Show this dialog before coloring actions

6. Note the coloring that was applied.


C:\WestStreet\Products\ABCM\SampleFiles\ABCM_Sample1_SimpleConditions.fm

West Street Sample File: Simple conditions - Product A vs. Product B

Simple conditions - Product A vs. Product B

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty. But wait... the following table has some conditional information, tagged at a more granular level:¶

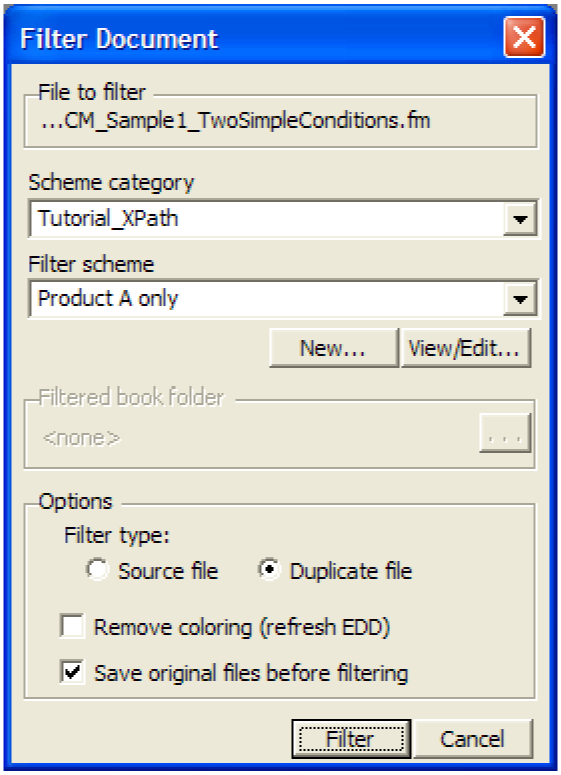
This row is for Product A - tagged at the `TableRow` element.§	Product A. Product A. Product A.§
This table row is unconditional.§	Unconditional table row. But...¶ This paragraph is for Product A.¶ This paragraph is unconditional.¶ This paragraph is for Product B.§
This row is for Product B - tagged at the `TableRow` element.§	Product B. Product B. Product B.§

Section 3 - Product B¶

This section is tagged for Product B, at the `Section` element. Because of this tag, the

The coloring you see is defined in the scheme that you selected. The scheme setup will be examined in more detail later.

7. Select **AXCM > Filtering > Filter Document**.
8. Set up the Filter dialog as shown below, with the **Tutorial_XPath** category, the **Product A only** scheme, and the other settings.



File to filter
...CM_Sample1_TwoSimpleConditions.fm

Scheme category: Tutorial_XPath

Filter scheme: Product A only

Filtered book folder
<none>

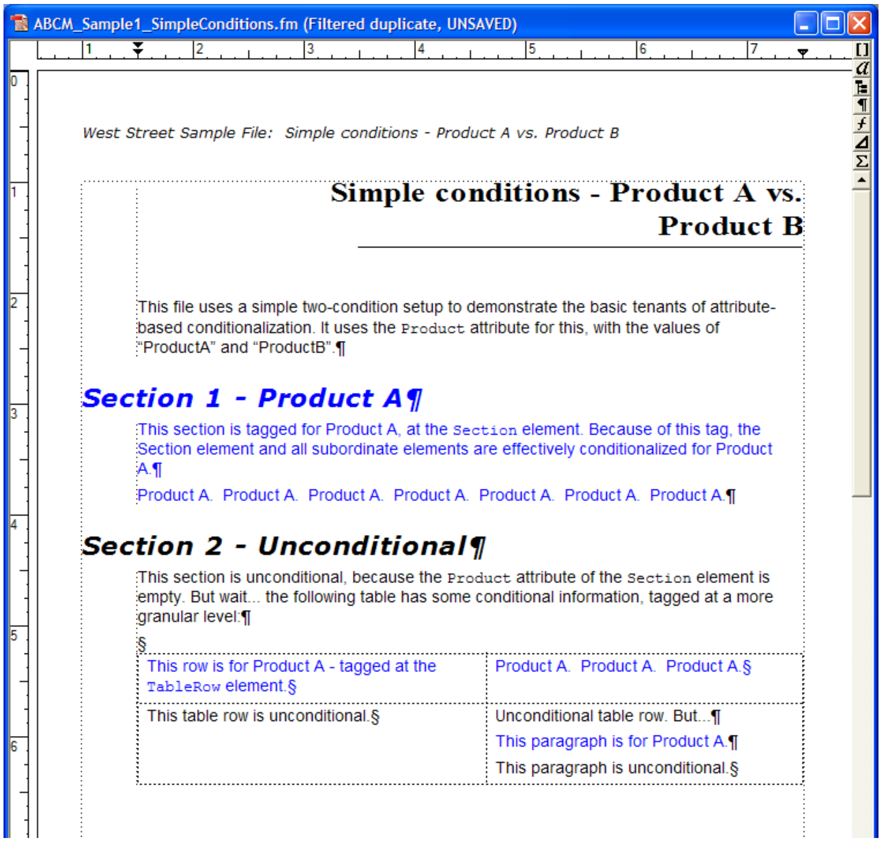
Options

Filter type:
○ Source file ● Duplicate file

☐ Remove coloring (refresh EDD)

☑ Save original files before filtering

9. Click **Filter** and note the results.


ABCM_Sample1_SimpleConditions.fm (Filtered duplicate, UNSAVED)

West Street Sample File: Simple conditions - Product A vs. Product B

Simple conditions - Product A vs. Product B

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

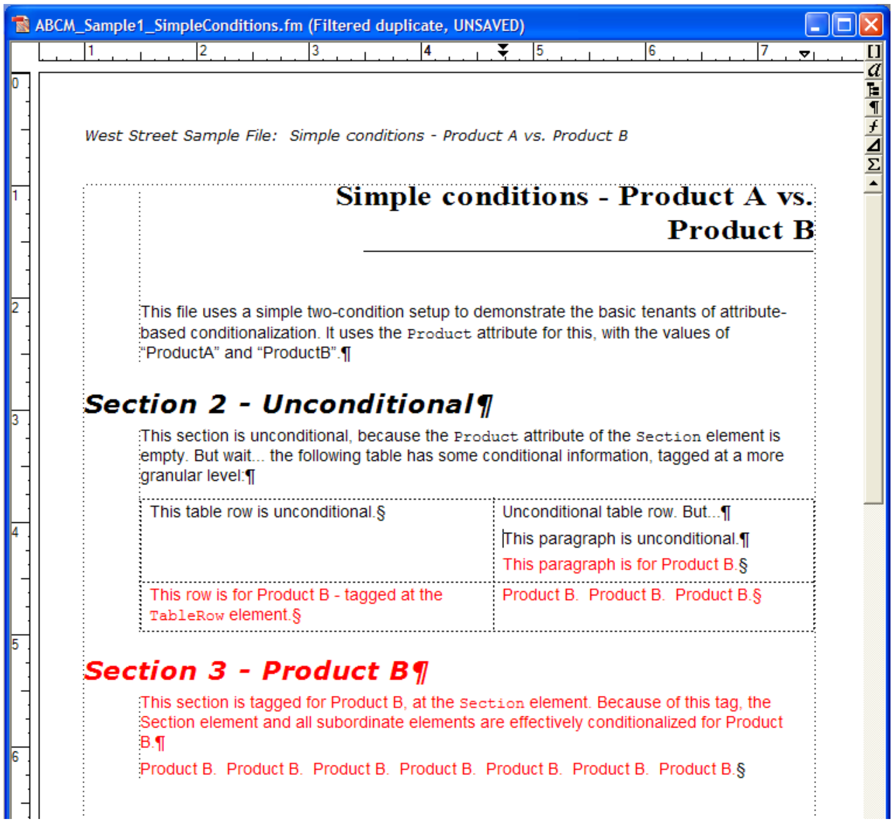
This section is unconditional, because the `Product` attribute of the `Section` element is empty. But wait... the following table has some conditional information, tagged at a more granular level.¶

§

This row is for Product A - tagged at the `TableRow` element.§	Product A. Product A. Product A.§
This table row is unconditional.§	Unconditional table row. But...¶ This paragraph is for Product A.¶ This paragraph is unconditional.§

A duplicate file has been created with all the Product B (red text) information removed. The filter process is analogous to the Show/Hide process used for native conditional text, except that all the processing logic is programmed into the scheme. You do not need to think about individual conditions when running an AXCM filter… you need only choose a scheme and go.

10. Close the duplicate, filtered file without saving changes.
11. Run the filter again on the original document, using the **Product B only** scheme this time.
12. Note the results. This time, all Product A material has been filtered out.


ABCM_Sample1_SimpleConditions.fm (Filtered duplicate, UNSAVED)

West Street Sample File: Simple conditions - Product A vs. Product B

Simple conditions - Product A vs. Product B

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty. But wait... the following table has some conditional information, tagged at a more granular level:¶

This table row is unconditional.§	Unconditional table row. But...¶ This paragraph is unconditional.¶ This paragraph is for Product B.§
This row is for Product B - tagged at the `TableRow` element.§	Product B. Product B. Product B.§

Section 3 - Product B¶

This section is tagged for Product B, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product B.¶

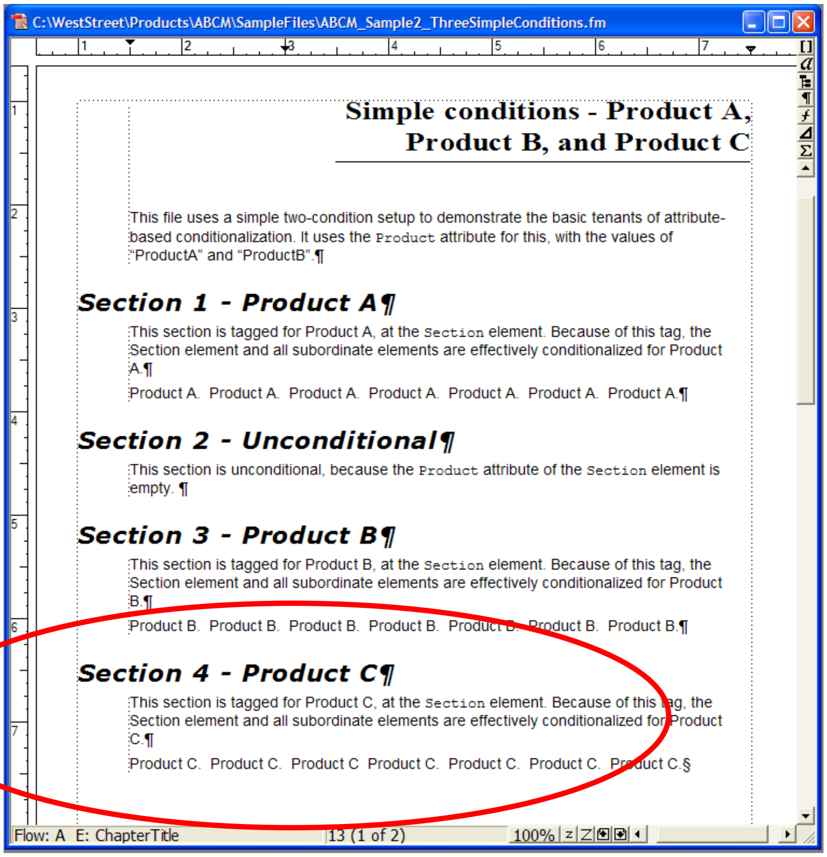
Product B. Product B. Product B. Product B. Product B. Product B. Product B.§

13. Close the filtered duplicate and the original file.

Part 3 – Introduction to scheme setup and editing

In this part, you will take a closer look at the schemes used in the previous procedure.

