SDL Tridion 2011
Event System Quick Start Guide

(A Collaborative San Jose Boot Camp Effort)

Created: 3/7/11
Last updated: 3/8/11

Audience

This guide shows developers how to create and configure a simple event system extension for SDL Tridion 2011. Reference and update it as needed as an example for those new to the 2001 event system.

Use Case

The event system provides a programmatic way to respond to events that occur as users create, save, check in, and publish content with the content management system (CMS). It lets developers create .NET dynamic link libraries (dlls) that can automate tasks such as validation, adding metadata, creating additional components, or starting processes outside of content management and publishing.

> *Developer expertise is required, but the event system could be used as a "phase 2" project to automatically create and publish pages (see REI example[1]) or notify users of publishing via email. Such implementations are beyond the scope of this document, however.*

Comparison

Feature	Old	New
Modularity	Single dll to handle all events	Separate dlls can handle separate or even the same events (ARR: is this correct?)
Performance	Old event system "blocked" on triggered events (ARR: edit as needed)	Events can be triggered asynchronously, allowing end-users the ability to continue working in the CMS
Maintenance	.NET through COM+ Interop required a lot more code (ARR: need technical definition for "lot more")	Core defines delegate methods—only modify required methods

Prerequisites

To best follow this guide, you should have:

[1] See http://www.sdltridionworld.com/community/2011_extensions/rapideditorialinterface.aspx. Also see http://www.julianwraith.com/2011/02/event-systems-in-sdl-tridion-2011/ for another example.

- Some familiarity with Visual Studio 2010 and .NET or other integrated development environment (IDE) and programming language
- Tridion Content Management System (CMS) installed on a virtual machine (VM) or development server
- Visual Studio 2010

If using a VM, both the CMS and IDE can be in the same virtual machine. If developing for a development server, consider running the IDE locally, outside of the server. System administrators may not take kindly to having Visual Studio running on the server.

Contents

Setup Tridion

The Tridion Content Management System (CMS) needs to have a basic setup including at least publications, schemas, and components. Templates, structure groups, pages, and publishing targets are required as needed based on the events you want to “capture.”

Target Environment

A virtual image was used to create this write-up, but the same process could be used modify a development server. Always test changes before releasing code to a production system.

Set Up Project

Open Visual Studio 2010 (choose Visual C# or a development environment of your choice)

