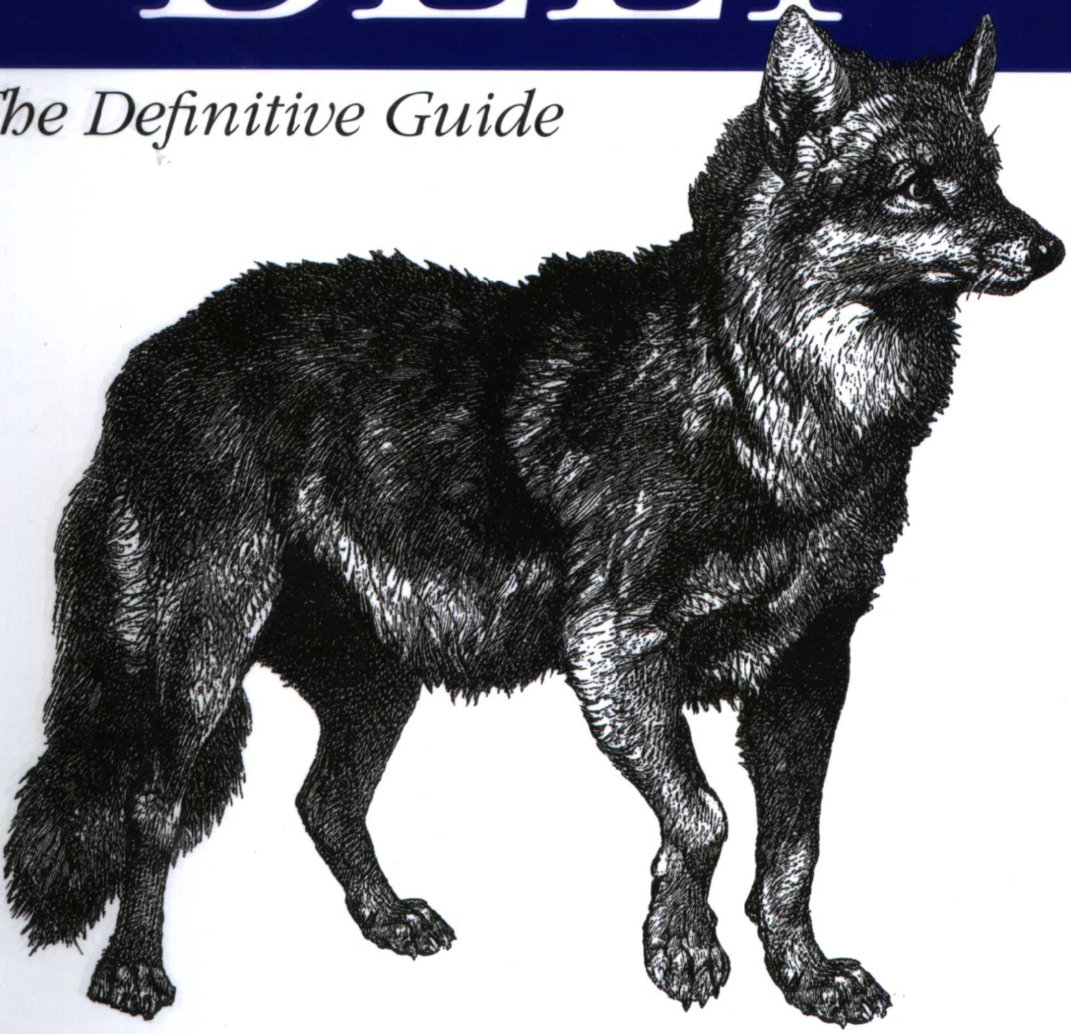
BEEP权威指南（影印版）

BEEP

The Definitive Guide



Marshall T. Rose 著

O'REILLY®

清华大学出版社

BEEP 权威指南（影印版）

BEEP: *The Definitive Guide*

Marshall T. Rose

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly & Associates, Inc. 授权清华大学出版社出版