1. Open `AXCM_Sample2_ThreeSimpleConditions.fm`, and note that there is now a section for Product C in the document, with the `Product` attribute on the last `Section` element set to "ProductC".


C:\WestStreet\Products\ABCM\SampleFiles\ABCM_Sample2_ThreeSimpleConditions.fm
Flow: A E: ChapterTitle
13 (1 of 2)
100%

Simple conditions - Product A, Product B, and Product C

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty. ¶

Section 3 - Product B¶

This section is tagged for Product B, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product B.¶

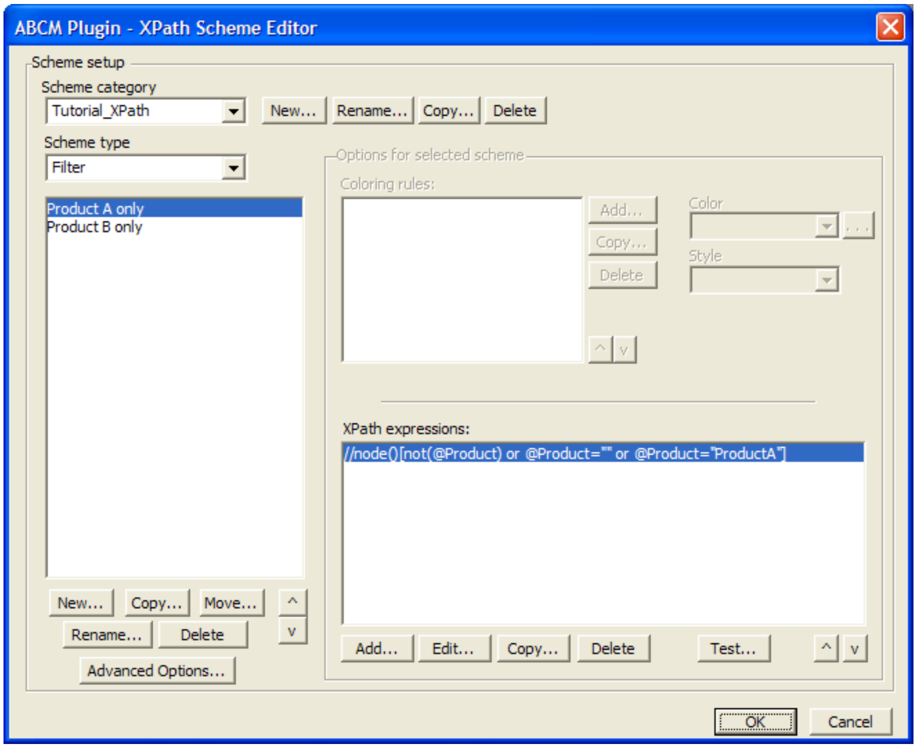
Product B. Product B. Product B. Product B. Product B. Product B. Product B.¶

Section 4 - Product C¶

This section is tagged for Product C, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product C.¶

Product C. Product C. Product C Product C. Product C. Product C. Product C.§

2. Select **AXCM > Main Settings > XPath Schemes** to open the XPath scheme editor.



Scheme setup

Scheme category: Tutorial_XPath

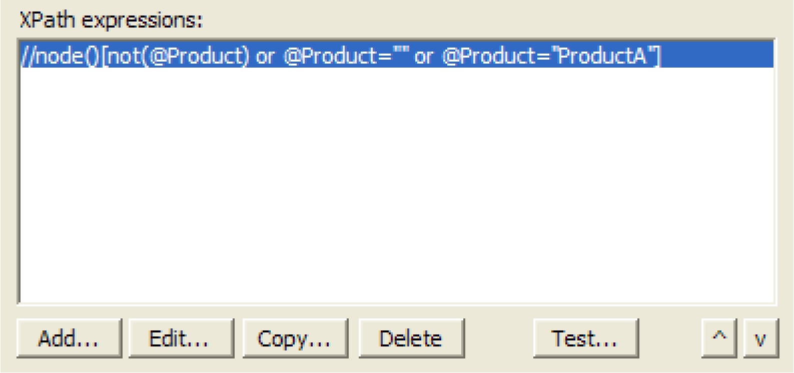
Scheme type: Filter

Product A only
Product B only

XPath expressions:

```
//node()[not(@Product) or @Product="" or @Product="ProductA"]
```

3. Under **Scheme category**, select **Tutorial_XPath**.
4. Under **Scheme type**, select **Filter**.
5. In the schemes list, select **Product A only**.
6. Take a close look at the lower right side of the dialog box, where the scheme expressions are defined. When performing a coloring or filtering action, AXCM runs these expressions against the structure tree(s) in the applicable document. An element (or perhaps text node) must be matched by all the expressions in order to be colored or to avoid being filtered out. For a more detailed explanation, consider the single expression specified for the selected filter scheme:



//node()[not(@Product) or @Product="" or @Product="ProductA"]

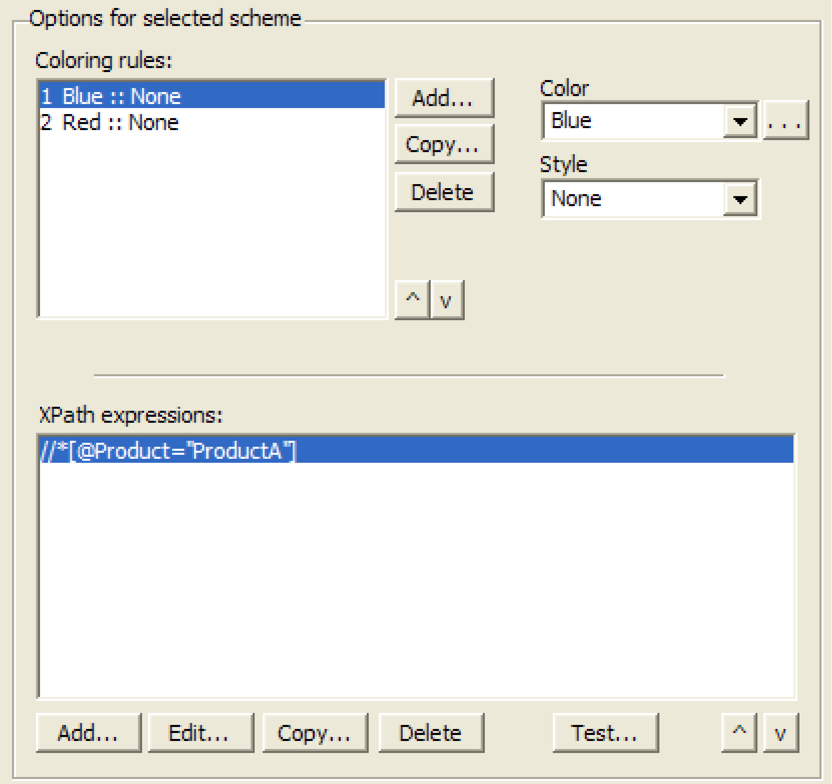
This expression:

```
//node()[not(@Product) or @Product="" or @Product="ProductA"]
```

...says literally to each node, “Filter me out UNLESS I don’t have a `Product` attribute OR my `Product` attribute is unspecified OR it is set to ‘ProductA’.” In other words, if an element has a `Product` attribute and it is unspecified (unconditional) or set to “ProductA”, it gets to stay. Otherwise, it is filtered out.

For filter schemes, keep in mind that you are specifying the content to keep, not the content to hide. If you are using attributes to denote conditions, it is common practice to use an unspecified value (or a non-existent attribute) to denote an unconditional state for any particular condition. Therefore, if you want to keep unconditional content (typical), you must account for that in your filter scheme expressions.

7. Under **Scheme type**, select **Coloring**.
8. Take a look at the **Product A vs. Product B** scheme. A coloring scheme is similar to a filter scheme, in that it is mostly just a list of expressions. The primary difference is that coloring schemes have coloring “rules,” each of which is evaluated independently with its own set of expressions. In this scheme, there are two rules, one for Blue and Red. During a coloring action, AXCM evaluates each element based on these rules. If the attribute setup for the Blue rule matches, the respective element gets colored blue. If not, it checks the Red rule in a similar fashion. If all rules are exhausted with no match, the element gets no coloring at all. Note, however, that an element may become colored when an ancestor element matches a coloring rule, even if it doesn’t match any rules itself.



Options for selected scheme

Coloring rules:

1 Blue :: None

2 Red :: None

Color: Blue

Style: None

XPath expressions:

//*[@Product="ProductA"]

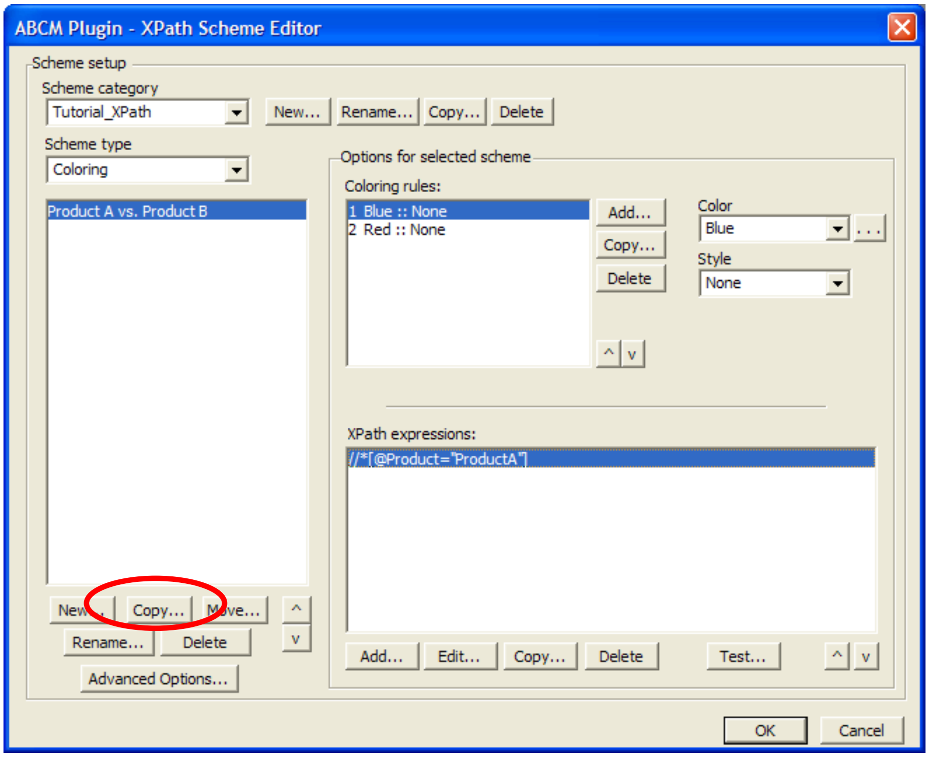
The Blue rule has the following single expression, which says literally “Match me if I have a `Product` attribute and it is set to “ProductA”:

```
//*[@Product="ProductA"]
```

