Advanced .NET Remoting

INGO RAMMER

Apress™

Advanced .NET Remoting
Copyright ©2002 by Ingo Rammer

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-025-2
Printed and bound in the United States of America 12345678910
Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.
Technical Reviewer: Kent Sharkey
Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore, Karen Watterson, John Zukowski
Managing Editor: Grace Wong
Project Manager: Alexa Stuart
Copy Editor: Ami Knox
Production Editor: Julianna Scott Fein
Compositor and Illustrator: Impressions Book and Journal Services, Inc.
Indexer: Valerie Haynes Perry
Cover Designer: Tom Debolski
Marketing Manager: Stephanie Rodriguez

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.
In the United States, phone 1-800-SPRINGER, Email `orders@springer-ny.com`, or visit `http://www.springer-ny.com`.
Outside the United States, fax +49 6221 345229, Email `orders@springer.de`, or visit `http://www.springer.de`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710.
Email info@apress.com, or visit http://www.apress.com.
The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.
The source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 1

Introduction to Remoting

THIS CHAPTER GIVES YOU a short introduction to the world of distributed application development and its respective technologies. Here you get a chance to examine some scenarios in which .NET Remoting can be employed and learn some historical background on the progress and development of various remoting frameworks during the last ten years.

What Is Remoting?

Remoting is the process of programs or components interacting across certain boundaries. These contexts will normally resemble either different processes or machines.[1] In the .NET Framework, this technology provides the foundation for distributed applications—it simply replaces DCOM.

Remoting implementations generally distinguish between *remote objects* and *mobile objects*. The former provides the ability to execute methods on remote servers, passing parameters and receiving return values. The remote object will always "stay" at the server, and only a reference to it will be passed around among other machines.

When mobile objects pass a context boundary, they are serialized (marshaled) into a general representation—either a binary or a human readable format like XML—and then deserialized in the other context involved in the process. Server and client both hold copies of the same object. Methods executed on those copies of the object will always be carried out in the local context, and no message will travel back to the machine from which the object originated. In fact, after serialization and deserialization, the copied objects are indistinguishable from regular local objects, and there is also no distinction between a server object and a client object.

[1] .NET extends this concept to include the ability to define additional contexts within one running application. Object accesses crossing these boundaries will pass the .NET Remoting Framework as well.

Scenarios for .NET Remoting

At the beginning of the client/server era, remoting was mostly used for accessing a server's resources. Every database or file server is an implementation of some technique that allows code to be executed remotely. Programming these older frameworks was so difficult a task that few products except for these server-side core services implemented remoting.

Nowadays the building of distributed applications has gotten a lot easier so that it's quite feasible to distribute business applications among various machines to improve performance, scalability, and maintainability.

Centralized Business Logic

One of the key scenarios for implementing remoting is the concentration of business logic on one or more central servers. This considerably simplifies the maintainability and operability of large-scale applications. Changes in business logic do not entail your having to roll out an application to your organization's 10,000 worldwide users—you just have to update one single server.

When this centralized business logic is shared among different applications, this labor-saving effect multiplies considerably; instead of patching several applications, you just have to change the server's implementation.

Physical Separation of Layers

The security of a company's vital databases represents a common concern in this time of Web-enabled businesses. The general recommendation is against directly connecting from the Web server to the database because this setup would allow attackers easy access to critical data after they have seized control of the Web server.

Instead of this direct connection, an intermediate application server is introduced. This server is placed in a so-called demilitarized zone (DMZ), located between two firewalls. Firewall #1 only allows connections from the Web server to the app server, and Firewall #2 only allows connections from the app server to the databases.

Because the application server doesn't allow the execution of arbitrary SQL statements, yet provides object-oriented or function-based access to business logic, a security compromise of the Web server (which can only talk to the app server) is noncritical to a company's operations.

Accessing Other Platforms

In today's mid- to large-scale enterprises, you will normally encounter a heterogeneous combination of different platforms, frameworks, and programming languages. It is not uncommon to find that a bunch of tools have been implemented: Active Server Pages (ASP), Java Server Pages (JSP), PHP, or ColdFusion for Web applications, Visual Basic or Java for in-house applications, C++ for server-side batch jobs, scripting languages for customizing CRM systems, and so on.