清华大学出版社

图书在版编目（CIP）数据

BEEP 权威指南：/（美）罗斯（Rose, T. M.）著 — 影印版．— 北京：清华大学出版社，2002.11

书名原文：BEEP: The Definitive Guide

ISBN 7-302-05983-7

Ⅰ.B… Ⅱ.罗… Ⅲ.软件工具.BEEP Ⅳ.TP311.56

中国版本图书馆 CIP 数据核字（2002）第 078713 号

北京市版权局著作权合同登记

图字：01-2002-4487 号

书　　名/	BEEP 权威指南（影印版）
书　　号/	ISBN 7-302-05983-7/TP·3565
责任编辑/	冯志强
封面设计/	Ellie Volckhausen，张健
出版发行/	清华大学出版社（www.tup.tsinghua.edu.cn）
地　　址/	北京清华大学学研大厦（邮政编码 100084）
经　　销/	各地新华书店
印　　刷/	北京市艺辉印刷有限公司
开　　本/	787 毫米×980 毫米　16 开本　14 印张
版　　次/	2002 年 11 月第一版　2003 年 6 月第二次印刷
印　　数/	1501-2000 册
定　　价/	99.00 元（册）

O'Reilly & Associates公司介绍

O'Reilly & Associates 公司是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的*The Whole Internet User's Guide & Catalog*(被纽约公共图书馆评为20世纪最重要的50本书之一)到GNN(最早的Internet门户和商业网站),再到WebSite(第一个桌面PC的Web服务器软件)，O'Reilly & Associates 一直处于 Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly & Associates是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly & Associates 公司具有深厚的计算机专业背景，这使得O'Reilly & Associates形成了一个非常不同于其他出版商的出版方针。O'Reilly & Associates 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly & Associates 还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly & Associates 依靠他们及时地推出图书。因为O'Reilly & Associates 紧密地与计算机业界联系着，所以O'Reilly & Associates 知道市场上真正需要什么图书。

出版说明

计算机网络与通信技术的成熟和广泛应用，以及Internet与Web的迅速发展，为人类的工业生产、商业活动和日常生活都带来了巨大的影响。网络与通信技术在我国的很多领域也已经广泛应用，并且取得了巨大的效益。然而，该领域的技术创新的速度之快也是有目共睹的。为了帮助国内技术人员和网络管理人员在第一时间掌握国外最新的技术，清华大学出版社引进了美国O'Reilly & Associates公司的一批在计算机网络理论和应用方面代表前沿技术领域的著作，以飨读者。本套丛书采用影印版的形式，力求与国外图书"同步"出版，"原汁原味"地展现给读者各种权威技术理论和技术术语，适合于相关行业的高级技术人员、科研机构研究人员和高校教师阅读。

首批图书包括以下几种:

- 《Web 安全与电子商务》
- 《无线 Java 入门》
- 《Web 缓存》
- 《BEEP 权威指南》
- 《802.11 无线网络权威指南》
- 《大规模局域网设计》
- 《IP 路由》
- 《DNS 与 BIND》

Dedicated to my good friend, Lee

Foreword

In 1998, in the middle of what looked like a boom but turned out to be a bubble, I started a company with the modest goal of "mapping the Internet." Our company didn't become the next Yahoo!, but we followed the yellow brick road all the way into the magic kingdom, which unfortunately turned out to be made of sand hills.

Much of the heavy lifting for our company was done by three people. I got all lawyered up and spent 18 months writing financial doodles and honing our pitch. My soon-to-be wife and long-time colleague, Rebecca Malamud, became our creative director. And Marshall Rose wrote (and implemented) a protocol.

Our company did the "dot-com" thing. We built something pretty cool, raised lots of money, and hired professionals who were better at spending money than we were. The professionals proceeded to spend money, Becky and I moved on to co-found *betterdogfood.com*, and Marshall worked on his protocol. When the professionals were done, there was no money left and the company dissolved.