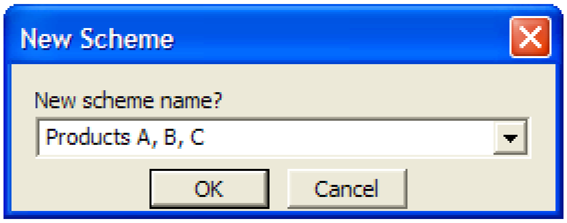
When coloring is run, AXCM steps through the structure tree and tests each element against the specified rules. The rules are processed in order and any element that matches all the expressions assigned to a rule gets that color. When a rule matches and its color is applied, AXCM moves to the next element and starts the evaluation over again. Therefore, with this scheme, any element that has a `Product` attribute set to “ProductA” will be matched by the Blue rule and receive Blue coloring. For any element, if the Blue rule does not match, AXCM will try the Red rule. If no rules match, AXCM simply moves on without applying any coloring.

Now, you will alter the scheme setup to work with the sample document you just opened, `AXCM_Sample2_ThreeSimpleConditions.fm`. More specifically, you will change the schemes to accommodate the new condition.

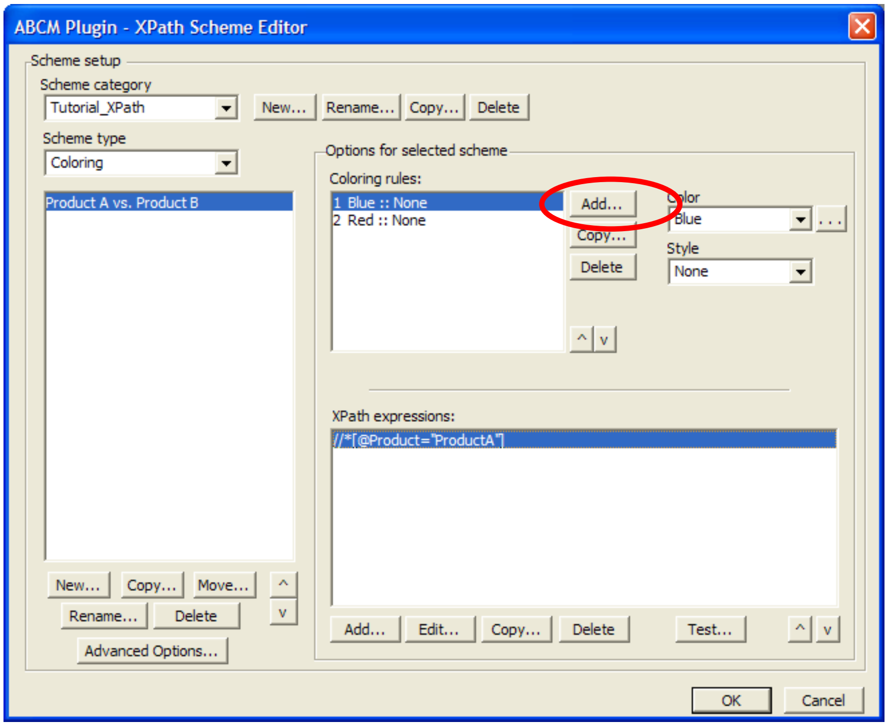
9. With the **Product A vs. Product B** coloring scheme selected, click **Copy** under the scheme list.



10. In the **New Scheme** dialog, enter a new scheme name such as **Products A, B, C**:



11. Click **OK**, and note that the new scheme appears in the list, and that it is an exact duplicate of the original scheme.
12. To the right of the list of rules, click **Add**.


ABCM Plugin - XPath Scheme Editor

Scheme setup

Scheme category: Tutorial_XPath

Scheme type: Coloring

Product A vs. Product B

Options for selected scheme

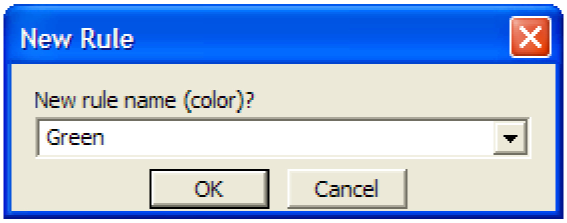
Coloring rules:
1 Blue :: None
2 Red :: None

Color: Blue

Style: None

XPath expressions:
//*[@Product="ProductA"]

13. In the **New Rule** dialog, select a new color such as **Green** and click **OK**:


New Rule

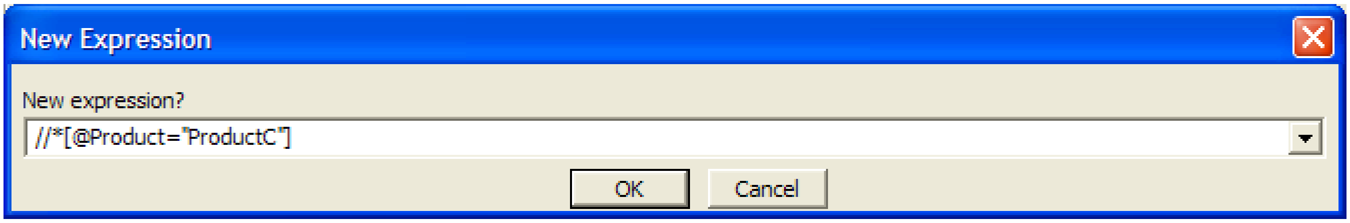
New rule name (color)? Green

You will now set up the rule to evaluate elements for Product C and color them green.

14. Under the **XPath expressions** list, click **Add**.

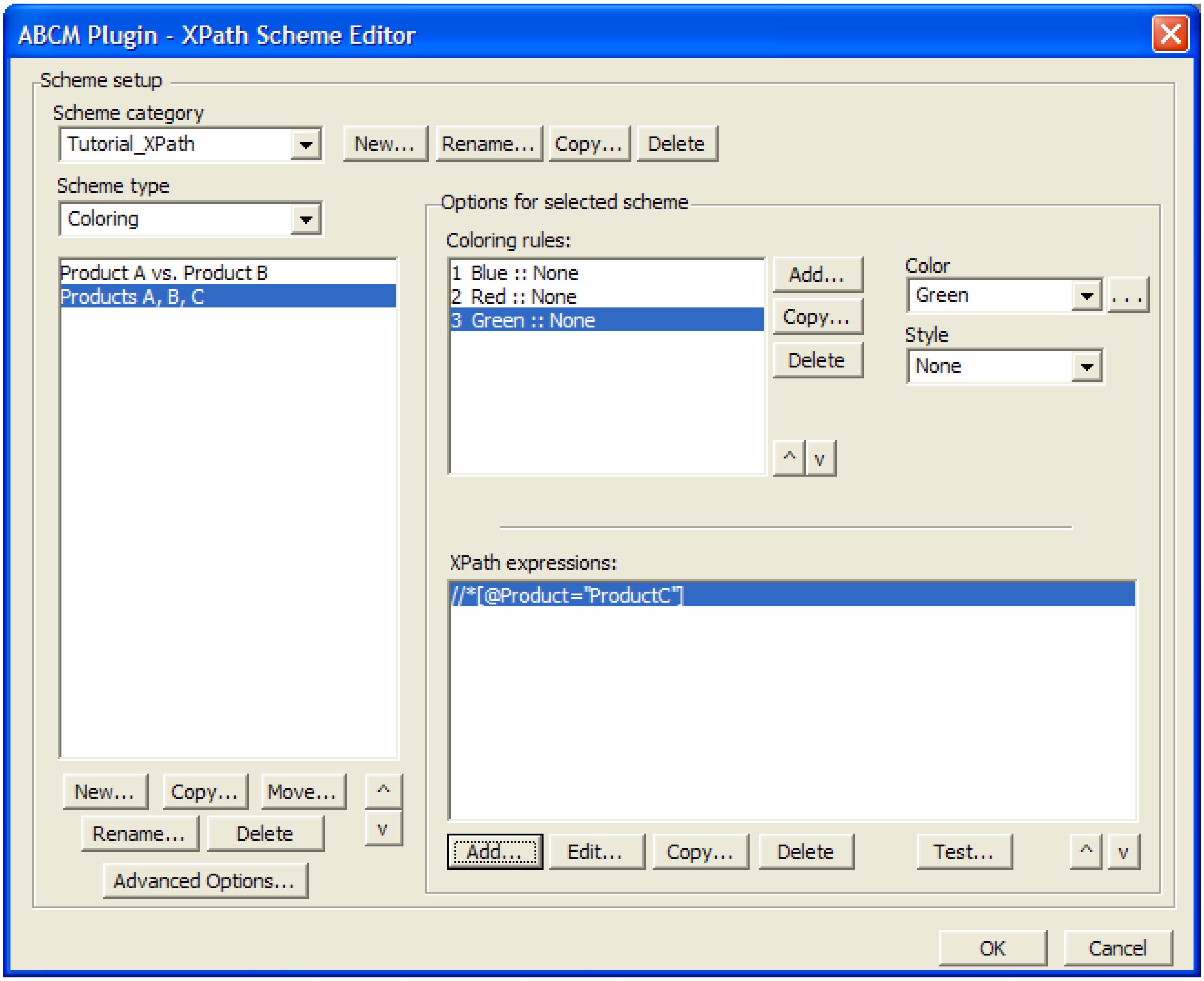
15. In the **New Expression** dialog, type the following XPath expression EXACTLY as shown below:

```
//*[@Product="ProductC"]
```


New Expression

New expression? //*[@Product="ProductC"]

