CHAPTER 9

XMLReader

XMLReader is a new stream-based parser in the PHP 5 lineup. If you skipped the previous chapter on the xml extension and do not know what a stream-based parser is, it may be beneficial to at least review that chapter because it explains in more detail what a stream-based parser is and how it works.

This chapter will introduce you to the XMLReader extension, explain the reasons for the existence of yet another stream-based parser, and show how to use this extension. The chapter will show how to use the API through short examples, with a complete example toward the end of this chapter. You can find additional examples of using this API in other chapters of this book, such as Chapter 14, which covers RDF, and Chapter 17, which covers REST. By the end of this chapter, you should understand what XMLReader is, know what its advantages and disadvantages are, and have a working knowledge of how to use the API in your everyday coding.

Caution Constants have been moved to class constants in PHP 5.1. This differs from PECL version 1.0.1 where constants are regular constants. The examples in this book use class constants to maintain compatibility with the PHP releases.

Introducing XMLReader

The XMLReader extension is an object-oriented API that uses the libxml2 implementation, which in turn is based on the C# implementation of the XmlTextReader API (`http://dotgnu.org/pnetlib-doc/System/Xml/XmlTextReader.html`). The XMLReader extension is a forward-only, stream-based parser, but unlike SAX, it is a pull rather than a push parser. As you move through a document, the parser's cursor positions itself on the different nodes, allowing you to access information from the current node. It offers many advantages over the xml extension, including additional functionality. As of PHP 5.1, this extension is part of the core PHP code base and can be built using the following configuration option:

```
--with-xmlreader
```

If you are still using PHP 5.0.*x*, the XMLReader extension is available from the PECL repository at `http://pecl.php.net/package/xmlReader`. You can install it using the PEAR installer or build it by adding it to your PHP source tree. Refer to the PHP manual for further information about building extensions.

R. Richards, *Pro PHP XML and Web Services,*
DOI 10.1007/978-1-4302-0139-7_9, © 2006 by Robert Richards

Push vs. Pull Parser

The previous chapter introduced you to stream-based parsing and the xml extension in particular. It explained that a *push* parser, in simple terms, pushes the data to your application while the XML is being parsed. The parser basically controls the flow of your application. A *pull* parser works much differently. It still operates on chunks of data at a time, providing a low memory footprint, but the application is in control of what data it wants and when the data is read from the stream. A pull parser allows you to free yourself from the control a push parser has over your application.

You can think of the difference between the two in terms of watching television. A push parser is like watching television without a digital video recorder. You are sitting there watching a show, and the commercials come on. If you are interested in anything in the commercials, you have to sit there and watch them, deciding which ones you like and which ones you don't. You can't get up and grab a snack, or you might miss something. You are not in control of the commercials. They, speaking in terms of a push parser, are pushed to your television, and you can't skip them, because you might want to watch one and can't pause them.

A pull parser, on the other hand, is like watching television with a digital video recorder (without the rewind feature, of course). By using the play, pause, and fast-forward buttons on the remote, you control the shows and commercials you watch. The current stream of XML data is comparable to the buffer of the digital video recorder. Like in the previous scenario, the commercials come on. Again, you might be interested in one of them. This time, you hit the pause button and grab something to eat. You return and decide you don't want to watch the commercial, so you fast-forward to the next one or even skip the next one. The push parser lets you control the movement of the parser, which is when data is read. When it stops at the point indicated, you can do anything you like in your code. The parser won't start reading more data until you tell it to do so. Your code could even stop reading the XML data and move on to something else. With a push parser, you really have no escape. Your application must read and act on everything in the buffer until all the data in the current stream has been read. It wouldn't be until the next `xml_parse()` call that you could safely stop reading XML data and have your application do something else.

This analogy may be a little over the top, but it should give you the idea. When using a pull parser, you are in control of the processing and of when data should be read. You are not at the mercy of the parser. As you will see in this chapter, this has many advantages over the traditional pull parser model, not to mention is much easier to use.

Advantages Over the xml Extension

If you followed along with the previous section, comparing a push and pull parser to watching television without and with a digital video recorder, you may already have realized one of the advantages of XMLReader over the xml extension. With the XMLReader extension, you control when the data should be accessed. This is just one of the many benefits of this extension. Other advantages include better namespace support, streaming validation support, a simple API, and potentially faster processing.

Namespace Support

From the examples in the previous chapter, you have most likely realized that processing namespaced documents is a real headache. The tag name is sent to the handler in the form