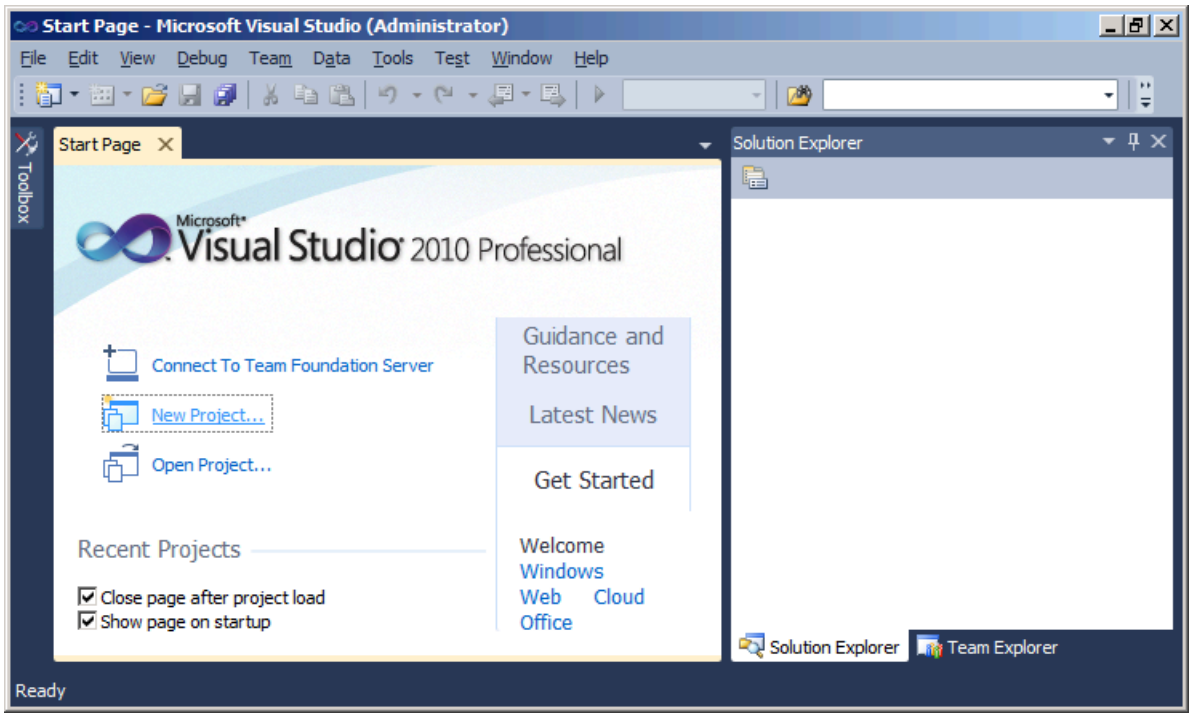

Figure 1 - Select "New Project..." from the Visual Studio 2010 Start Page to get started.

Create a New Project using the Visual C# Class Library (.NET Framework 4)

- Check “Create directory for solution”
- Optionally check “Add to source control” based on your development preferences

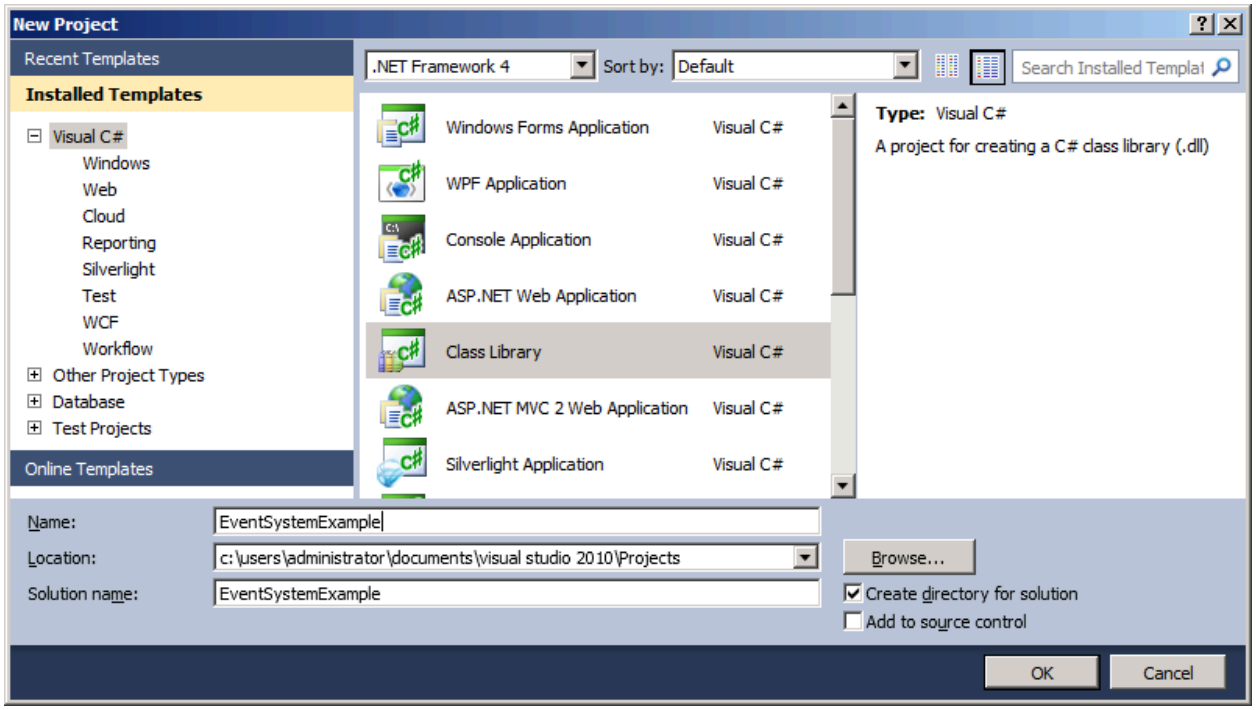

Figure 2 - Use (C#) Class Library to start creating the dynamic link library (dll).