16. Click **OK**. The scheme editor should now appear as follows:



ABCM Plugin - XPath Scheme Editor

Scheme setup

Scheme category: Tutorial_XPath

Scheme type: Coloring

Product A vs. Product B
Products A, B, C

Options for selected scheme

Coloring rules:
1 Blue :: None
2 Red :: None
3 Green :: None

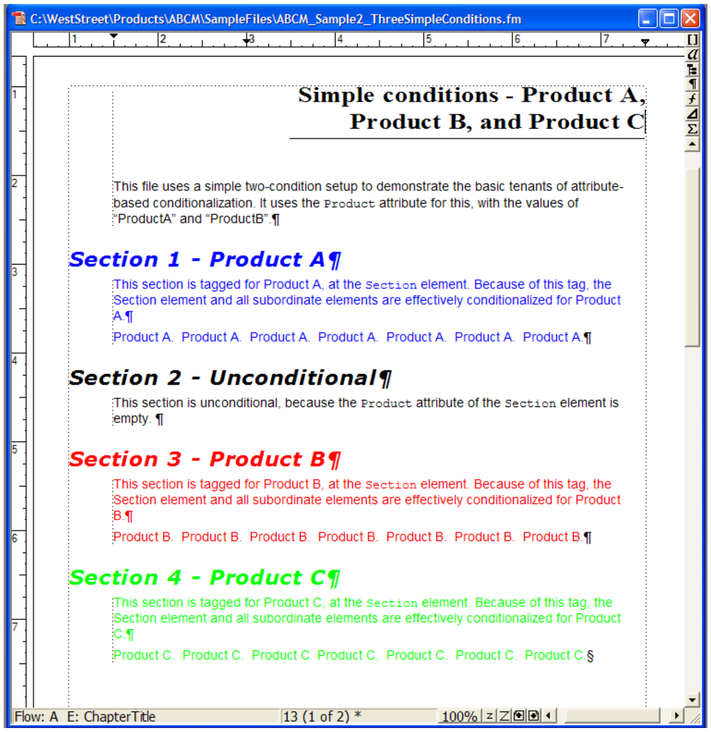
Color: Green

Style: None

XPath expressions:

```
//*[@Product="ProductC"]
```

17. Click **OK** to close the scheme editor.
18. With the three-condition sample document active, select **AXCM > Coloring > Color Document**.
19. Select the **Tutorial** category, the new **Products A, B, C** scheme, and click **OK**. Note the coloring for the Product C section.


C:\WestStreet\Products\ABCM\SampleFiles\ABCM_Sample2_ThreeSimpleConditions.fm
Flow: A E: ChapterTitle
13 (1 of 2) *
100%

Simple conditions - Product A, Product B, and Product C

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty. ¶

Section 3 - Product B¶

This section is tagged for Product B, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product B.¶

Product B. Product B. Product B. Product B. Product B. Product B. Product B.¶

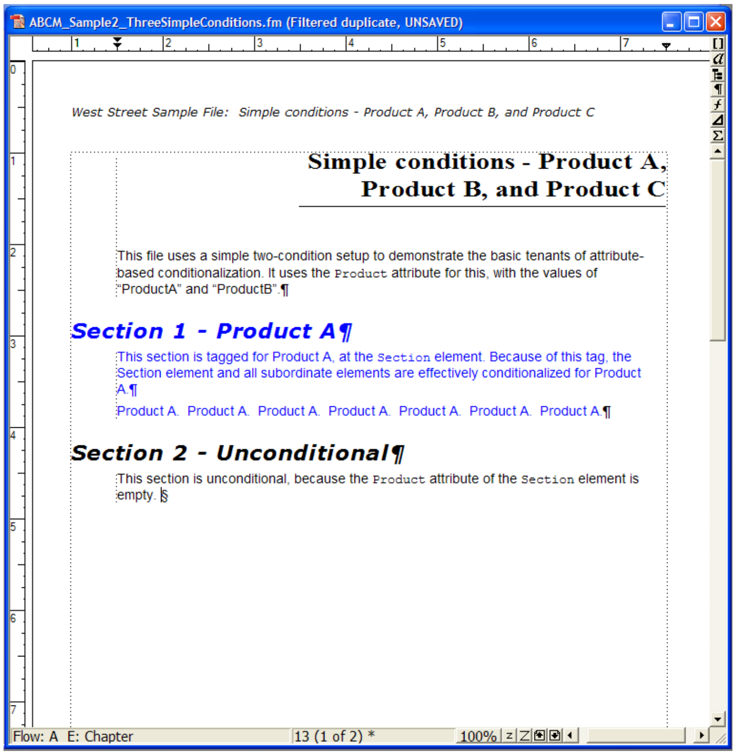
Section 4 - Product C¶

This section is tagged for Product C, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product C.¶

Product C. Product C. Product C Product C. Product C. Product C. Product C.§

Did it work? If not, recheck the syntax of the XPath expression you entered. XPath expressions must be exactly right, including case-sensitivity.

20. Select **AXCM > Filtering > Filter Document**, and filter the document with the **Product A** only scheme. Note that the Product B and Product C material is removed.


ABCM_Sample2_ThreeSimpleConditions.fm (Filtered duplicate, UNSAVED)
Flow: A E: Chapter
13 (1 of 2) *
100%

West Street Sample File: Simple conditions - Product A, Product B, and Product C

Simple conditions - Product A, Product B, and Product C

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

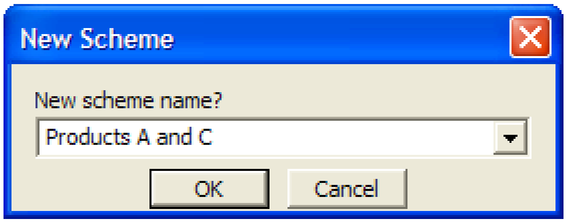
Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty. §

Also note that you did not have to change the filter scheme, even though a new condition was introduced to the document. This is a big advantage to AXCM filtering logic, versus native conditional text show/hide functionality. With a filter scheme, you are specifying the content you want to keep, and all other content is removed by default. The addition of a new condition may not affect your filtering logic at all, because the desired conditions in the output are still the same.

21. Close the new, filtered duplicate.
22. Open the XPath scheme editor again (**AXCM > Main Settings > XPath Schemes**)
23. Under **Scheme type**, select **Filter**.
24. Select the **Product A only** scheme.
25. Under the **Schemes** list, click **Copy** and give the new scheme the name **Products A and C**.



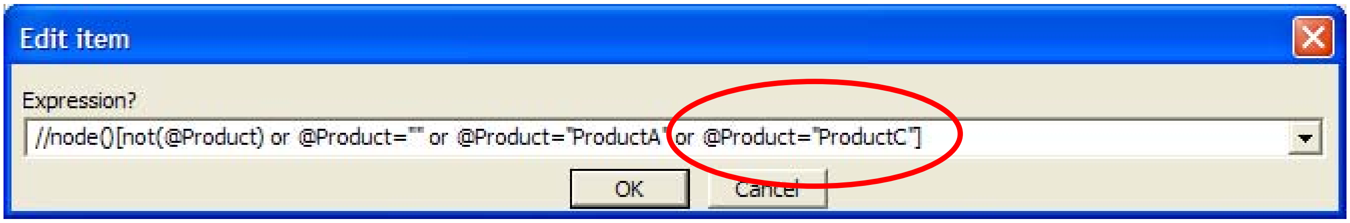
New scheme name? Products A and C

26. Click **OK** in the **New Scheme** dialog.
27. With the new scheme selected and the first XPath expression selected in the list, click **Copy** under the **XPath expressions** list.

    You are now preparing to alter this scheme to produce a composite output for both Product A and Product C. For convenience, you will use the expression copy function, versus the error-prone process of manually typing new expressions.

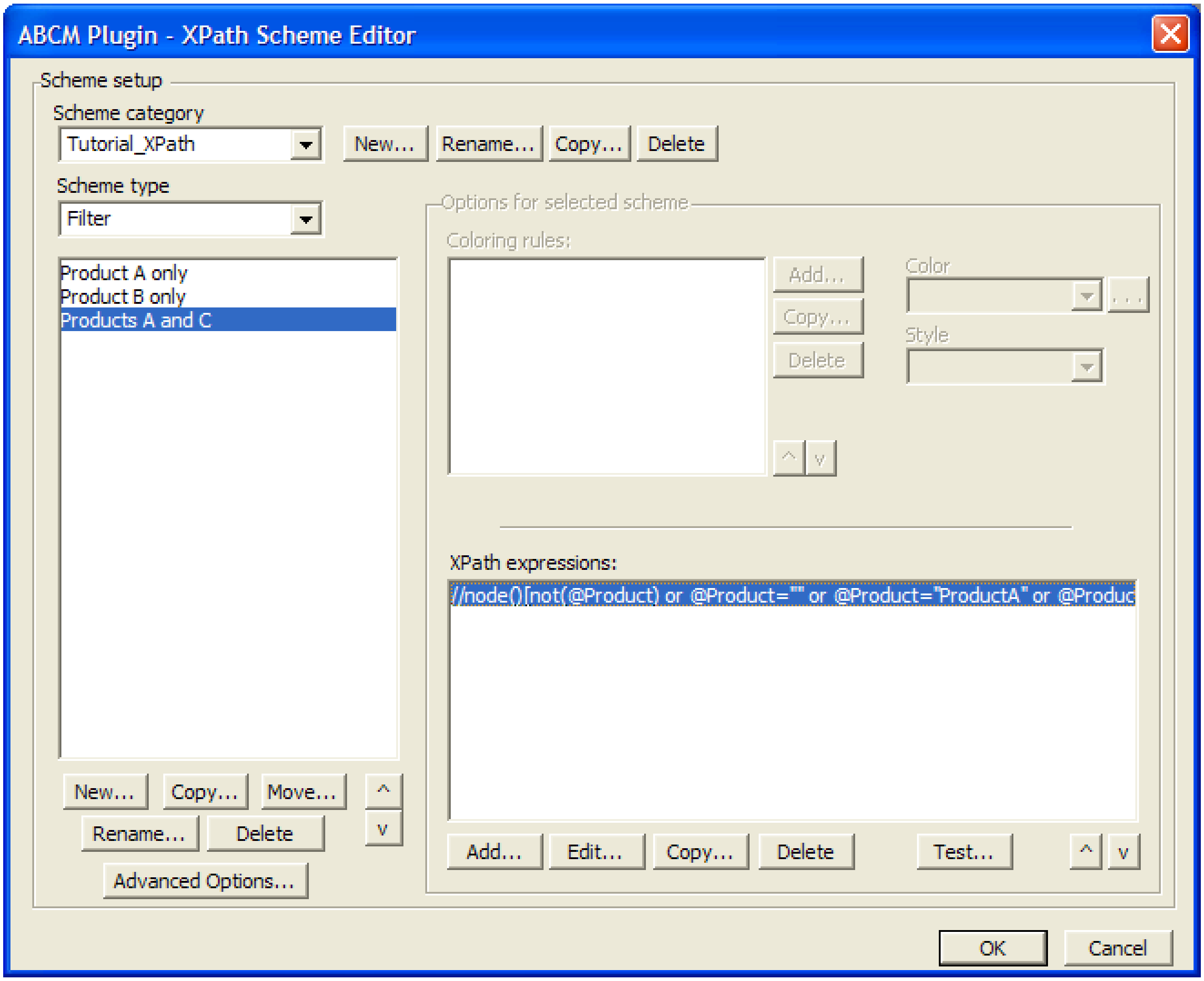
28. In the **New Expression** dialog, add ‘or @Product="ProductC"’ to the expression so it appears EXACTLY as below:

```
//node()[not(@Product) or @Product="" or @Product="ProductA" or @Product="ProductC"]
```



