SOAP

Cross Platform Web Service Development Using XML

- Integrate your enterprise applications across the Web!
- The platform-independent guide to SOAP
- SOAP programming with C++, Perl, C#, Visual Basic®, and Java™
- Build an industrial-strength SOAP system from scratch
- CD-ROM: SOAP for Windows®, Linux® and UNIX®, plus an extensive source code library!

SCOTT SEELY

Technical Reviewers: Yves LaFon, Chair of the SOAP W3C Committee
Kent Sharkey, .NET Frameworks Technical Evangelist, Microsoft
Appendix on SOAP and Perl by Paul Kulchenko, author of *SOAP Lite*

SOAP:

Cross Platform Web Service Development Using XML

SCOTT SEELY

PH PTR

Prentice Hall PTR, *Upper Saddle River, NJ 07458*
http://www.prenhall.com

ISBN 0-13-090763-4

9 780130 907639 90000

Editorial/Production Supervision: *Kathleen M. Caren*
Acquisitions Editor: *Tim Moore*
Editorial Assistant: *Allyson Kloss*
Marketing Manager: *Debby Van Dijk*
Buyer: *Maura Zaldivar*
Cover Design: *Anthony Gemmellaro*
Cover Design Director: *Jerry Votta*
Art Director: *Gail Cocker-Bogusz*
Interior Series Design and Page Makeup: *Meg Van Arsdale*

Prentice-Hall, Inc.
Upper Saddle River, NJ 07458

The publisher offers discounts on this book when ordered in bulk quantities.
For more information, contact
Corporate Sales Department,
Prentice Hall PTR
One Lake Street
Upper Saddle River, NJ 07458
Phone: 800-382-3419; FAX: 201-236-714
Email (Internet): corpsales@prenhall.com

Printed in the United States of America.
10 9 8 7 6 5 4 3 2

ISBN 0-13-090763-4

Pearson Education LTD.
Pearson Education Australia PTY, Limited
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd.
Pearson Education Canada, Ltd.
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education—Japan
Pearson Education Malaysia, Pte. Ltd.
Pearson Education, Upper Saddle River, New Jersey

Jean, Vince, and Angeline,
thanks for inspiring me to do great things.

Foreword

As long as there have been two computers, there has been difficulty getting them to communicate. Dozens, possibly hundreds, of strategies have arisen, each with their own strong and weak points. However, the end result is that still, it is difficult to get two computers to agree on a strategy for communication. Everyone wants everyone else to change to meet their strategy's needs. Thus, we end up with the "Communication Wars," CORBA vs. DCOM, DCOM vs. RMI, messaging vs. RPC, and so on.

Into this tangled mass of communication comes SOAP (Simple Object Access Protocol). SOAP does not try to solve all problems; it only defines a simple, XML-based communication format. However, with this simple goal, and a powerful extensibility mechanism, SOAP bears the promise of being a true cross-everything communication protocol–cross-programming language, cross-operating system, cross-platform. As long as a computer, operating system, or programming language can generate and process XML (that is, text), it can make use of SOAP. Since the initial release, almost every major software vendor has either produced, or announced, an implementation of SOAP. We've seen standalone SOAP, SOAP built into Web servers, application servers, communication tools and even messaging middleware using SOAP. In the future, SOAP will become even more prevalent, as companies and organizations like Microsoft, IBM, Apache, and Sun add even more SOAP support to their applications, operating systems and programming languages.

As the SOAP specification winds its way through the W3 standardization process, I'm certain that we will see changes. However, please don't let this stop you from experimenting and using SOAP in your applications. Yes, there will be changes, but these should be relatively minor, and each implementation should hide many of these details.

I first "met" Scott because of a mailing list—DevelopMentor's excellent list devoted to SOAP discussions (*http://discuss.develop.com/soap.html* if you're interested in joining). There he tirelessly helped others understand what he obviously thought as an important technology. Therefore, I was glad to hear that he was also working on this book. He has packed a great deal of practical development advice into these pages. I also love the fact that he shows a variety of the implementations available, and that they are all communicating nicely.

I hope that as you read this book, you see why Scott and I think SOAP is so important. So, whether you are a Java developer using the Apache implementation of SOAP, a VB developer using the Microsoft SOAP Toolkit, or a C# developer using .NET Web Services, or one of the many other implementations available, I hope that you join us in using SOAP in your applications. Perhaps together we can all learn to communicate.