Integrating these systems can be a daunting task for system architects. Remoting architectures like CORBA, SOAP, and .NET Remoting are an absolute necessity in large-scale enterprise application integration. (CORBA and SOAP are introduced and compared later in this chapter.)

Third-Party Access

Opening systems to third parties in a business-to-business environment is quite common nowadays. This process started with hard-to-implement EDI documents, transferred via proprietary networks, and is recently opening up for smaller companies due to the possibility of using SOAP, which is fairly easier to implement.

Order-entry applications, which allow your business partners to directly place orders from one ERP system to the other, constitute one example of an application utilizing this kind of remoting. More sophisticated applications are starting to be developed—address verification, customer creditworthiness ratings, and online price-comparison systems are just the beginning.

Evolution of Remoting

The scenarios presented thus far have only been possible due to the constant evolution of remoting frameworks. The implementation of large-scale business applications in a distributed manner has only been practicable after the technical problems have been taken care of by the frameworks. CORBA, COM+, and EJB started this process several years ago, and .NET Remoting simplifies this process even more.

To underscore how far remoting has evolved from its cumbersome beginnings, the following sections give you a brief history of the various remoting frameworks.

DCE/RPC

Distributed Computing Environment (DCE), designed by the Open Software Foundation (OSF) during the early 1990s, was created to provide a collection of tools and services that would allow easier development and administration of distributed applications. The DCE framework provides several base services such as Remote Procedure Calls (DCE/RPC), Security Services, Time Services, and so on.

Implementing DCE is quite a daunting task; the interfaces have to be specified in Interface Definition Language (IDL) and compiled to C headers, client proxies, and server stubs by an IDL compiler. When implementing the server, one has to link the binary with DCE/Threads, which are available for C/C++. The use of programming languages other than these is somewhat restricted due to the dependence on the underlying services, like DCE/Threads, with the result that one has to live with single-threaded servers when refraining from using C/C++.

DCE/RPC nevertheless is the foundation for many current higher-level protocols including DCOM and COM+. Several application-level protocols such as MS SQL Server, Exchange Server, Server Message Block (SMB), which is used for file and printer sharing, and Network File System (NFS) are also based on DCE/RPC.

CORBA

Designed by the Object Management Group (OMG), an international consortium of about 800 companies, CORBA's aim is to be the middleware of choice for heterogeneous systems. OMG's CORBA, which stands for *Common Object Request Broker Architecture*, is only a collection of standards; the implementation of object request brokers (ORBs) is done by various third parties. Because parts of the standard are optional and the vendors of ORBs are allowed to include additional features that are not in the specifications, the world has ended up with some incompatible request brokers. As a result, an application developed to make use of one vendor's features could not easily be ported to another ORB. When you buy a CORBA-based program or component, you just can't be sure if it will integrate with your CORBA applications, which probably were developed for a different request broker.

Aside from this potential problem, CORBA also has quite a steep learning curve. The standard reads like a complete wish list of everything that's possible with remoted components—sometimes it simply is too much for the "standard business." You'll probably end up reading documents for days or weeks before your first request is ever sent to a server object.

Nevertheless, when you have managed to implement your first CORBA application, you'll be able to integrate a lot of programming languages and platforms. There are even layers for COM or EJB integration, and apart from SOAP, CORBA is the only true multiplatform, multiprogramming language environment for distributed applications.

DCOM

Distributed Component Object Model (DCOM) is an "extension" that fits in the Component Object Model (COM) architecture, which is a binary interoperability standard that allows for component-oriented application development. You'll usually come in contact with COM when using ActiveX controls or ActiveX DLLs.

DCOM allows the distribution of those components among different computers. Scalability, manageability, and its use in WANs pose several issues that need to be addressed. DCOM uses a pinging process to manage the object's lifetimes; all clients that use a certain object will send messages after certain intervals. When a a server receives these messages it knows that the client is still alive; otherwise it will destroy the object.