Expression? //node()[not(@Product) or @Product="" or @Product="ProductA" or @Product="ProductC"]

29. Click **OK** in the **New Expression** dialog. The scheme editor should appear as follows:



Scheme category: Tutorial_XPath

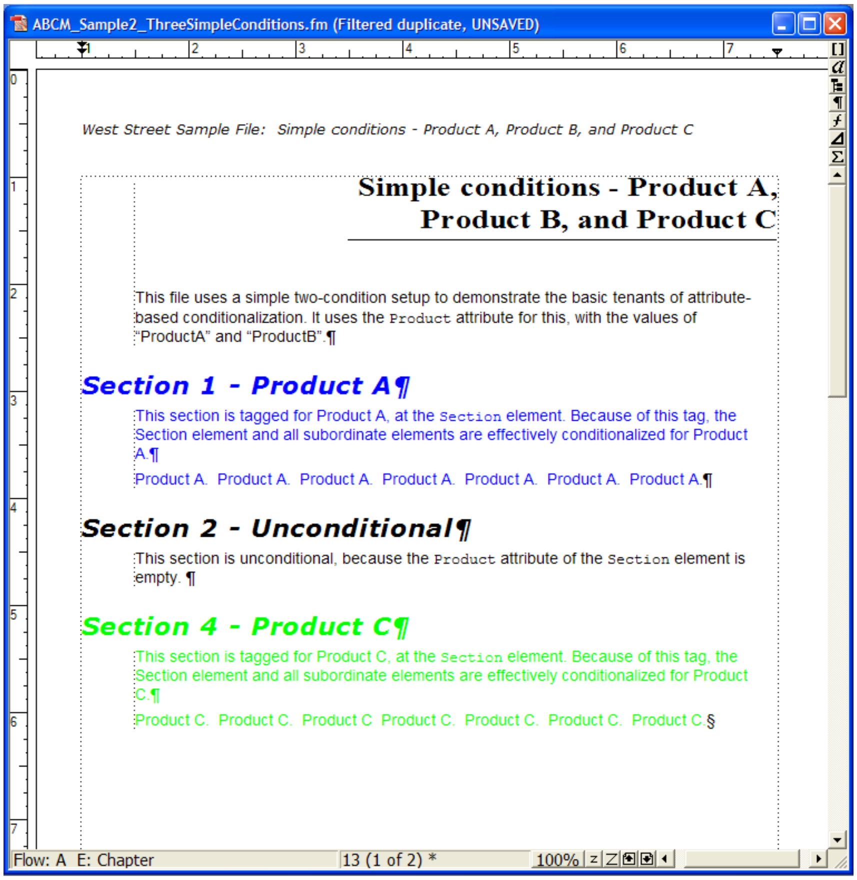
Scheme type: Filter

Product A only
Product B only
Products A and C

XPath expressions:

```
//node()[not(@Product) or @Product="" or @Product="ProductA" or @Produc
```

30. Click **OK** to close the scheme editor.
31. Filter the original document with the new scheme and note that the output contains Product A and Product C material.


ABCM_Sample2_ThreeSimpleConditions.fm (Filtered duplicate, UNSAVED)
Flow: A E: Chapter
13 (1 of 2) *
100%

West Street Sample File: Simple conditions - Product A, Product B, and Product C

Simple conditions - Product A, Product B, and Product C

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty. ¶

Section 4 - Product C¶

This section is tagged for Product C, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product C.¶

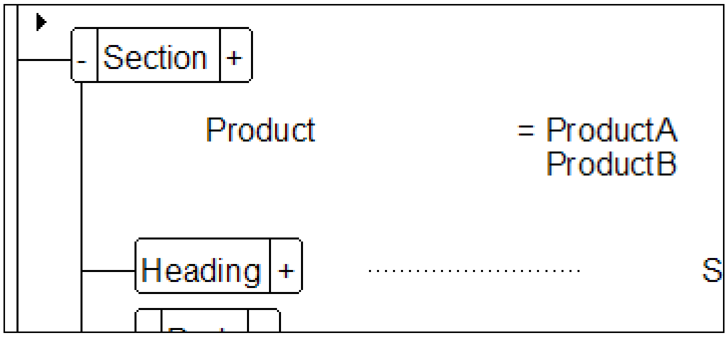
Product C. Product C. Product C Product C. Product C. Product C. Product C §

32. Close the filtered duplicate and the original document.

Part 4 – Overlapping conditions and coloring rule order

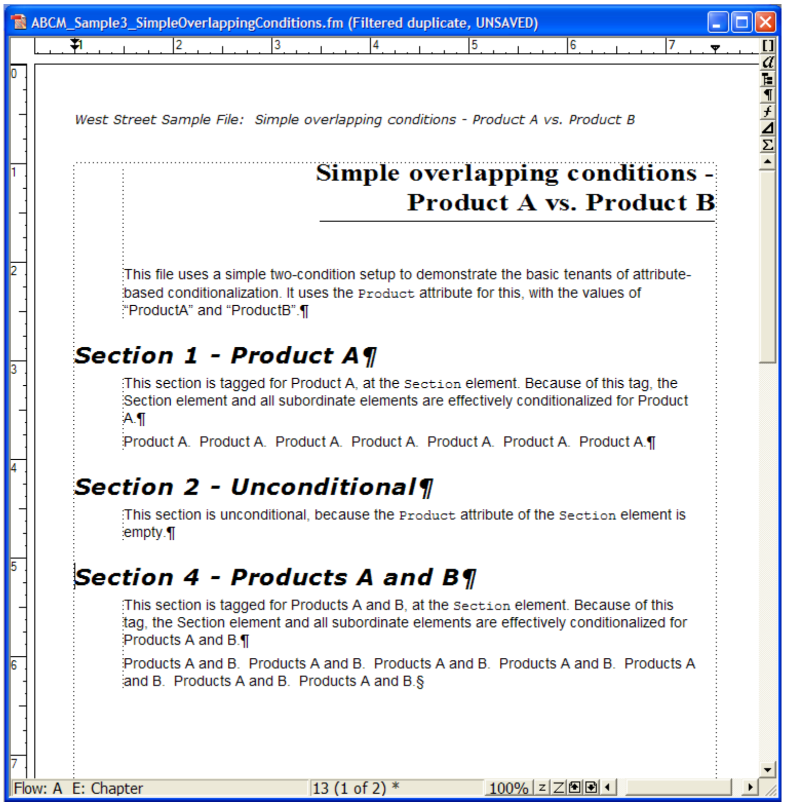
In a structured document, an element can have multiple attributes, each of which may be configured to contain multiple values (as allowed by the EDD). When multiple attributes and/or values are used to denote conditions, this is equivalent to the concept of overlapping conditions. This situation, however, is much easier to handle when using structural markup to denote conditions, rather than native conditional text tags.

1. Open `AXCM_Sample3_SimpleOverlappingConditions.fm`, and note that there is now a section for both Product A and Product B, with the `Product` attribute on the `Section` element set to "ProductA" and "ProductB".



Product = ProductA
ProductB

2. Filter the document (**AXCM > Filtering > Filter Document**) using the **Product A only** and the **Product B only** schemes, and note that in both cases the composite paragraph remains.


ABCM_Sample3_SimpleOverlappingConditions.fm (Filtered duplicate, UNSAVED)
Flow: A E: Chapter
13 (1 of 2) *
100%

West Street Sample File: Simple overlapping conditions - Product A vs. Product B

Simple overlapping conditions - Product A vs. Product B

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

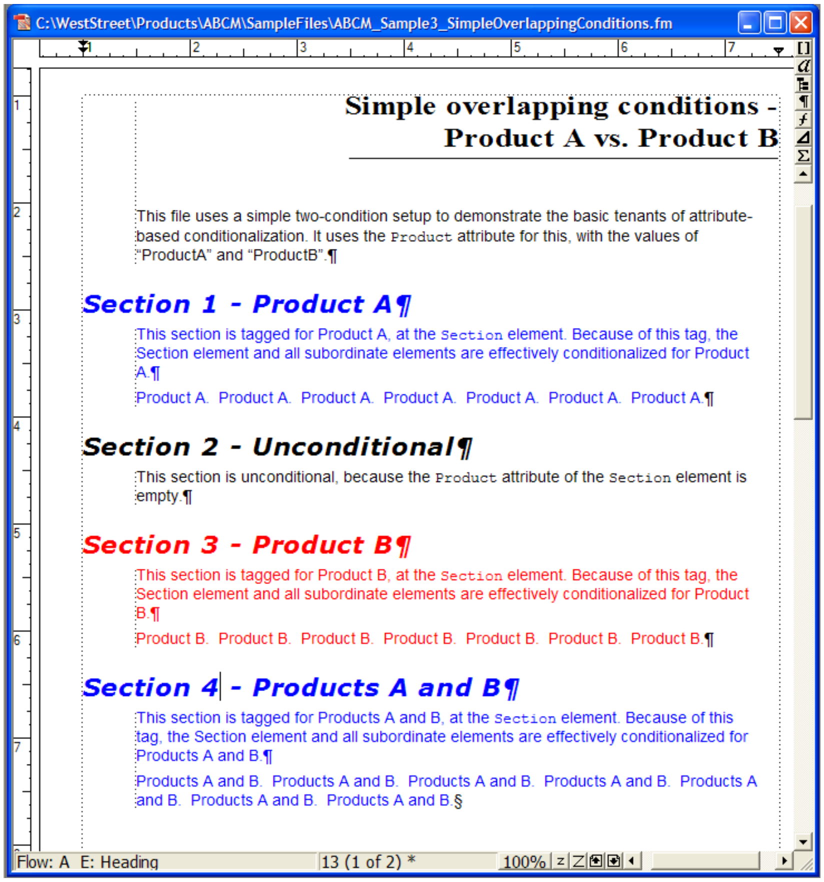
This section is unconditional, because the `Product` attribute of the `Section` element is empty.¶

Section 4 - Products A and B¶

This section is tagged for Products A and B, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Products A and B.¶

Products A and B. Products A and B. Products A and B. Products A and B. Products A and B. Products A and B. Products A and B.§

3. Close any filtered duplicates you created.
4. Color the document with the **Product A vs. Product B** scheme and note the results (**AXCM > Coloring > Color Document**).