> *Preferences may vary, but consider using standard practices when creating event system libraries. Naming conventions, documentation, testing, and versioning (source control) will go a long way towards ensuring "trust" in the CMS. The best event system implementations are practically "invisible."*

Create a Class Library. Choose an appropriate Name, Location, and Solution name.

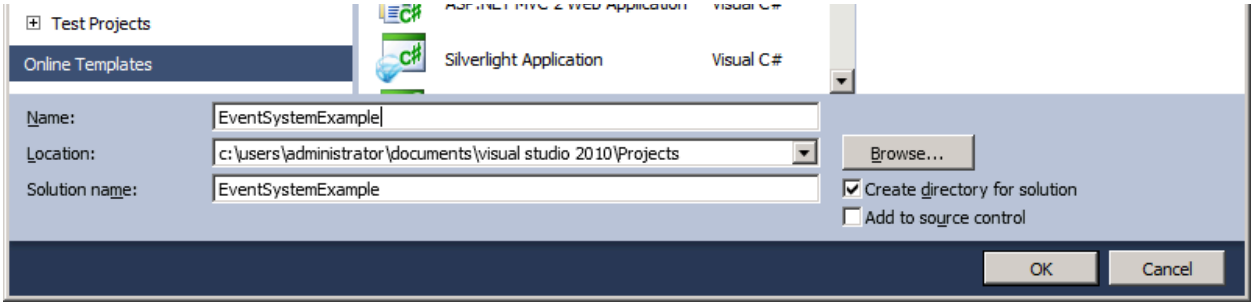

Figure 3 - Use an appropriate name, location, and solution name for your class library project.

Class1.cs is created for you.

