

## The Problem of Context

One of the biggest problems I've had working with Ektron over the last 18 months and 15 or so implementations is context. Content blocks don't live in isolation. Just as important as what's in the content block is *where the content block is located relative to other content*.

There are settings that need to have values based on a content block's position in the tree. These things range from what menus to use to what color the header should be. This is true of most every Web site I've done beyond the trivially simple.

I haven't found a good solution to this. Here's what I've tried:

### Premier Bankcard

For my first project – the Premier Bankcard intranet – this problem was really acute. The users were creating pages for their department sub-sites. I needed to render the pages differently based on what department the content was in. The Collections department, for instance, needed to have the Collections department menu on the left, and it needed to have a link back to the Collections department home page.

The problem is, *a content block knows very little about where it sits in the tree except for its own folder ID*. So my first plan, naturally, was to key off the folder ID to figure out what to render around the content. This presented two problems:

1. I would have to hard-code a bunch of folder IDs and map them to settings.
2. What if the users started creating subfolders? I needed all these settings to inherit.

What I ended up doing was using the one thing I could get to inherit – the URL.

I created “context files” for each department. These are XML files sitting in a designated folder on the file system that hold the settings for a department (or a “context”). So, the file for a department might look like this:

```
<Context>
  <MainMenuID> 43< /MainMenuID>
  <HomePageLink>/page.aspx? cxt=dept_co&id=456</HomePageLink>
  [etc.]
</Context>
```

Then, in the template for the departments folder, I set it to:

```
/page . aspx? cxt=dept_co
```

The “cxt” URL param is the context file to use. So the page called from above would open the “dept\_co.xml” context file and use it to retrieve its settings. Since the template path inherits, all content below a given point in the tree will inherit settings.

This works okay, but there are a couple problems:

1. I hate goofy URLs (we gotta talk about the SmartMenus at some point...)
2. Someone can view content in the wrong context by just changing the URL. Not a big deal, but annoying, and it might be a real problem in a different setting.
3. Context is very rigid. Even if I just want to change one thing in a subfolder, I have to create a whole new context. Put another way, a page can only pull one context.

(Final note: in retrospect, these context files should have just been SmartForm content blocks, so that the settings could be content managed. Instead of passing in the filename on the “cxt” parameter, I should have just passed in the relevant content block ID. But, it was my first Ektron project, so I didn’t have the perspective then that I have now.)

## Clarke College

(Note: this is the system I showed to Ed.)

Clarke was a lot like Premier in its core problem – context mattered a lot. There were scads of little sub-sites that all needed to carry their own settings. I learned from the lessons of Premier, and did something different here.

I created a SmartForm for “Folder Settings,” which had fields for settings I needed to store: home page link, color scheme, header graphic, etc. So, at any time, you could create a “folder settings object” in a folder.

Then I created a class called “ContentContext.” The constructor for this class took in a folder ID, and I called it like this:

```
ContentContext MyContext = new
ContentContext(MainContentBlock.EkItem.FolderId);
```

It started at the folder which was passed in. It created an XML element that looked like this:

```
<Folder id="43">
  <Name> Collections< /Name>
  <Description>The collections department .</Description>
</Folder>
```

Then it looked in that folder for a piece of content based on the Folder Settings SmartForm. If it found one, it embedded the whole thing. So the final element might look like this:

```
<Folder id="43">
  <Name> Collections< /Name>
  <Description> The collections department .</Description>
  <Settings>
    <SectionName>Collections Department</SectionName>
    <HomePageLink> / collections< /HomePageLink>
    <MenuID>104</MenuID>
  </Settings>
</Folder>
```

Then it went to the next folder up, and did the same thing. It did this all the way back to root, returning a big document like this:

```
<Context>
  <Folder id="43">
    <Name> Collections< /Name>
    <Description> The collections department .</Description>
    <Settings>
      <SectionName>Collections Department</SectionName>
      <HomePageLink> / collections< /HomePageLink> <Menu
        I D>< /MenuI D>
    </Settings>
  </Folder>
  <Folder id="2">
    <Name> Departments< /Name>
    <Description>Department folders .</Description>
    <Settings>
      <SectionName>Departments< /SectionName>
      <HomePageLink>/departments< /HomePageLink>
      <Menu ID> 25< /MenuID>
    </Settings>
  </Folder>
  [etc]
</Context>
```

I had methods on my ContentContext object like this:

### **GetSetting(Name, FolderOffset =0)**

So by calling `GetSetting ("SectionName")` I could get the name of the section the content was in. `GetSetting ("SectionName", 1)` would give me the parent's section (the current folder, offset by one).

### **GetNearestWithValue(Name)**

This was the workhorse. By calling `GetNearestWithValue ("MainMenuID")` I would get the *nearest MainMenuID that had a value*. So, in the example above, this would return "25" – the main menu ID of the Departments folder, since the Collections folder didn't have a MainMenuID defined (it was blank; you could also have removed the element from the SmartForm, if you wanted).

And, of course, this is an XML document, so I could query it with XPath all I wanted. I often wrote some bizarre XPath queries to get into every nook and cranny of the context.

This system works pretty well. I can selectively override arbitrary settings and have them cascade down the tree. Clarke uses this to define their menus and header images.