The part that lasted was the protocol. Marshall thought long and hard about the problems we were trying to solve and, as he has always done, decided to solve another problem. He invented BEEP. BEEP helped us solve our metadata management problems, but it's also a core that supports a wide range of applications. BEEP is an application layer framework, a long-missing building block for the Internet.

Once Marshall wrote (and implemented) the protocol, some of the best minds in the Internet were asked to review the work. It was then submitted to the IETF for standardization. This fundamental piece of technology has been a quiet revolution, used by a growing cadre of developers trying to build new kinds of applications without repeating old kinds of mistakes.

Marshall has always been one of the seminal thinkers on the Internet. He helped the Internet vanquish OSI by the then-novel technique of implementing the specs and showing that they tried to solve so many problems that they solved none. His work on network management and messaging helped make those two crucial applications work. He was releasing open source software long before the term was invented. And, with BEEP, Marshall has shown us a better way to write network applications.

—Carl Malamud
San Francisco, California

Preface

BEEP is something like "the missing link between the application layer and TCP."

This statement is a horrific analogy because TCP is a transport *protocol* that provides reliable connections, and it makes no sense to compare a protocol to a layer. TCP is a highly-evolved protocol; many talented engineers have, over the last 20 years, built an impressive theory and practice around TCP. In fact, TCP is so good at what it does that when it came to survival of the fittest, it obliterated the competition. Even today, any serious talk about the transport protocol revolves around minor tweaks to TCP. (Or, if you prefer, the intersection between people talking about doing an "entirely new" transport protocol and people who are clueful is the empty set.)

Unfortunately, most application protocol design has not enjoyed as excellent a history as TCP. Engineers design protocols the way monkeys try to get to the moon—i.e., by climbing a tree, looking around, and finding another tree to climb. Perhaps this is because there are more distractions at the application layer. For example, as far as TCP is concerned, its sole reason for being is to provide a full-duplex octet-aligned pipe in a robust and network-friendly fashion. The natural result is that while TCP's philosophy is built around "reliability through retransmission," there isn't a common mantra at the application layer.

Historically, when different engineers work on application protocols, they come up with different solutions to common problems. Sometimes the solutions reflect differing perspectives on inevitable tradeoffs; sometimes the solutions reflect different skill and experience levels. Regardless, the result is that the wheel is continuously re-invented, but rarely improved.

So, what is BEEP and how does it relate to all this? BEEP integrates the best practices for common, basic mechanisms that are needed when designing an application protocol over TCP. For example, it handles things like peer-to-peer, client/server, and server/client interactions. Depending on how you count, there are about a dozen or so issues that arise time and time again, and BEEP just deals with them. This means that you get to focus on the "interesting stuff."

BEEP has three things going for it:

- It's been standardized by the IETF, the so-called "governing body" for Internet protocols.
- There are open source implementations available in different languages.
- There's a community of developers who are clueful.

The standardization part is important, because BEEP has undergone a lot of technical review. The implementation part is important, because BEEP is probably available on a platform you're familiar with. The community part is important, because BEEP has a lot of resources available for you.

The Intended Audience

This book is not for everyone. It is written with two audiences in mind:

- Designers who want to understand how BEEP works and when to use it
- Developers who want to use one of the open source APIs for BEEP

Please note that there are two market segments excluded from this list:

- Administrators who want to understand what's being used in their networks
- Developers who want to write a BEEP library

This book doesn't focus on administrators because the open source APIs for BEEP don't have much to offer the administrator. In time, that will likely change, but for now, there just isn't much to write about.

For the second audience, if you're going to write an API, this book will help by providing the context for what your customer expects, but you'll probably have a lot of questions that this book won't answer. Why is that?

In brief, it's a lot simpler to use an API for BEEP than to write one. Although most of BEEP's concepts are straightforward, there are a lot of interactions between them that make for some tricky implementation strategies. Considering that there aren't thousands of API developers for BEEP, you can see why it's not a good idea to clutter this book with that kind of detail.