C:\WestStreet\Products\ABCM\SampleFiles\ABCM_Sample3_SimpleOverlappingConditions.fm
Flow: A E: Heading
13 (1 of 2) *
100%

Simple overlapping conditions -
Product A vs. Product B

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of "ProductA" and "ProductB".¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty.¶

Section 3 - Product B¶

This section is tagged for Product B, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product B.¶

Product B. Product B. Product B. Product B. Product B. Product B. Product B.¶

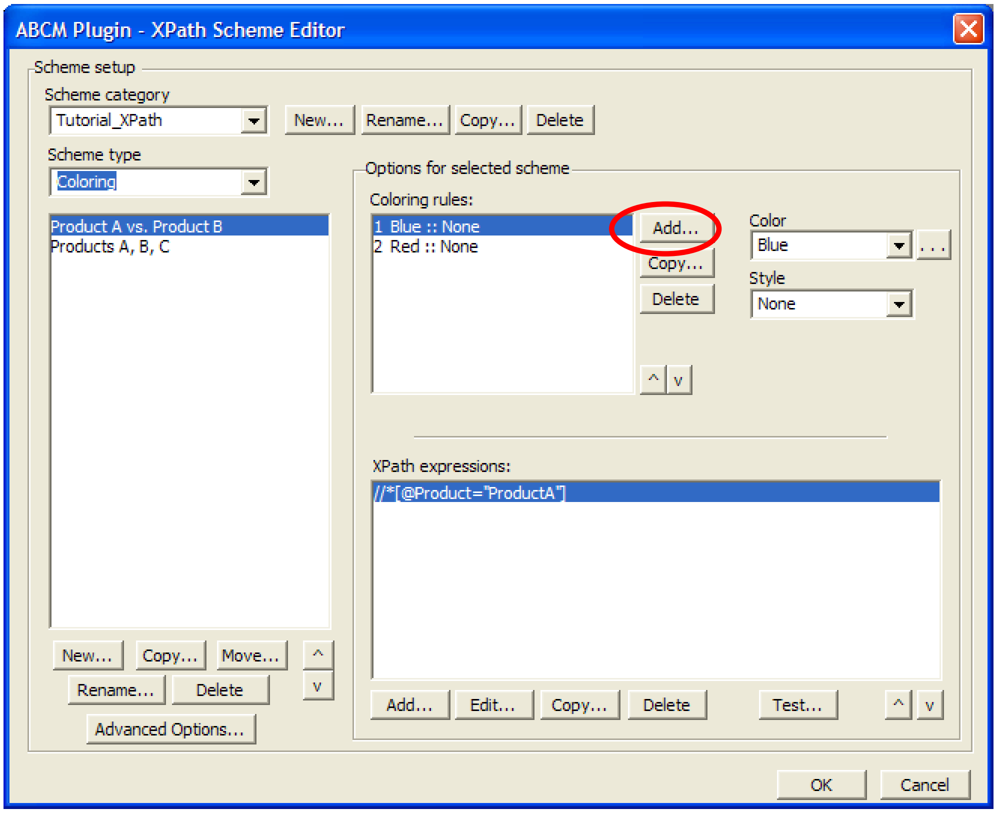
Section 4 - Products A and B¶

This section is tagged for Products A and B, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Products A and B.¶

Products A and B. Products A and B. Products A and B. Products A and B. Products A and B. Products A and B. Products A and B.§

The composite section received blue, the color for Product A. This happened because the first rule in the scheme matched the composite section, because its XPath expression matched the `Section` element. Currently, there is no rule in the scheme designed specifically to match a composite Product A and B section, so AXCM simply used the first matching rule it found. In the following steps, you will change the scheme to include a rule for the composite material.

5. Open the XPath scheme editor (**AXCM > Main Settings > XPath Schemes**).
6. Select the **Tutorial_XPath** category, the **Coloring** scheme type, and the **Product A vs. Product B** scheme.
7. To the right of the coloring rules list, click **Add**.



Scheme category: Tutorial_XPath

Scheme type: Coloring

Product A vs. Product B
Products A, B, C

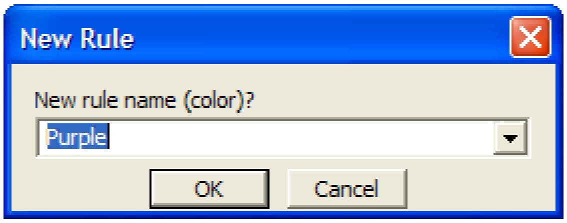
Coloring rules:
1 Blue :: None
2 Red :: None

Color: Blue

Style: None

XPath expressions:
//*[@Product="ProductA"]

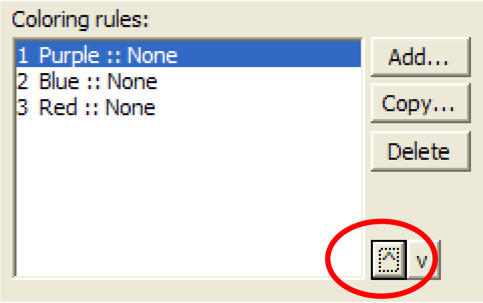
8. In the **New Rule** dialog, select a new color such as **Purple**.



New rule name (color)? Purple

9. Use the “^” button under the rules to slide the new Purple rule to the top of the list.

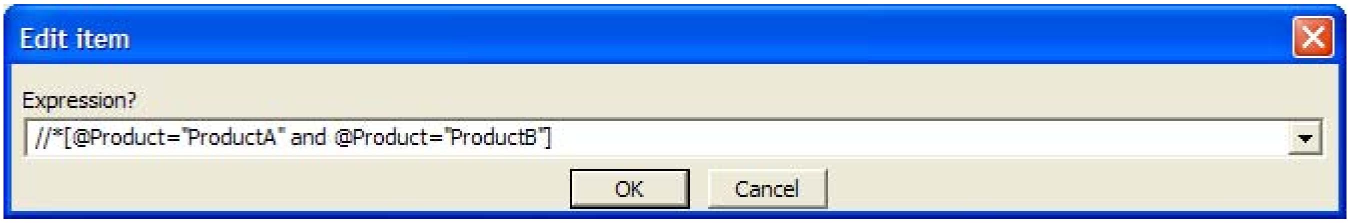
NOTE: This is a very important step, because rule order can have a significant impact on the behavior of a coloring scheme. For more explanation on this subject, see your *User Guide*.



Coloring rules:
1 Purple :: None
2 Blue :: None
3 Red :: None

10. Under the **XPath expressions** list, click **Add** and enter the following expression exactly as follows:

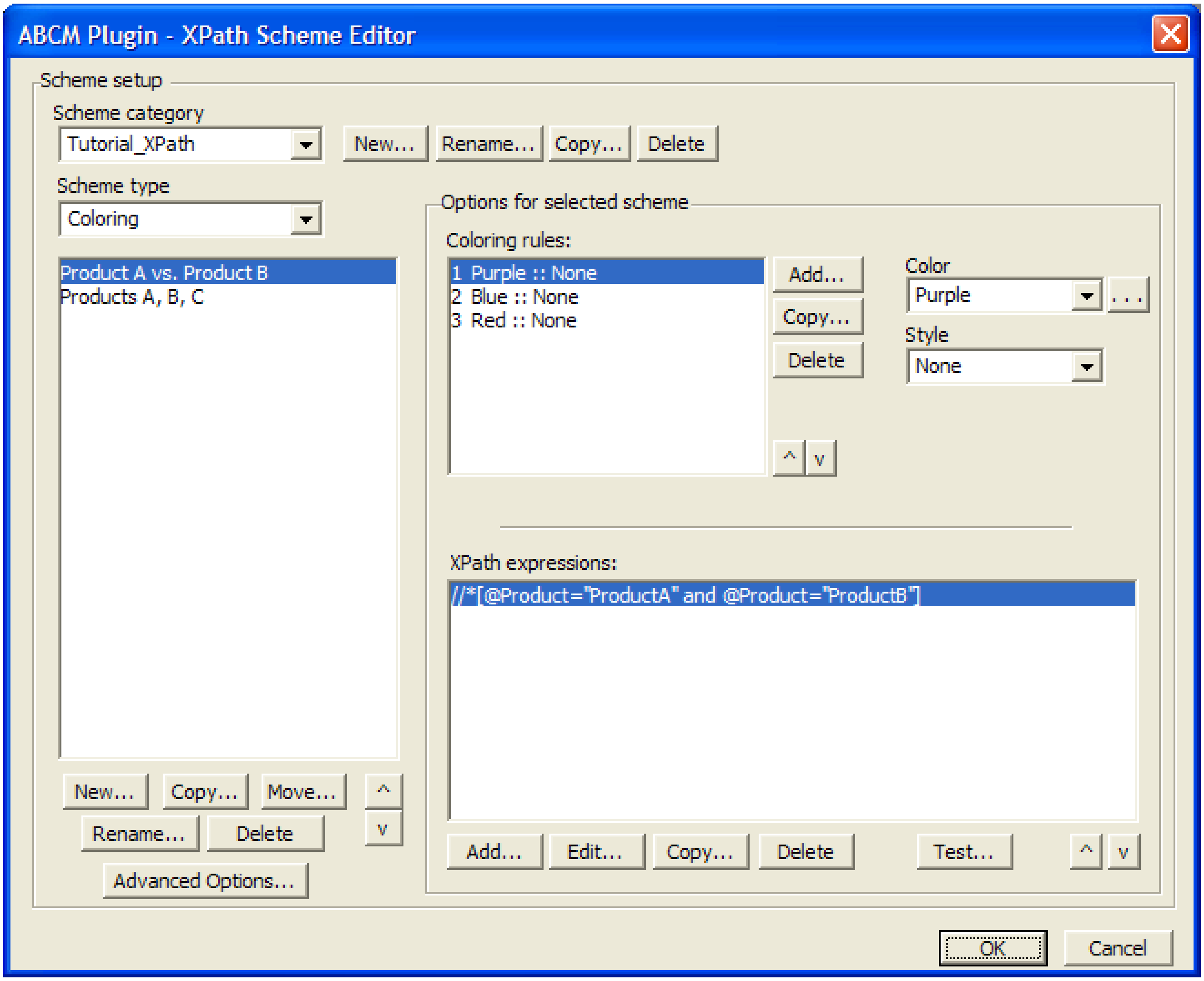
```
//*[@Product="ProductA" and @Product="ProductB"]
```



Expression?
`//*[@Product="ProductA" and @Product="ProductB"]`

This expression says literally, “Match me if I have a `Product` attribute set to “ProductA” and “ProductB”, such as the `Section` element that is the current focus. Any element with this setup would also be matched by the Blue or Red rule, hence the importance of putting the Purple rule at the top to be evaluated first.