Additionally, reliance on the binary DCE/RPC protocol poses the need for direct TCP connections between the client and its server. Use of HTTP proxies is not possible. DCOM is available for Microsoft Windows and for some UNIX dialects (ported by the German Software AG).

MTS/COM+

COM+, formerly *Microsoft Transaction Server* (MTS), was Microsoft's first serious attempt to reach into the enterprise application domain. It not only serves as a remoting platform, but also provides transaction, security, scalability, and deployment services. COM+ components can even be used via Microsoft Message Queue Server to provide asynchronous execution of methods.

Despite its advantages, COM+ does not yet support the automatic marshalling of objects to pass them by value between applications; instead you have to pass your data structures using ADO recordsets or other means of serialization. Other disadvantages that keep people from using COM+ are the somewhat difficult configuration and deployment, which complicates its use for real-world applications.

Java RMI

Traditional *Java Remote Method Invocation* (Java RMI) uses a manual proxy/stub compilation cycle. In contrast to DCE/RPC and DCOM, the interfaces are not written in an abstract IDL but in Java. This is possible due to Java being the only language for which the implementation of RMI is possible.

This limitation locked RMI out of the game of enterprise application integration. Even though all relevant platforms support a Java Virtual Machine, integration with legacy applications is not easily done.

Java EJB

Enterprise Java Beans (EJB) was Sun's answer to Microsoft's COM+. Unlike CORBA, which is only a standard, EJB comes with a reference implementation. This allows developers to check if their products run in any standard-complying EJB container. EJB has been widely accepted by the industry, and there are several container implementations ranging from free open source to commercial implementations by well-known middleware vendors.

One problem with EJB is that even though a reference implementation exists, most vendors add features to their application servers. When a developer writes a component that uses one of those features, the application will not run on another vendor's EJB container.

Former versions of EJB have been limited to the Java platform because of their internal reliance on RMI. The current version allows the use of IIOP, which is the same transfer protocol CORBA uses, and third parties already provide commercial COM/EJB bridges.

Web Services/SOAP/XML-RPC

Web Services provided the first easy to understand and implement solution to true cross-platform and cross-language interoperability. Web Services technically are stateless calls to remote components via HTTP POST with a payload encoded in some XML format.

Two different XML encodings are currently in major use: XML-RPC and SOAP. *XML-RPC* can be described as a poor man's SOAP. It defines a very lightweight protocol with a specification size of about five printed pages. Implementations are already available for a lot of programming environments, ranging from AppleScript to C/C++, COM, Java, Perl, PHP, Python, Tcl, and Zope—and of course there's also an implementation for .NET.

SOAP, or *Simple Object Access Protocol*, defines a much richer set of services; the specification covers not only remote procedure calls, but also the *Web*

Services Description Language (WSDL) and *Universal Description, Discovery, and Integration* (UDDI). WSDL is SOAP's interface definition language, and UDDI serves as a directory service for the discovery of Web Services. Those additional protocols and specifications are also based on XML, which allows all SOAP features to be implemented on a lot of platforms.

The specifications and white papers for SOAP, WSDL, UDDI, and corresponding technologies cover several hundred pages, and you can safely assume that this document will grow further when topics like routing and transactions are addressed. Fortunately for .NET developers, the .NET platform takes care of *all* issues regarding SOAP.

.NET Remoting

At first look, .NET Remoting is to Web Services what ASP has been to CGI programming. It takes care of a lot of issues for you: contrary to Web Services, for example, .NET Remoting enables you to work with stateful objects. This single fact allows it to be the base of tomorrow's distributed applications.

In addition to the management of stateful objects, .NET Remoting gives you a flexible and extensible framework that allows for different transfer mechanisms (HTTP and TCP are supported by default), encodings (SOAP and binary come with the framework), and security settings (IIS Security and SSL come out of the box).

With these options, and the possibility of extending all of them or providing completely new implementations, .NET Remoting is well suited to today's distributed applications. You can choose between HTTP/SOAP for the Internet or TCP/binary for LAN applications by literally changing a single line in a configuration file.

