Debugging a SharePoint customization

SharePoint has a large and complex code base, and some of the details aren't as transparent or well-documented as developers need. This is a short overview showing how bugs can arise when working with incomplete documentation, and how we can use .NET Reflector to fix those bugs.

In this scenario we're working on a web portal, and we want to add a set of controls to the ribbon. This is simple enough, but we run into an unexpected error. We use .NET Reflector to explore the SharePoint API that is throwing the exception, so we can understand the problem and troubleshoot the control.

Defining custom controls

We're looking at an issue in SharePoint 2010, where XML comments in definition files cause an exception. The issue can arise when creating custom fields, content types, and other custom features that require XML definition files. The XML schema for SharePoint Features is documented, but the documentation doesn't always give rich implementation information.

The custom action behind the control is defined in the XML file, and it's part of a group of controls being used in the ribbon. Because these files quickly become long and complex, we want to comment the code.

This is a simplified "hello world" example of a custom action definition file:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
        <CustomAction Id="Ribbon.WikiPageTab.CustomGroup" Location="CommandUI.Ribbon">
                <CommandUIExtension>
                        <CommandUIDefinitions>
                                <CommandUIDefinition Location="Ribbon.WikiPageTab.Groups._children">
                                        <groups>
                                                <!-- Group containing the new "hello world" controls -->
                                                <Group Id="Ribbon.WikiPageTab.CustomGroup" Sequence="55"
Description="Custom Group" Title="Custom" Command="EnableCustomGroup" Template="Ribbon.Templates.Flexible2">
                                                        <Controls Id="Ribbon.WikiPageTab.CustomGroup.Controls">
                                                                <Button Id="Ribbon.WikiPageTab.CustomGroup.
CustomGroupHello" Command="CustomGroupHelloWorld" LabelText="Hello, World" TemplateAlias="o2" Sequence="15" />
                                                                <Button Id="Ribbon.WikiPageTab.CustomGroup.
CustomGroupGoodbye" Command="CustomGroupGoodbyeWorld" LabelText="Good-bye, World" TemplateAlias="o2" Sequence="
18" />
                                                        </Controls>
                                                </Group>
                                        </groups>
                                </CommandUIDefinition>
                        </CommandUIDefinitions>
                        <CommandUIHandlers>
                                <CommandUIHandler Command="EnableCustomGroup" CommandAction="javascript:return
true;" />
                                <CommandUIHandler Command="CustomGroupHelloWorld" CommandAction="javascript:
alert('Hello, world!');" />
                                <CommandUIHandler Command="CustomGroupGoodbyeWorld" CommandAction="javascript:
alert('Good-bye, world!');" />
                        </CommandUIHandlers>
                </CommandUIExtension>
        </CustomAction>
</Elements>
```

(In practice, we could be dealing with hundreds of lines of XML)

When the portal page renders, the controls appear. But when the control is actually used, SharePoint displays an error.

Debugging the error

SharePoint error messages typically have a Correlation ID , a GUID that lets us track the error through the logs. From these logs, we see that the error is a null reference exception.

This is surprising, because in building our control, we've followed the SharePoint documentation pretty thoroughly. At first glance, there appears to be nothing wrong with any of our own code, and the custom action definitions match the documented schema.

To solve the problem, we look into the code with .NET Reflector.

We see that the exception is thrown by Microsoft.Web.CommandUI.Ribbon.CreateRenderContext, so we search for that library using .NET Reflector desktop, and decompile it to view the underlying source code.

The code .NET Reflector shows us is:

```
DataNode node4 = uiproc.GetResultDocument().SelectSingleNode("/spui:CommandUI/spui:Ribbon/spui:Tabs/spui:Tab
[@Id='" + this.InitialTabId + "']/spui:Groups");
if (node4 == null)
{
        node4 = uiproc.GetResultDocument().SelectSingleNode("/spui:CommandUI/spui:Ribbon/spui:ContextualTabs
/spui:ContextualGroup/spui:Tab[@Id='" + this.InitialTabId + "']/spui:Groups");
}
Hashtable hashtable = new Hashtable();
if (node4 != null)
{
        foreach (DataNode node5 in node4.ChildNodes)
{
XmlAttribute attribute2 = node5.Attributes["Id"];
if (attribute2 != null)
{
        hashtable[attribute2.Value] = node5;
}
```

If a node is a comment, rather than a conventional element, the value returned is null. This is because XmlNode.Attributes returns as null for all nodes that are not of the XmlNodeType.Element type. This null result is unexpected, so an exception is thrown.

Luckily, the solution is simple, and removing the comments prevents the exception. This may not be an ideal workaround, but because the limitation lies in a third-party assembly, it can't be fixed directly.

Improving on documentation

The XML comments issue is a small example of an undocumented limitation. It's simple to fix with decompilation tools, but would be a substantial debugging task without the ability to look inside the underlying code.

Unfortunately, incomplete (or entirely absent) documentation is quite common for 3rd party tools libraries, and frameworks. SharePoint itself is elaborate, complex, and difficult to debug and understand using only the published documentation. In practice, you can encounter unusual errors, or require customizations that depend on entirely undocumented behavior.

In these cases .NET Reflector saves troubleshooting time, and lets you explore 3rd party libraries in context, to better understand how to integrate with them and build upon them.