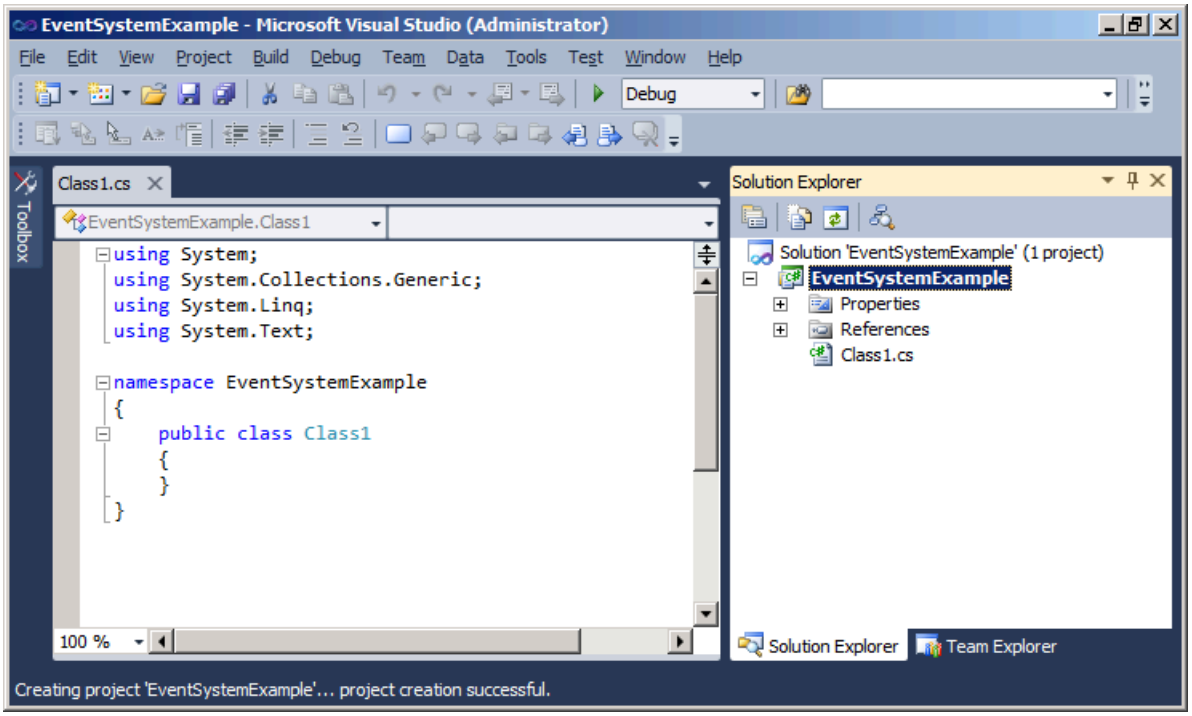

Figure 4 - The default C# file, "Class1.cs" is created for you. Modify this file to extend the event system.

In the Solution Explorer window, right click on the References folder to add references from the bin > client folder for Tridion:
C:\Program Files (x86)\Tridion\bin\client
(add all .dll except TcmUploadAssembly)

Create Code

Add the following imports and modify the following sample code to match the schemas and locations in your project.[2]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Tridion.ContentManager.Extensibility;
using Tridion.ContentManager.ContentManagement;
using Tridion.ContentManager.Extensibility.Events;
using Tridion.ContentManager;
using System.Text.RegularExpressions;
using Tridion.ContentManager.CommunicationManagement;
using Tridion.ContentManager.ContentManagement.Fields;
using Tridion.Logging;

namespace TridionEventSystem
{
    [TcmExtension("My Event")] public class MyEvent : TcmExtension // MyEvent should be
instantiated as a TcmExtension
    {
```

[2] The Webdav paths reflect a specific VM setup. Consider moving these to an attributes or configuration file or even global variables to improve maintenance.

```
        public MyEvent() // constructor
        {
            EventSystem.Subscribe<Component,
CheckInEventArgs>(ComponentCheckInTransactionCommitted,EventPhases.TransactionCommitted);
        }

        public void ComponentCheckInTransactionCommitted(Component subject, CheckInEventArgs
args,EventPhases phase)
        {
            Logger.Write("ComponentCheckInTransactionCommitted started", "Event System",
LoggingCategory.Configuration);
            if (subject.Schema.Title != "Content")
                return; // only process for schemas named "Content"
            Session s = subject.Session;

            String title = subject.Title;
            String filename = Regex.Replace(title, "\\W", "");
            StructureGroup sg = s.GetObject("/webdav/050%20Site/Root") as StructureGroup; //
update to a folder in your blueprint
            ComponentTemplate ct =