Interface description does not have to be manually coded in any way, even though it's supported if you like to design your applications this way. Instead, metadata can be extracted from running servers, where the WSDL is automatically generated, or from any .NET assembly.

Summary

This chapter provided a short introduction to the world of distributed application development and the respective technologies. You now know about the various scenarios in which .NET Remoting can be applied and understand how it differs from other distributed application protocols and techniques.

Index

NUMBERS

2xx-5xx SMTP response code classes, explanations of, 318

A

abstract base classes
- creating in shared assemblies, 78
- role in sample multiserver configuration, 70, 72

<activated> property, role in configuration files, 107–108, 110

Activator.GetObject() call
- calling for nonwrapped proxy metadata, 84–85
- examining in MarshalByRefObject in Singleton mode, 21
- using with client in factory design pattern, 46–47

algorithms for encryption, producing with key generators, 269–271

anonymous use, deploying server applications for, 120–123

APOP command, RFC for, 320

architecture of .NET Remoting, illustration of, 10

assemblies
- displaying strong names for, 166
- types of, 15

assemblyBinding entry in configuration files, resolving assembly versions with, 173–174

AssemblyVersion attribute, changing for versioned SAOs, 169

asynchronous calls
- example of, 58–60
- explanation of, 52
- first stage of, 227
- role in compression sinks, 254
- using, 179–185

asynchronous delegates
- advisory about implementation of, 180–181
- advisory about using events with, 187, 190
- declaring for asynchronous calls, 58–59
- example of, 60–61
- explanation of, 58
- invoking without [OneWay] methods, 197–198
- managing with .NET Remoting messages, 340–341
- shipping destination assemblies to callers for, 191–196
- using for asynchronous calls, 179
- using with new clients, 61–63
- using with SoapSuds-generated DLLs, 180–181
- using with wrapper methods, 184–185

asynchronous IClientChannelSink processing, performing, 224–226

asynchronous IMessageSink processing, performing, 222–224

asynchronous messaging, performing, 222

asynchronous one-way calls, example of, 63–67

asynchronous requests, generating, 226–230

asynchronous responses, handling, 230–233

AsyncProcessMessage() function, implementing, 287

AsyncProcessResponse() function
- role in creating sinks for clients, 352–354
- role in server sinks, 364
- sample implementation for server-side asynchronous processing, 235
- using with compression sinks, 257–258, 264–265

AsyncResponseHandler class, implementing for client sinks, 354–355
attributes
  for <channels> tag in configuration files, 101
  for <lifetime> tag in configuration files, 99
  for HTTP channels, 101
authenticated use, deploying for, 125–133
authentication entries, checking for URLs of current messages, 302–305
authentication methods, configuring for anonymous-use deployments, 120–121

B
base objects, role in .NET Remoting, 12
Base64, RFC for, 321
BaseRemoteObject, role in sample multiserver configuration, 73
Basic Authentication scheme
  advisory about, 125
  enabling, 132
  explanation of, 131
BeginInvoke() method, using with delegates, 179
binary encoding via HTTP, using with configuration files, 105
BinaryFormatter, introduction to, 211
BinaryServerFormatterSink, explanation of, 219
bindTo attribute, role in HTTP channels for configuration files, 102
BroadcastEventWrapper, using with shared assemblies, 192–193
BroadcastMessage() method,invoking for refactored event handling, 194–195
business logic
  checking constraints for, 391
  documenting at metadata level, 378
  rejecting constraints for, 390
business rules for servers, implementing for MarshalByRefObject in Singleton mode, 23–24
byte arrays, encoding with Base 64, 321
ByValue objects
  explanation of, 14
  using, 27–28