11. Ensure that the scheme editor appears as follows, and click **OK**.



Scheme category: Tutorial_XPath

Scheme type: Coloring

Product A vs. Product B
Products A, B, C

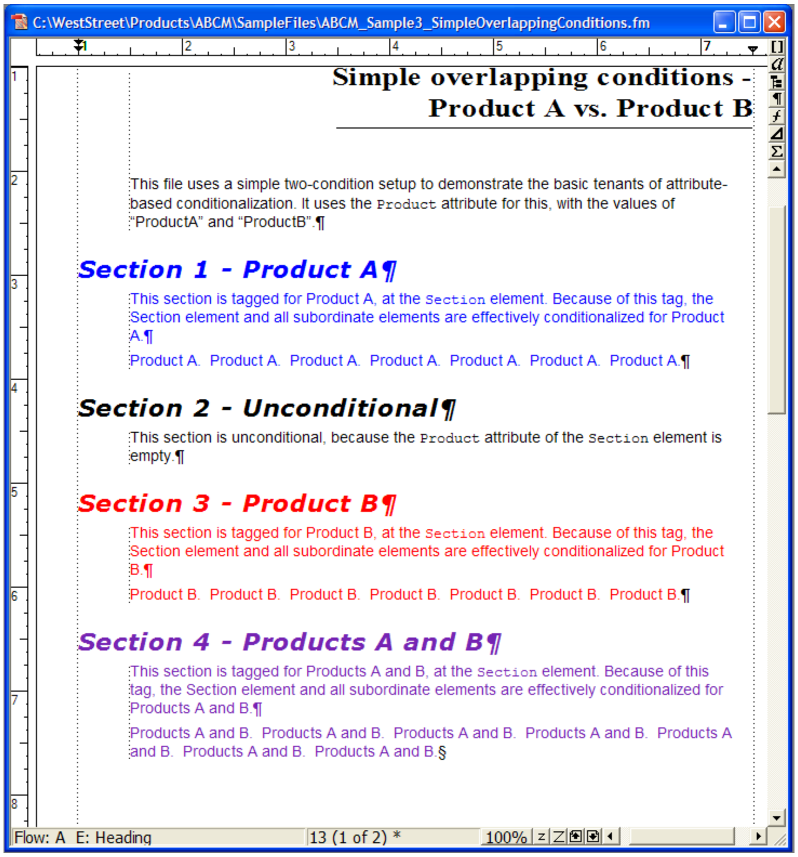
Coloring rules:
1 Purple :: None
2 Blue :: None
3 Red :: None

Color: Purple

Style: None

XPath expressions:
`//*[@Product="ProductA" and @Product="ProductB"]`

12. Color the document again (**AXCM > Coloring > Color Document**) with the **Product A vs. Product B** scheme and note the results.


C:\WestStreet\Products\ABCM\SampleFiles\ABCM_Sample3_SimpleOverlappingConditions.fm
Flow: A E: Heading
13 (1 of 2) *
100%

Simple overlapping conditions - Product A vs. Product B

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of “ProductA” and “ProductB”.¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty.¶

Section 3 - Product B¶

This section is tagged for Product B, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product B.¶

Product B. Product B. Product B. Product B. Product B. Product B. Product B.¶

Section 4 - Products A and B¶

This section is tagged for Products A and B, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Products A and B.¶

Products A and B. Products A and B. Products A and B. Products A and B. Products A and B. Products A and B. Products A and B.§

NOTE: All AXCM coloring with regards to overlapping conditions operates in this fashion. AXCM colors exactly what the respective scheme indicates, and nothing else. No more magenta text or automatically-generated composite colors, unless your scheme specifically says so!

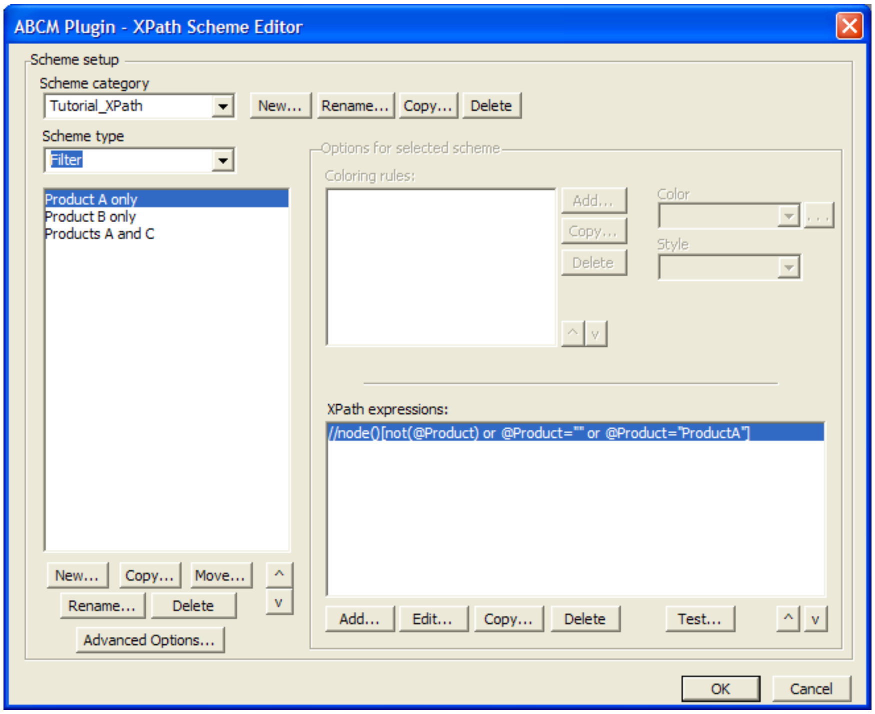
13. Close the sample document.

Part 5 – Using element tags and multiple expressions

A significant advantage to using XPath schemes (versus classic schemes) is the flexibility to use markup characteristics other than attributes to denote and manage conditions. For example, you can use element tags, hierarchy, relationships, and element text to indicate conditions, all of which are supported by the XPath syntax for coloring and filtering actions.

In this section, you will enhance one of the sample schemes to recognize and filter out an element based on the element tag, regardless of any attributes it may have. As implied above, the flexibility extends far beyond this capability, the extent of which depends heavily on your skills with the XPath syntax.

1. Open `AXCM_Sample4_AuthoringComments.fm`, and note the `Comment` elements. These are intended as a tool for authors to add personal comments to the documentation, something which is very important for internal purposes but should likely never appear in a published output. Therefore, a filter scheme would commonly be designed to remove all of these element by name, rather than relying on attribute markup.
2. Open the XPath scheme editor (**AXCM > Main Settings > XPath Schemes**).
3. Select the **Tutorial_XPath** category, the **Filter** scheme type, and the **Product A only** scheme.


ABCM Plugin - XPath Scheme Editor
Scheme setup
New... Rename... Copy... Delete
Options for selected scheme
Coloring rules:
Add... Copy... Delete
Color
Style
New... Copy... Move...
Rename... Delete
Advanced Options...
Add... Edit... Copy... Delete Test...
OK Cancel

Scheme category: Tutorial_XPath

Scheme type: Filter

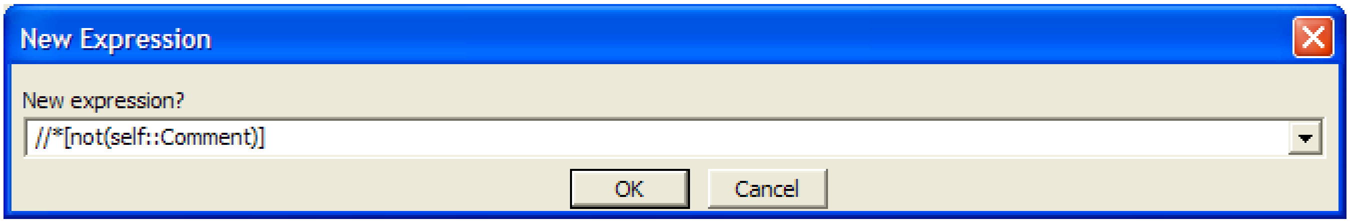
Product A only
Product B only
Products A and C

XPath expressions:

```
//node()[not(@Product) or @Product="" or @Product="ProductA"]
```

4. Under the **XPath expressions** list, click **Add** and enter the following expression exactly as shown:

```
//*[not(self::Comment)]
```



New expression? //*[not(self::Comment)]

This expression says literally, “Match me if I am not a `Comment` element”. Take a moment to consider the current setup, which includes the following expressions:

```
//node()[not(@Product) or @Product="" or @Product="ProductA"]
//*[not(self::Comment)]
```

Keeping in mind that a filter scheme specifies the content to keep and that an element must match all expressions to be kept (that is, to avoid being filtered out), this scheme setup effectively says literally,

“Match me if I have no `Product` attribute, an empty `Product` attribute, or a `Product` attribute set to “ProductA”

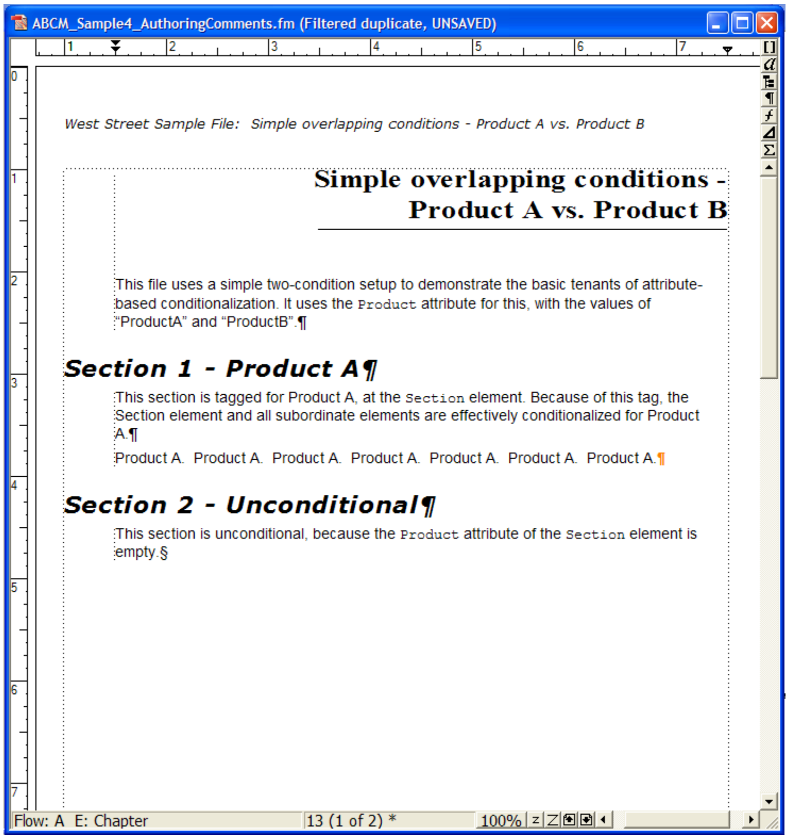
-and-

“I am not a `Comment` element”

Otherwise, an element would be filtered out. A key point to remember is that the plugin uses an "AND" logic between multiple expressions.

NOTE: As you may be noticing, XPath schemes can involve a considerable level of complexity, especially when multiple expressions are required. To assist with the generation of expressions, the scheme editor includes a **Test** button which serves as a basic XPath query tool for testing your expressions. Along with this utility, you may find that copying and modifying known-effective expressions are the most efficient means for scheme management. In many cases, the same basic constructs tend to repeat themselves frequently.