s.GetObject("/webdav/050%20Site/Building%20Blocks/System/Component%20Template/Generic%20Compon
ent%20Template.tctcmp") as ComponentTemplate;
            Page p = new Page(s, sg.Id);
            p.Title = title;
            p.FileName = filename;
            TcmUri ComponentInContext = new TcmUri(subject.Id.ItemId,
ItemType.Component,sg.Id.PublicationId);
            Component c = s.GetObject(ComponentInContext) as Component;
            p.ComponentPresentations.Add(new ComponentPresentation(c, ct)); // component
presentation = component + component template
            p.Save(true);
        }
    }
}
```

Copy DLL

When done, build the solution from the Build menu (build solution) or press F6. Find the compiled dll in the solution folder by right-clicking on the solution in the Solution Explorer. Select “Open Folder in Windows Explorer” to navigate to the solution folder, open bin > Debug. Copy the .dll (EventSystemExample.dll in this example) to a folder available to the CMS.

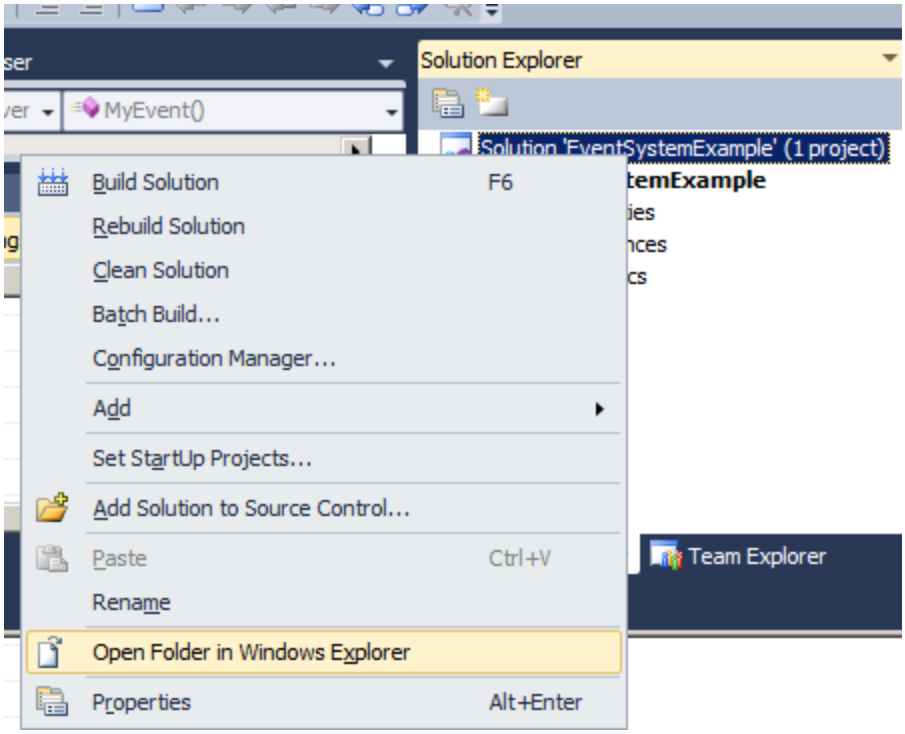

Figure 5 - The Open Folder in Windows Explorer is a built-in feature in Visual Studio 2010.

Copy the dll

Use a copy of your compiled dll because the Tridion COM+ service will lock the file, making development difficult.

To copy over the dll, stop the SDL Tridion Content Manager under COM+ Applications in Component Services

