Chapter 1. Active content

When you publish to electronic media, you can create active content, that is, content that has behavior as well as formatting. Some examples:

Dynamic arrangement	Part of the presentation algorithm is arranging content on the page or screen, but with online media you can allow the reader to arrange the content. For instance, you can publish tables that readers can sort for themselves.
Adaptive content	Similarly, you can create content that adapts itself dynamically to the view port in which it is displayed. For instance, displaying in multiple columns on a wide view port, and in a single column on a narrow one.
Progressive disclosure	You can present content in a way that only shows part of the content on the screen initially but reveals more when the user clicks on a link or takes another action. For instance, you might show the high-level of a procedure and provide a link that opens detailed steps for those who need them. This is a way to cater to audiences with different levels of preparedness.
Transclusion	You can pull in content from another source.
Feeds and dynamic sources	You can include content that comes from an external source which updates independently of your content such as a feed or a web service.
Interactive media	You can include graphics and other media that the user can interact with.

As always, you can represent these behaviors in each of the structured writing domains.

- In the media domain, you encode the behavior itself. This means that JavaScript or other forms of executable code may become part of your media domain content.
- In the document domain, you encode an abstract representation of these behaviors or of the document structures on which these behaviors act. In the document domain, the executable code that implements the behavior is factored out, but the document structures that support the behavior or depend on it are made explicit. The document domain, in other words, may call for the behavior, but it does not contain the implementation of the behavior.
- In the subject domain, as always, you record information about the subject matter that may be used to make a decision to implement certain behavior. The subject domain markup neither calls for not implements the behavior. You may, however, choose which parts of the subject matter to markup us explicitly based on the behavior you want to implement.

Supporting active content in structured writing comes down to one thing. If you create static document structures, you will have static content. If you create structures that an algorithm can interpret in different ways, you have the foundation for active content.

In the media domain this means shipping the algorithm itself along with its data set (thought the data set may reside somewhere else and be queried by that algorithm). Of course, all content arrives at the media domain as it is published, so this is always what you are going to deliver. You will always be creating (or borrowing) an algorithm to ship with your content, and setting up your content so that algorithm can read it.

In the document domain, this means defining document structures to explicitly support certain kinds of manipulation. For example, a generic table does not support the action of allowing the reader to sort

on any column. Sorting by column only makes sense if the content consists of identically structured rows. In other words, when you sort a table on a column, you are actually sorting the rows. Sorting this table by column would accomplish nothing meaningful:

item	table	stool	shooting stick	chair
legs	4	3	1	4
price	$400	$20	$75	$60

As laid out on a page, a table may group related information by row or by column. Unless you know which is which, you don't know which sort makes sense. Still other tables provide a single value lookup based on two inputs. Sorting this kind of table makes no sense either way. The only logical sorts are on the first column and the first row, which presumable are sorted correctly to begin with.

Still other tables may be lists in disguise. Imagine the consequence of sorting this table on the second column:

1.	Don protective clothing.
2.	Clear the area.
3.	Block all entrances.
4.	Activate the destruct sequence.

Even with row-oriented tables sorting on every column does not always make sense. Sorting on the name column or the size columns or the price column makes sense. Sorting on the description column or the picture column does not.

To implement column sorting at the document domain level, therefore, you need some sort of sortable table structure which assures that the sorting behavior is only applied to columns or rows where it makes sense in tables where it makes sense.

In the subject domain, of course, you are not creating a table at all, or at least, not a table in the publishing sense of the word. You are creating a subject-based structure to capture information which can then be used to build different document domain presentations for different purposes. The fact that some of these presentation may be static and some may be active is orthogonal to how the information is represented in the subject domain. In other words, by the time we move content to the subject domain we have factored out the behavior.

This is an idea we have looked at before. When we looked at separating content from formatting we noted that in a world that includes interactive media, we also needed to think about separating content from behavior. We mentioned then that separating content from formatting means at minimum moving it to the document domain. We also saw that when it comes to differential single sourcing you often need to move your content to the subject domain in order to create different document domain presentations for different media. The same is true of separating content from behavior.

In the subject domain you present a document domain structure that is capable of being acted on by an algorithm, but you factor out the algorithm itself. But in a differential single sourcing scenario, you might want a different structure for different devices. You might want a static table, perhaps laid out differently, for static paper or PDF presentation, a sortable table for presentation in desktop web browsers or tables with sufficient room for a table, and a different kind of lookup mechanism altogether for presentation on the limited real estate of a phone. In this case, you can factor out the particular active content structure by moving the content to the subject domain.

In the subject domain, this might mean using a different kind of table: a database table. The form of a database table is simple: it is a series of rows with a common structure. The values in each row are presented in cells in the same order with the same type of data in each cell. For instance, in the recipe example we have been using, the list of ingredients has this form:

```
ingredients:: ingredient, quantity, unit
    eggs, 3, each
    salt, 1, tsp
    butter, .5, cup
```

In SAM, the markup language used for most of the examples in this book, there is a specific markup structure for this kind of information. It is called a record set, and the markup above is an example of it.

With this markup, you can construct a table this way:

eggs	3	each
salt	1	tsp
butter	.5	cup

Or this way:

eggs	salt	butter
3	1	.5
each	tsp	cup

We know, from the semantics of the record set, which rows or columns it would make sense to be sortable, and we can create a document domain sortable table accordingly.

Of course, we can also construct this:

- 3 eggs
- 1 tsp salt
- .5 cup butter

Or this:

- eggs, 3
- salt, 1 tsp
- butter, .5 cup

We can generate almost any kind of active content that we can think of on a piece of content as long as we know its subject domain semantics. For instance, if you annotate a stock symbol:

```
Microsoft ({NASDAQ:MSFT}(ticker)) is a large software company.
```

Then you can look up the current stock price when the page loads and display it like this:

> Microsoft (58.02USD -0.18 (0.31%)) is a large software company.

Alternatively, you could use the ticker symbol to annotate the company name:

```
{Microsoft}(company "NASDAQ:MSFT") is a large software company.
```

This can be output with active behavior in exactly the same way:

> Microsoft (58.02USD -0.18 (0.31%)) is a large software company.

The subject domain markup would be translated to the document domain as part of the publishing algorithm. Here’s what that might look like:

```
<p>Microsoft (<lookup type="stock-price" symbol="NASDAQ:MSFT"/>) is a large sof
```

This would then be translated into the media domain as a call to a particular web service that provides stock quotes, and perhaps some JavaScript code to format the result.

As you can see, creating your content in the subject domain gives you the greatest flexibility to generate active content in ways that are appropriate to the subject matter and the device, and to do so without requiring authors to understand or even think about how the active content might work.

This does not mean that active content is a free gift of the subject domain, however. Apart from the fact that you still have to design and implement the content behavior, you also have to think about the subject domain structures and annotation that you will need in your content to drive these behaviors and design them into your content structures. You will also need to make sure that you get a high degree of conformance to these structured from your writers, as it is difficult to validate the correct operation of every active content algorithm on every content set at run time. The success of your active content strategy is going to depend heavily on the quality and consistency of your input data.