5. Click **OK** in the scheme editor.
6. Filter the document with the **Product A only** scheme and note the results.


ABCM_Sample4_AuthoringComments.fm (Filtered duplicate, UNSAVED)
Flow: A E: Chapter
13 (1 of 2) *
100%

West Street Sample File: Simple overlapping conditions - Product A vs. Product B

Simple overlapping conditions - Product A vs. Product B

This file uses a simple two-condition setup to demonstrate the basic tenants of attribute-based conditionalization. It uses the `Product` attribute for this, with the values of “ProductA” and “ProductB”.¶

Section 1 - Product A¶

This section is tagged for Product A, at the `Section` element. Because of this tag, the Section element and all subordinate elements are effectively conditionalized for Product A.¶

Product A. Product A. Product A. Product A. Product A. Product A. Product A.¶

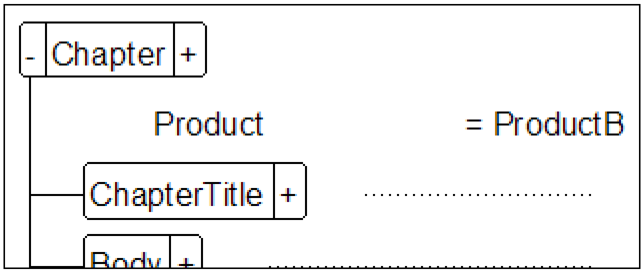
Section 2 - Unconditional¶

This section is unconditional, because the `Product` attribute of the `Section` element is empty.§

Part 6 – Filtering a book

Book filtering is similar to document filtering, except that multiple files are processed together. One important feature of book filtering is the ability to conditionalize an entire chapter of a book and remove it during a filter process. This feature will be demonstrated in the following steps, used in conjunction with a duplicate file book filter.

1. Open `AXCM_SampleBook.book`, and all its chapter files.
2. Take a look at `AXCM_Sample5_ProductB_WholeDoc.fm`, and note how it is conditionalized at the highest-level element for Product B.

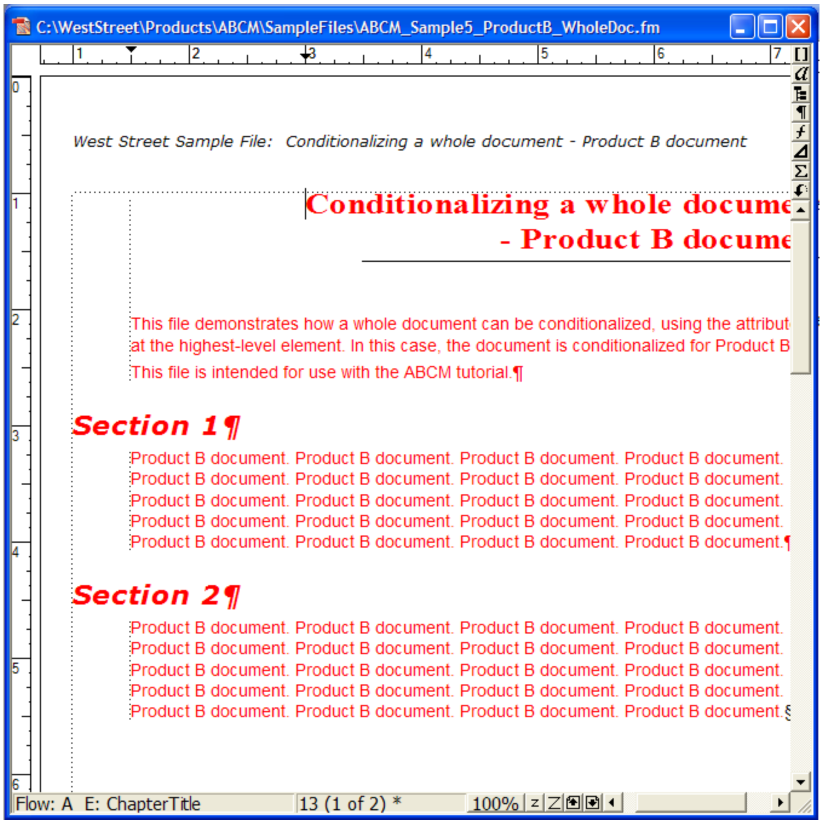


Product = ProductB

3. Bring the book window back to the front.

4. Select **AXCM > Coloring > Color Book**.
5. Color the book with the **Product A vs. Product B scheme**, under the **Tutorial_XPath** category, and note the results.

   All three files are colored, and the Product B file is colored entirely for the Product B color.



C:\WestStreet\Products\ABCM\SampleFiles\ABCM_Sample5_ProductB_WholeDoc.fm

West Street Sample File: Conditionalizing a whole document - Product B document

Conditionalizing a whole docume

- Product B docume

This file demonstrates how a whole document can be conditionalized, using the attribut at the highest-level element. In this case, the document is conditionalized for Product B This file is intended for use with the ABCM tutorial.¶

Section 1¶

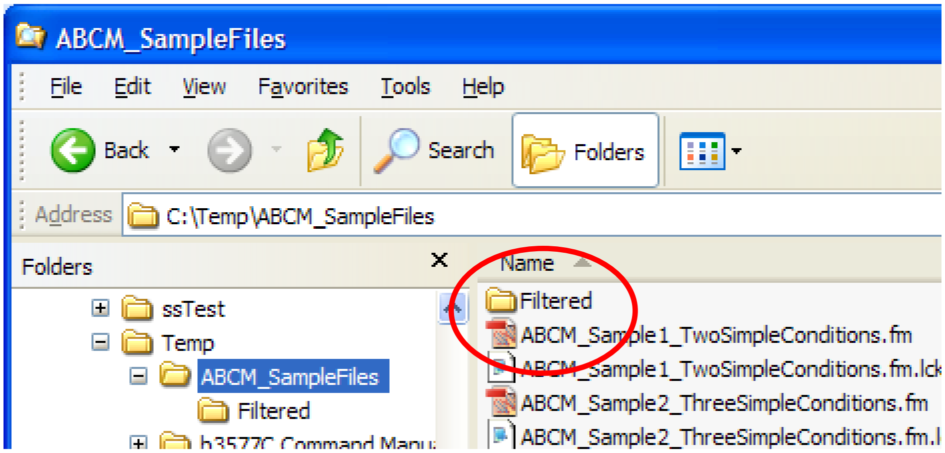
Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document.¶

Section 2¶

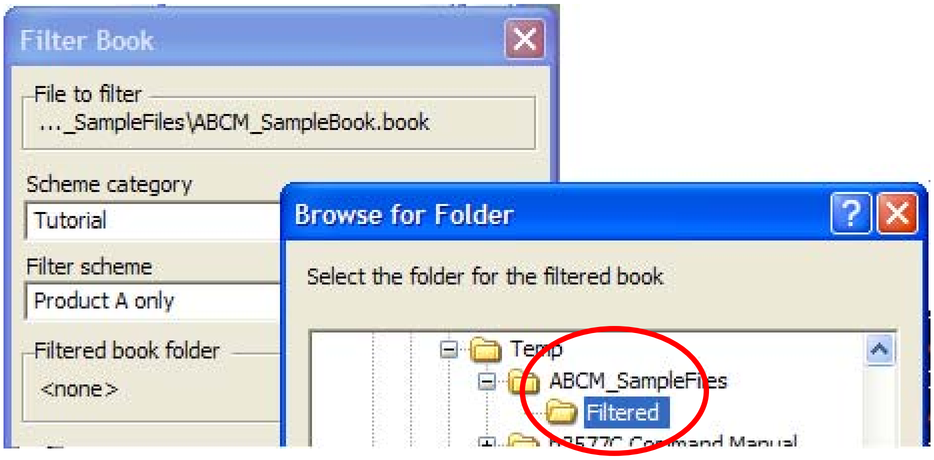
Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document. Product B document.§

Flow: A E: ChapterTitle 13 (1 of 2) * 100%

6. Using Windows Explorer or some other file management utility, create a subfolder called `Filtered` in your sample files area.



7. Back in FrameMaker, bring the book window to the front.
8. Select **AXCM > Filtering > Filter Book**.
9. Select the **Tutorial** category and the **Product A only** scheme.
10. Under **Filtered book folder**, click the "**. . .**" button and browse to the `Filtered` subfolder you just created.



File to filter
..._SampleFiles\ABCM_SampleBook.book

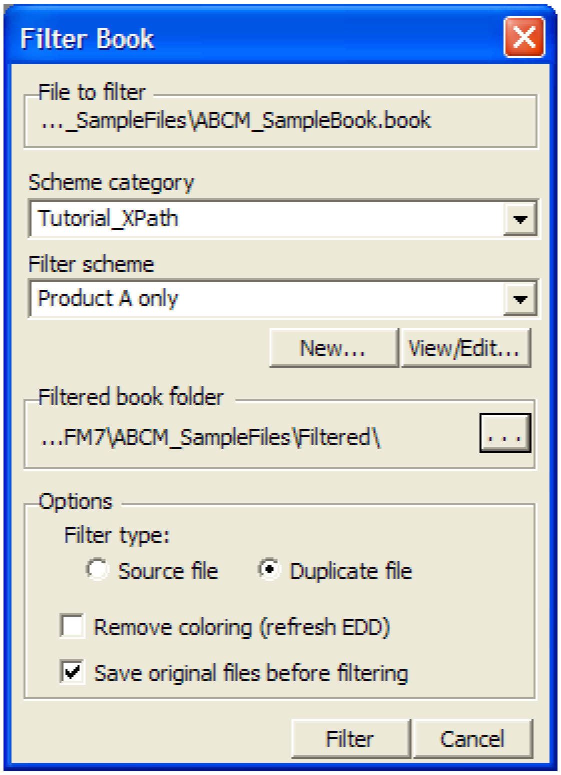
Scheme category
Tutorial

Filter scheme
Product A only

Filtered book folder
<none>

Select the folder for the filtered book

11. Ensure the **Filter** dialog now looks something like the following:



File to filter
..._SampleFiles\ABCM_SampleBook.book

Scheme category
Tutorial_XPath

Filter scheme
Product A only

Filtered book folder
...FM7\ABCM_SampleFiles\Filtered\

Options
Filter type:
○ Source file ● Duplicate file
☐ Remove coloring (refresh EDD)
☑ Save original files before filtering

12. Click **Filter** and note the results. All files in the book are filtered, and the Product B chapter has been completely filtered out.
13. Close all the sample files.

Part 8 – Finishing up and looking ahead

This tutorial introduced some of the fundamentals concerning AXCM. If you intend to incorporate the software into your workflow, the following are some things to keep in mind:

- In this tutorial, all conditionalization was done at the paragraph level or higher, for simplicity. That is, conditional attributes were set on paragraph-containing or higher-level attributes only. In practice, you can put conditional attributes on any element, including text spans, markers, table components, and variables, and the software will treat them the same way. AXCM cares about markup only… it doesn't care what the elements themselves contain.

- In this tutorial, all the schemes dealt with one or two expressions. Remember that schemes can address any number of expressions, as necessary to achieve the desired result.
- The software has many facets to the rule base used for filtering and coloring. This tutorial covered only a small subset of common features. If you plan to use AXCM for production work, consider giving the *User Guide* some time, such that you can discover ways to make the software work better for you.

Thank you for taking the time to check out AXCM. We hope that it can be of some use to you.