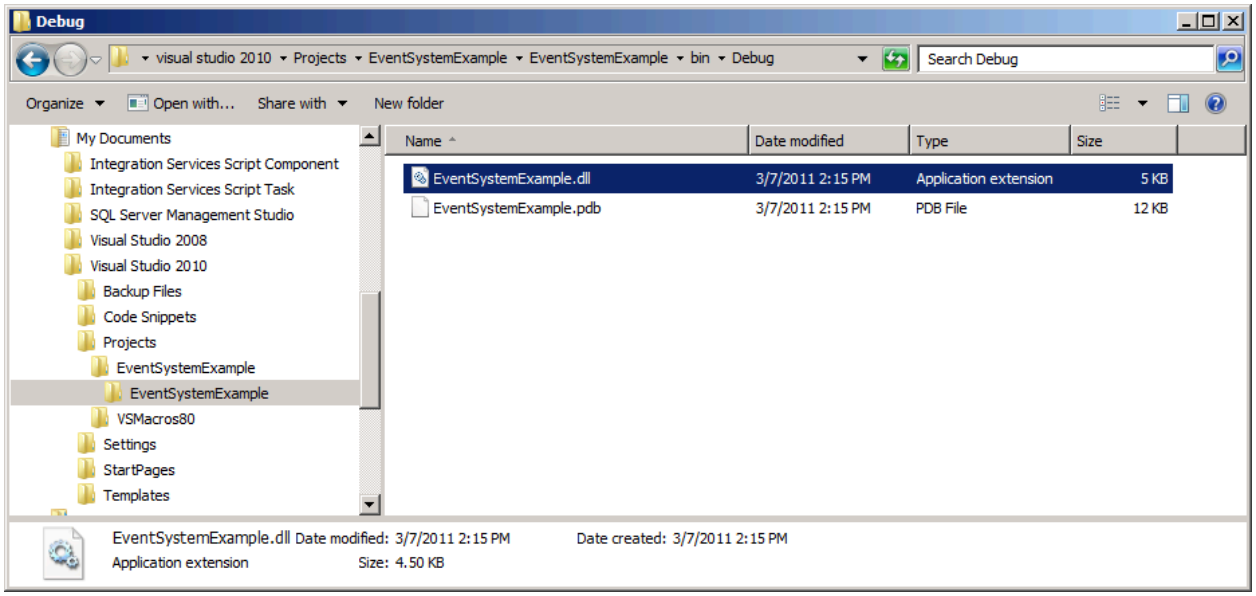

Figure 6 - By default, the "Debug" version of the dll is created. Copy this file to a folder accessible to the CMS.

Configure the CMS

Make the CMS recognize this event system extension by adding the path to the dll copy to the Tridion.ContentManager.Config configuration file.

Under <extensions> node, add <add assemblyFileName="C:\path_to_dll\myextension.dll"/>.

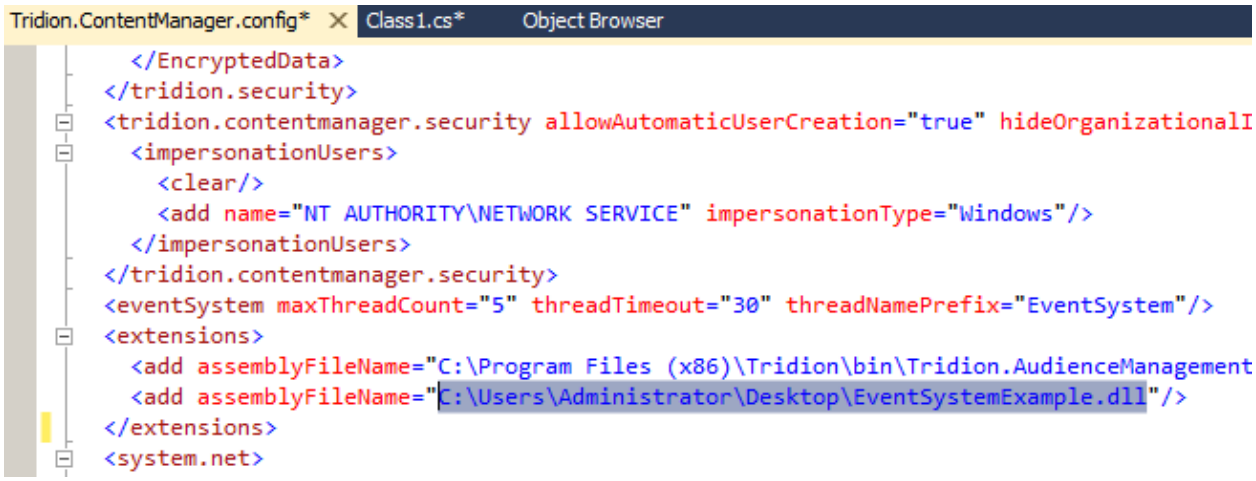

Figure 7 - Add your custom event system extension in the <extensions> node.

Restart the COM+ service in Component Services and the CMS Website in IIS.

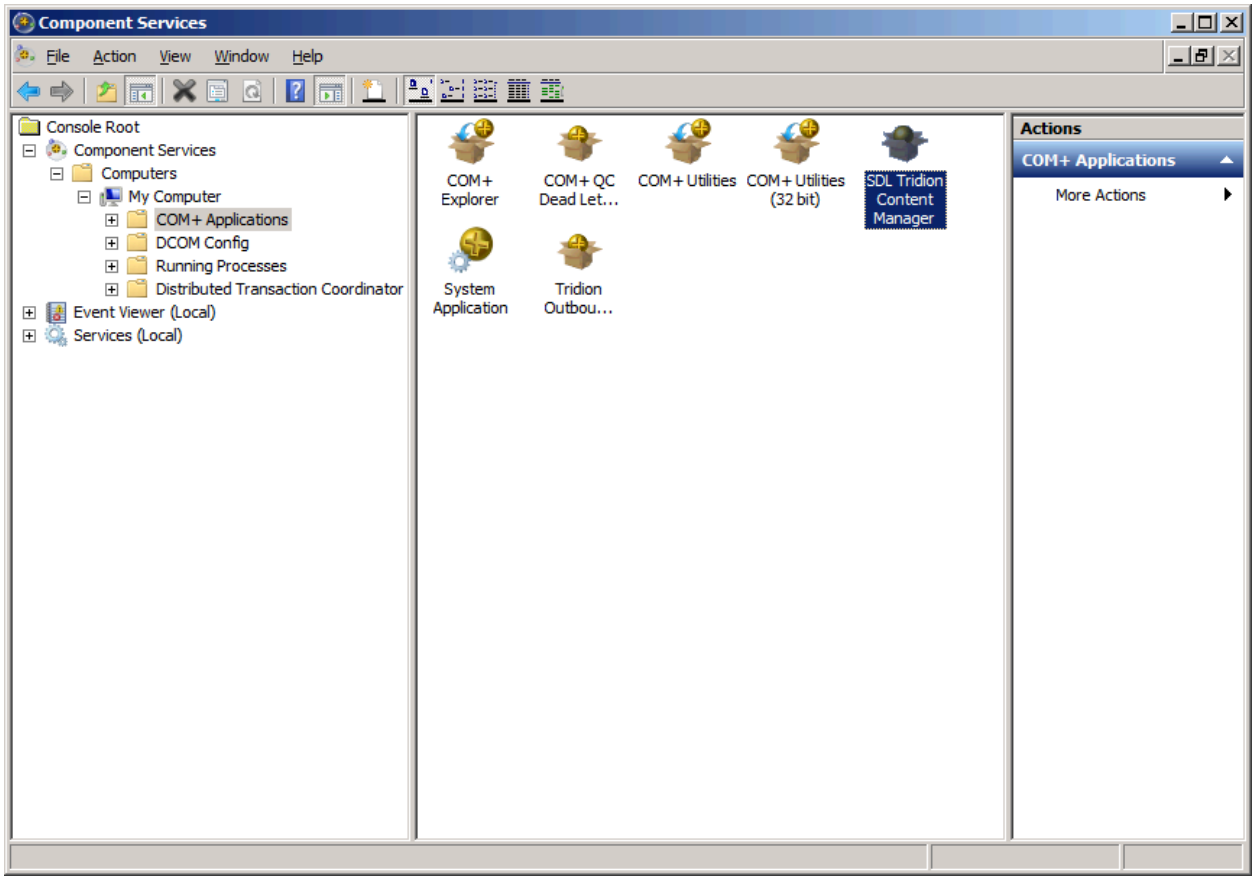

Figure 8 - Find the COM+ Applications folder in Component Services, under My Computer. Right-click and restart the SDL Tridion Content Manager.

Though developer expertise and understanding of the Tridion Content Manager objects is required, the 2011 implementation of the event system further "modularizes" the Tridion event system, providing a flexible way .

Appendix A: Developer Q&A

Event System Granularity

Question: "Is it possible to disable specific events?"

Answer: only if each event is handled by a separate library (dll). Consider this a "best practice" tip. Suggestion for Tridion Ideas: easier-to-maintain event system extensions via an interface (Content Manager Explorer or other GUI) to managed this configuration outside of the CMS XML config file.

On Syntax…

The opening square bracket syntax indicates a C# attribute which adds (code, not CMS) metadata that can be handled during runtime or during design. The String parameter "My Event" and class name "MyEvent" below are customizable—use an appropriate name.

```
 [TcmExtension("My Event")] public class MyEvent : TcmExtension // MyEvent
should be instantiated as a TcmExtension
```

The colon (":") is C#'s syntax to designate a class should be instantiated as another class. [ARR: would subclass, interface, or "extends" be appropriate here?]

To inspect the signature for a given method, use F12 in Visual Studio. For example EventSystem.Subscribe has two signatures, one of which uses a "delegate" to seemingly pass three parameters where it appears two are expected.
[ARR: have a C# programmer review—this was a follow-up on a point from Mihai, basically – how would we know what to implement and/or get info on the correct parameters)

Can we debug?

*Yes. Attach to process to **dllhst3g.exe** (which should be running under the mtsuser or "Tridion" user).*

Who will write a CMS GUI extension to make loading Event System dlls easier?

Could it be… you?!