It took a little work to explain the abstraction to them, but Tricia totally understands it now and has actually come to me with ideas for other settings she wants to cascade. Sure, it's a logical abstraction, but I've found that it's one that users understand pretty easily.

The other benefit to this system – and this can't be stated strongly enough – is that settings *are treated as content*. So they are:

1. Subject to permissions.
2. Subject to versioning.
3. Subject to workflow.

*Settings are just as important as content*, and this system treats them as such. If someone screws up a setting, I can go in and roll it back to a prior version. By setting the permissions right, only Tricia can see the settings objects.

There are drawbacks here:

1. The “settings object” can get lost among other content. I should have told Tricia to use a naming convention that puts them at the top of the list (call them all “\_settings” or something).
2. There's just one SmartForm, so if she needs a setting for a single folder, I have to add it to the SmartForm for all settings objects.
3. Access to settings is binary. Someone who needs access to a single setting gets access to them all.

## **The Future: PMI**

I'm doing a big intranet for PMI which will rival Premier's in terms of scope. And the core problem is identical: content has very deep context. Every page lives in a department, and it needs to be rendered as such.

For this, I plan to extend the Clarke System to solve some problems. I'll also likely roll up some of the functionality into a plugin.

Instead of a “settings object,” I'll have a “settings folder.” This will be a subfolder that contains multiple settings objects for that folder. So a folder might have a subfolder called “\_settings” that contains several SmartForm-based content blocks. When the ContentContext object crawls up the tree, it will take all the content in the “\_settings” folder and throw them in the XML it returns. (How will it identify the folder with the settings? Name, probably. Unless you have a better idea.)

This gives us a couple benefits:

1. We can start to segregate settings by having multiple objects. I can have a SmartForm for “Navigation” which defines the menus to be used. I can have another SmartForm for “Colors” which defines the color scheme for the page. I can manage permissions for these differently – perhaps I want to let someone change the color scheme for the Collections department, but not the navigation.
2. It removes settings objects from the rest of the content. This means that ListSummaries won't return settings with other content.

3. We can have multiple settings objects of the same type. I actually want to create a SmartForm for “Miscellaneous Setting” which would have just two fields: “Setting Name” and “Setting Value.” This means I could create a single, arbitrary setting for specific folder, and get to it via XPath on the context XML. These would be great for the odd exceptions. (For Clarke, I could have actually created one for “Master Page Override” since there was one tiny section in there that needed a completely different master page than everything else in the site.)

## What Ektron Could Do

Since this is a hack, there are some rough edges. But there are some things Ektron could change to make it work more smoothly.

### Create “Hidden Content”

There needs to be a way to keep settings objects out of the normal flow of the other content. Specifically, they tend to get returned by recursive ListSummary controls. It would be nice if I could specify that content was “hidden,” so that it exists in the workarea, but nowhere else. This content would never get returned by any public-facing control.

(Actually, I could probably hack this up now by automatically expiring everything in the “\_settings” folder, but that might cause other problems.)

There are a couple ways Ektron could bring this about:

1. The quick and easy way: Give content a checkbox: “Hide this content.” Better yet, put this on the folder: “Hide all content in this folder.”
2. The smooth but more complex way: Create another “view” of a folder. Right now you have nine ways of viewing a folder: content, images, files, hyperlinks, quicklinks, forms, calendars, collections, and menus. Add a tenth: settings. (But, in doing this, *do not stop treating settings as content*. They need to be versioned, subject to permissions, and subject to approval chains.)

### Allow Derivative Content Naming

It gets a little silly that you have to “name” these settings objects. The navigation object should be called “Navigation Settings.”

In general, my users often have to enter the same thing in two fields. If I have a SmartForm for an “Article,” I want an element for “Title,” but the content block also has a title. I can’t just use the content block title because you guys don’t allow some characters up there, and the titles have to be unique (why is this, by the way?), so this can be a problem sometimes. So the users just enter the same thing both fields – the actual title, and the “Title” element – which is a little silly.

I’ve always thought that content titles should be derivable from the XML of a SmartForm. When I define a SmartForm, allow me to specify “Title XPath.” If I specify this, the content block can’t be titled by the user. Instead, when published, this XPath is used to extract content from the XML and use it as the title.

For an “Article” SmartForm, the “Title XPath” could be “/root/Title”. For a “Staff Member” SmartForm, it could be “/root/Name”.

I can sort of approximate this with a plugin – I could extract the value and rename the content block on publish. However, the workarea requires a title to be entered, so the users would have to enter *something* in the title field.

### **Allow Permissions on a Per-SmartForm Basis**

This might be hard, and it could open up a huge can of worms. But it would be handy to be able to define a permission set for a SmartForm, so any content created from that SmartForm would be subject to the same permissions. Again, probably tricky from a technical and logical perspective, but there have been times when this would have been very handy.

### **Allow Limits on SmartForms Per Folder**

Again, this is tricky, but it would be very handy. When I specify the SmartForms allowed for a folder, let me put a limit on it. I could limit a folder to one “Navigation Settings” SmartForm, for instance. If not an arbitrary limit, let me just say that only *one* of this SmartForm can exist in a folder – that would handle 90% of the situations where I want this.