Chapter 7

Scripting via IronPython

“There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.”

—C. A. R. Hoare

We’ve already discussed ways to extend application functionality in the CMS, primarily through reliance on the new Managed Extensibility Framework. This is an extremely effective method of modifying the behaviors and capabilities of the system but certainly doesn’t represent the limit of our options. In this chapter, we’ll look at how to implement a scripting engine in the CMS based on IronPython, as well as how we can leverage the language and interact with the CMS in a very fluid, natural way.

How Does the CMS Benefit from Scripting Capabilities?

There are a variety of reasons to incorporate scripting in the CMS: easier debugging of troublesome pages and content, rapid prototyping of ideas, and potentially allowing users to customize their pages at a code level come to mind immediately. Let’s look at each of these possibilities in more detail.

Easier Debugging

As far as the developer experience of working with the CMS goes, we have already gained a significant amount of architecture flexibility by using the Managed Extensibility Framework to promote the use of plug-ins as opposed to changing the core CMS platform every time we need to make a change to the system.

A flexible architecture is great, but our real-time debugging abilities are still hindered somewhat by the fact that we (currently) have no way to interact with a page beyond the tracing and debugging facilities that the .NET Framework provides out of the box. We can capture memory dumps and analyze them in WinDbg (a topic covered in Chapter 8), but we’re still a bit removed from a problematic event and the forensic analysis component of fixing problems.

By incorporating a scripting capability into the CMS, we effectively open to the door to interacting with the page in real time, which can be a tremendous asset while diagnosing problems that occur in a production environment. If we create not only the functionality to script the CMS, but the ability to persist those changes, we provide ourselves with the convenience of being able to hot-patch content without taking the CMS offline.
CHAPTER 7 ■ SCRIPTING VIA IRONPYTHON

This ability to develop in real time and see the effect on the production system requires careful consideration and responsible development to ensure that it is used appropriately, but the benefits tie very closely into the next point, which is the efficiency provided with regard to rapid prototyping.

Rapid Prototyping

As we’ll see, IronPython scripting allows us to operate on content in the CMS and make full use of the .NET Framework. From the developer standpoint, this has some pretty powerful implications. It’s possible to create an environment such that a developer can load a piece of content in the CMS, script entirely new controls and behaviors for that page, and persist those changes to the database to be used each time that content is loaded.

In practice, the act of creating complex controls becomes a bit trickier than that; truthfully, the best result is attained by taking a prototyped idea and executing it in a server-side control via the MEF plug-in system the CMS has established thus far. It does prove to be useful in that a developer can essentially work on CMS components from any machine with a functional Internet connection and web browser.

Note I have actually done this exact routine on a production instance of the CMS. There was an issue with a plug-in that occurred only under specific conditions, and at a very late hour I was able to eliminate the problem with an IronPython script. The following morning (and after some more rest), the problem was resolved in the plug-in, and the system was immediately updated accordingly. The combination of scripting capabilities and the MEF plug-in architecture gave me the ability to respond quickly without bringing any systems offline.

An Introduction to IronPython and Its Syntax

For the purposes of the CMS, we’ll be incorporating IronPython as our scripting language of choice. Although a .NET language, IronPython is very different from C#; if you’re unfamiliar with the language and its syntax, we will cover the fundamentals here before adding scripting capabilities to the CMS core later in the chapter. If you’re already a well-versed Pythonista, feel free to skim or skip directly ahead to the “Building Scripting Capabilities into the CMS” section.

What Is IronPython?

The shortest answer to this question is that “IronPython is an implementation of Python that is designed to run on the .NET Dynamic Language Runtime.” All the traditional Python keywords and built-in functions are present, but with the additional capability of being able to make use of the .NET Framework libraries natively. As such, you could write code in IronPython that doesn’t actually use any of the framework libraries at all if you so desired. If you’re already well-versed in the Python language, you’ll simply be gaining .NET features rather than replacing the language syntax you’ve become accustomed to.

The DLR is a set of services built on top of the Common Language Runtime; these services provide a dynamic type system suitable for languages such as Python and Ruby, as well as a hosting capability. As a result, code written in a traditional .NET language such as C# is able to operate on objects created