Kent Sharkey

.NET Frameworks Technical Evangelist
Microsoft Corporation

Acknowledgments

It is amazing what one little protocol can do for an individual's life. In early 2000, Tim Moore at Prentice Hall wrote to a number of technical authors, including me, and asked what we thought of the Simple Object Access Protocol (SOAP). Never having heard of it, I went out and found the v0.9 specification and joined the SOAP mailing list hosted by DevelopMentor.[1] After completing my initial survey, I could not shake the feeling that SOAP was about to become something very big and important. By May 2000, I was signed up to write the book you now hold.

Writing a book on a new technology helps you understand how experts are created. You learn to understand the various nuances of what you can do with the protocol. Other standards may emerge to handle things the early adopters deem necessary (WSDL and UDDI emerged almost in response to SOAP's existence) and you learn those as well. If you participate in a discussion group and give good answers, you get noticed. For the few of you who subscribe to the DevelopMentor list, you know what happened to me. I gave a number of good answers and I open-sourced a fairly small SOAP engine (presented in Chapter 4). These actions resulted in Microsoft approaching me and eventually hiring me. Like I said, interesting things can happen to your life.

By coming over to Microsoft, I have been able to see how they do things on the inside with respect to SOAP. Right now, they really are concerned

[1] You can login their various lists by browsing over to *http://discuss.develop.com*.

about getting interoperability to work, both between their own products and between their products and other ones. I believe that the company will do the right thing and keep this attitude of making internal and external interoperability a priority. Inside, a lot of employees care about doing the right thing for everyone, not just for Microsoft.

I wrote this book during what seems to be the busiest year of my life (so far). I finished writing *Windows Shell Programming* in May 2000. Once that was done, I started in on this SOAP book. Besides that:

- I moved four times—twice in Wisconsin, twice in Washington.
- I interviewed and got a job writing for MSDN.
- My wife and I conceived and gave birth to Angeline (born March 15, 2001).

I could not have done all this without the support and help of my family. In between the sale of my house in Hartford, Wisconsin and my move to Oak Creek, Wisconsin, my grandfather let us move in for a bit. We all enjoyed spending that time together. I also have to thank my wife Jean and my two children, Vince and Angeline. They give me the strength and support to do amazing things. I have to thank my parents and my sister for believing in me. Finally, I want to thank Tim Moore and his staff at Prentice Hall. Thanks for cutting me some slack while writing this. My schedule slipped a number of times and they just took it in stride.

Contents

Chapter 1

HOW WE GOT TO SOAP

To understand why we need a technology such as the Simple Object Access Protocol (SOAP) we need to spend a bit of time looking at how computing technology has evolved. SOAP itself started out as a way to make distributed computing platform agnostic. We have always had the concept of distributed computing. The idea of having people perform calculations that they are good at and then handing the work off to other mathematicians is nothing new. For example, logarithms take a long time to compute. Because of this, people wrote out and reproduced logarithmic tables for other mathematicians to use.

To review the history, I would like to take a look at the things we have done in moving from the abacus to mechanical calculators and then to distributed computing. Understanding (or simply reviewing) this history gives some perspective of where we have come from and highlights why so many people are excited about SOAP. The idea of ubiquitous computing is moving from being just a neat idea to a reality. SOAP provides a way for all those computers to talk to each other and request services of each other. Indulge me as I present a little history lesson showing where our pursuit of automated number crunching has taken us.

The Abacus

The abacus has been used as a calculator for thousands of years, and you can still find it in use in China, Japan, and the Middle East. The most common form of the abacus can register numbers from 1 to 9,999,999,999,999.[1] The abacus does this using 13 rows of beads as shown in Figure 1–1. The user of an abacus reads the numbers from the beads touching the center bar. Each bead touching the center bar on the bottom half of the abacus equals one times the units column. Each bead touching the center bar from the top half of the abacus equals five times the units column. Figure 1–2 shows how you would represent the number 23.