However, assuming that you want to develop an API for BEEP, here's my gift to you—a list of issues that your API should transparently handle:

- Encoding piggyback data during channel creation
- Avoiding race conditions when closing a channel
- Enforcing the bidirectional "at most once" limitation on using a SASL mechanism with an active security layer or a transport security profile

- Enforcing the unidirectional "at most once" limitation on user authentication
- Making sure that stuff sent by the transport mapping doesn't interfere with transport security negotiations

Once you know about stuff like this, developing a robust implementation is a simple matter of coding. However, my point is that this book *does not* talk about those kinds of issues.

In fact, this book is a bit different than most of the books in the O'Reilly series, in that the concepts/software ratio is about 60/40. (I think that the typical O'Reilly ratio is closer to 20/80.) So keep this in mind.

Conventions Used in This Book

The following font conventions are used in this book:

Italic is used to introduce new terms and for URLs and filenames.

`Constant Width` is used to indicate code sections and for methods, objects, interfaces, class names, and package names.

This icon indicates a tip, suggestion, or general note.

This icon indicates a warning or caution.

We'd Like to Hear from You

Please address comments and questions concerning this book to the publisher:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/beep

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Acknowledgments

There's a long list of people whose work led to this book.

The original work leading up to BEEP was done at Invisible Worlds, a company founded by Carl Malamud. Carl gets credit for providing the initial problem to get solved, and allowing the solution to be presented to the IETF. The early work on BEEP was reviewed by the Invisible Worlds' Protocol Advisory Board, who, at that time, consisted of David Clark, David Crocker, Steve Deering, Danny Goodman, Paul Mockapetris, and Paul Vixie. Also during that time Brad Burdick and Frank Morton implemented the two initial prototypes of BEEP's predecessor.

In the standards world, Keith McCloghrie, one of my long-time collaborators on network management, was kind enough to chair the IETF working group for BEEP. David Crocker, Graham Klyne, and Darren New provided most of the heavy lifting in the working group and in many BEEP-related activities. Pete Resnick now chairs the follow-on working group for BEEP.

Back in the real world, Huston Franklin led the group (consisting of Eric Dixon, Jay Kint, Bill Mills, Scott Pead, and Mark Richardson) that produced two different implementations of BEEP. Darren New, as Free Radical at Invisible Worlds, was responsible for design and implementation of the "core" of beepcore-c, while Kris Magnusson developed and managed the *http://beepcore.org* community web site. Huston was also kind enough to rework a lot of the beepcore-c API in response to my first draft of this book. The resulting API is a lot cleaner. (Of course, I then had to rewrite the chapter, thereby proving the old adage, "In the hell that is the Internet, sinners get exactly what they ask for.")

Finally, David Crocker, Huston Franklin, Bruce Mitchener, Darren New, Chris Newman, and Pete Resnick were gracious in their comments as reviewers.

Personal Notes

This is the ninth book I've written, and the first for O'Reilly & Associates. If you're familiar with my earlier works, you may find the following of interest:

- There are no soapboxes or insider index entries in this book, so don't bother looking—I decided not to renew my liability coverage.
- Cheetah, the alley cat, is now 17 years old and still doing well.
- He is joined by a 3-year-old, 220-pound Mastiff named Oatman.

Or not, as the case may be. Perhaps the more interesting question is, why did I stop writing books in 1998?

The short answer is that nothing much has happened in the Internet since then—well, nothing interesting of a technical nature anyway. Although the Internet got a lot of press, and small fortunes were made and large ones were lost, none of this was due to technology. T.A. Edison was a brilliant businessman and engineer, yet he never once applied for a patent on a business process. The things he did patent were advances in technology. And this, perhaps, provides a concise explanation of the Internet boom and bust. For myself, I'll confess to having "done" a few start-ups, and having modestly reduced the performance of several venture funds. While few of these experiences were good ones, meeting a whole new class of gentry was certainly educational.

Finally, if you've gotten this far, you've noticed that my writing style is different from what you find in most of the books in O'Reilly's excellent series. (Perhaps the pet update was a tip-off.) While this causes a lot of grief to some, I think it makes my writing more readable. At the very least, it makes it more writable. Enjoy.

—Marshall T. Rose
Sacramento, California
November, 2001

Table of Contents

CHAPTER 1

Introduction

An application protocol is a set of rules that says how your application talks to the network. Over the last few years, HTTP has been pressed into service as a general-purpose application protocol for many different kinds of applications, ranging from the Internet Printing Protocol (IPP) to SOAP. This is great for application designers: it saves them the trouble of having to design a new protocol and allows them to reuse a lot of ideas and code.

HTTP has become the reuse platform of choice, largely because:

- It is familiar.
- It is ubiquitous.
- It has a simple request/response model.
- It usually works through firewalls.

These are all good reasons, and—if HTTP meets your communications requirements—you should use it. The problem is that the widespread availability of HTTP has become an excuse for not bothering to understand what the requirements really are. It's easier to use HTTP, even if it's not a good fit, than to understand your requirements and design a protocol that does what you really need.

That's where BEEP comes in. It's a toolkit that you can use for building application protocols. It works well in a wide range of application domains, many of which weren't of interest when HTTP was being designed.

BEEP's goal is simple: you, the protocol designer, focus on the protocol details for your problem domain, and BEEP takes care of the other details. It turns out that the vast majority of application protocols have more similarities than differences. The similarities primarily deal with "administrative overhead"—things you need for a working system, but aren't specific to the problem at hand. BEEP mechanizes the similar parts, and lets you focus on the interesting stuff.

Application Protocol Design

Let's assume, for the moment, that you don't see a good fit between the protocol functions you need and either the email or the Web infrastructures. (We'll talk more about this later on in the section "The Problem Space.") It's time to make something new.

First, you decide that your protocol needs ordered, reliable delivery. This is a common requirement for most application protocols, including HTTP and SMTP. The easiest way to get this is to layer the protocol over TCP.*

So, you decide to use TCP as the underlying transport for your protocol. Of course, TCP sends data as an octet stream—there aren't any delimiters that TCP uses to indicate where one of your application's messages ends and another one begins. This means you have to design a framing mechanism that your application uses with TCP. That's pretty simple to do—HTTP uses an octet count and SMTP uses a delimiter with quoting.

Since TCP is just sending bytes for you, you need to not only frame messages but also have a way of marking what's in each message (e.g., a data structure, an image, some text, and so on). This means you have to design an encoding mechanism that your application uses with the framing mechanism. That's also pretty simple to do—HTTP and SMTP both use something called MIME (which you can find out about in *Programming Internet Email*).

Back in the early 80s, when I was a young (but exceptionally cynical) computer scientist, my advisor told me that protocols have two parts: data and control. It looks like the data part is taken care of with MIME, so it's onto the control part. If you are fortunate enough to know ahead of time every operation and option that your protocol will ever support, there's no need for any kind of capabilities negotiation. In other words, your protocol doesn't need anything that lets the participants tell each other which operations and options are supported. (Of course, if this is the case, you have total recall of future events, and really ought to be making the big money in another, more speculative, field.)

The purpose of negotiation is to find common ground between two different implementations of a protocol (or two different versions of the same implementation). There are lots of different ways of doing this and, unfortunately, most of them don't work very well. SMTP is a really long-lived, well-deployed protocol, and it seems to do a pretty good job of negotiations. The basic idea is for the server to tell the client what capabilities it supports when a connection is established, and then for the client to use a subset of that.

* If you're not familiar with these acronyms, you'll need to consult some books on Internet basics, such as *Internet Core Protocols: The Definitive Guide* by Eric Hall for TCP, *HTTP Pocket Reference* by Clinton Wong for HTTP, and *Programming Internet Email* by David Wood for SMTP. (Of course, since you're designing an application protocol, presumably you're already familiar with the protocols behind these acronyms.)