The abacus shines when adding and subtracting numbers. Practiced users can usually outpace a person using a modern adding machine. As users add the second number in, they slide the beads up and down. Every time all five bottom beads touch the center bar, one bead from the same column on the top bar must come down. Then, all five beads must be returned to the bottom

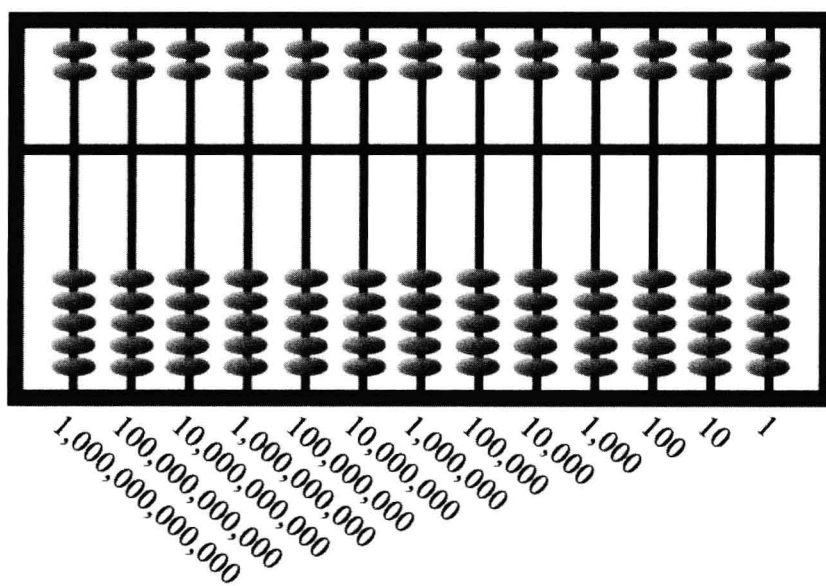

Figure 1–1 Thirteen–column abacus.

[1] This particular form of abacus has 13 rows. As a rule, an abacus can handle smaller or larger numbers depending on its construction. Smaller numbers need fewer rows—you could handle numbers through 9,999 with a four-row abacus. For each power of 10 that you want to handle, just add another row.

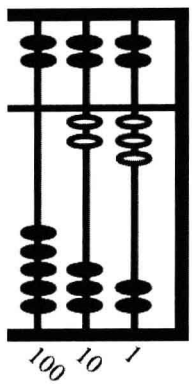

Figure 1–2
Representing 23 on the abacus (white beads).

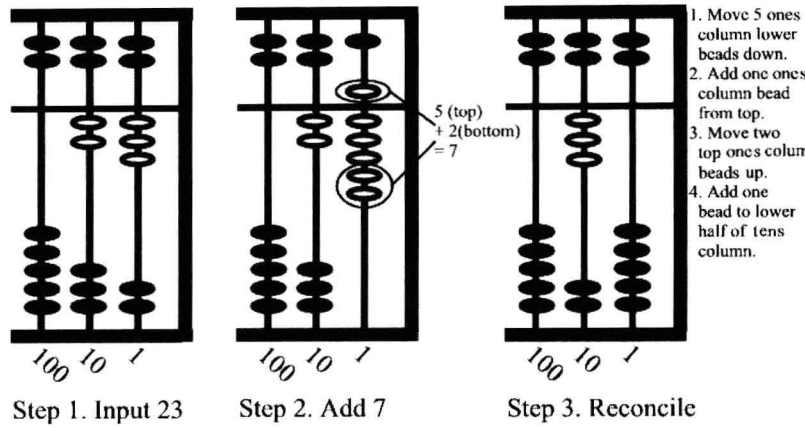

Figure 1–3 Adding 23 and 7 using abacus.

again. Likewise, if both top beads touch the center bar these beads must moved away from the center bar and one bottom bead from the next highest rank gets moved up. Figure 1–3 shows how one would execute 7 + 23 and reconcile that to 30. To subtract 7 from 30 and get 23 you would reverse the process.

Does that all make sense to you? Here is another way to look at the abacus. For this example, we use people and their fingers instead of beads to build a human abacus. After all, the abacus is based on this same idea. The bottom five beads represent the five fingers on a hand. The top two beads represent two hands. Each "hand" equals five "fingers." We use our human abacus like this: When all the fingers on one hand fill up, we start counting on the other hand. When the person in the ones column gets to 10, that person sets their fingers to zero and the person in the tens column remembers one on their hands (by raising one finger). This counting continues through the ranks until the capabilities of the fingers in the abacus are used up.

Because of the abacus's ability to aid in addition and subtraction, the tool has endured for a long time. Due to its construction it does not handle multiplication and division very well. Multiplication essentially involves adding the numbers over and over again (25 * 4 = 25 + 25 + 25 + 25). Division is also possible but time consuming. Not surprisingly, the abacus does not help us do any serious number crunching. It does allow for distribution of computing tasks. You may ask two or more people to manipulate the same series of numbers just to verify that the results are correct. Alternatively, you can also split