C
C# code
  generating with SoapSuds, 94
  for versioning client-activated objects, 172–173
calls, intercepting with ContextAttributes, 379
CAOs (client-activated objects)
  enabling for server-side channels, 360
  explanation of, 38
  implementing different lifetimes for, 49–50
  and ObjRef object, 204, 206–207
  versus SAOs, 206
  specifying in server-side configuration files, 107–108
  using <client> property with, 109
  using with server-side sponsors, 160
chains of providers
  creating sinks from, 241
  role in client-side sinks, 239
chains of sinks, defining in configuration files, 237–238
channels
  examining basic implementation of, 347–348
  registering for clients in MarshalByRefObject in Singleton mode, 20
  returning base URLs for, 359
  wrapping, 366–369
<channels> tag, using with configuration files, 100–102
character encoding, essentials of, 321
classes
  locating, 137
client-activated objects, versioning, 172–174
client assemblies, explanation of, 15
client channels, implementing, 346–350
client-side keepalive thread, starting, 160
client-side messaging. *See also* messages, server-side messaging
  examples of, 214–216
  versus server-side messaging, 217
client-side PriorityEmitterSink code, 289–292
client-side sink providers, creating, 258–260
client-side sinks. *See also* server-side sinks, sinks
  configuring for encryption, 281
  creating, 239–243
  diagram with compression sink, 250
  implementing for compression sinks, 251–255
  versus server-side sinks, 257
client-side sponsors
  advisory about, 149, 155
  registering to avoid premature termination of objects, 154–155
  using, 150–155, 150–155

client-side synchronous message handling, diagram of, 205
<client> property, role in configuration files, 108–109
ClientChannelSinkStack
  appearance before call to SOAP formatter, 233
  appearance before first sink is called, 232
  contents in asynchronous requests, 229
clientConnectionLimit attribute
  role in HTTP channels for configuration files, 101
  using with delegates, 179
ClientContextTerminatorSink, explanation of, 215
ClientProviders and ServerProviders properties, role in configuration files, 102–106
clients
  accessing client-activated objects with, 41
  advisory about distributing SAO implementations to, 44
  advisory when using events with, 187
  building for versioned SAOs, 167
  calling timed-out CAO with, 50–51
  creating for synchronous calls, 55–57
  creating providers for, 350–355
  creating sinks for, 350–355
  developing for versioned SAOs, 170–172
  enabling connections to assemblies, 170
  example for SingleCall SAO, 32
  implementing for asynchronous one-way calls, 64–67
  implementing for MarshalByRefObject in Singleton mode, 20–22, 24–25
  implementing for nonwrapped proxy metadata, 84–86
  implementing for SoapSuds-generated metadata, 81–82
  implementing with asynchronous calls, 61–63
  pinging server-side sponsors with, 157–158
  role in configuration files, 96–98
  role in sample multiserver configuration, 70–71
  using binary encoding via HTTP with, 105
  using factory design pattern with, 46–47
  using wrapper functions with, 184
command-line operations, advisory about, 40, 163
compressed streams, creating class for, 253–254
compression library, Web site for, 251
compression of request streams, verifying, 263
compression sinks
  creating, 250–251
  extending, 263–268
  implementing client-side sinks for, 251–255
configuration filenames, defaults for, 89
configuration files
  benefits of, 87
  changing after using SoapSuds -gc with SAO, 182
  defining chain of sinks in, 237–238
  effect of typos on, 97
  example for lifetime management, 147–148
  placing sinks in, 261–262
  role of <client> property in, 108–109
  role of versioning behavior in, 104–105
  using, 88–89, 95–98
  using <activated> property with, 107–108, 110
  using <channels> tag with, 100–102
  using <lifetime> tag with, 99
  using <service> property with, 106
  using <wellknown> property with, 106–107, 109–110
  using binary encoding via HTTP with, 105
  using formatters and providers with, 103–104
  using HTTP channel with, 101–102
  using with Windows services, 115
configuration options, standards for, 99–102
console applications, deploying servers as, 111
ConstructionCall messages versus MethodCall, 207
content replies, role in POP3 conversions, 320
ContextAttribute, intercepting calls with, 379
ContextBoundObject class code, 383–384
contexts
  creating, 378–387
  explanation of, 246
CORBA (Common Object Request Broker Architecture) remoting framework, evolution of, 4–5

core modules, preparing for transport-channel examples, 372
CreateMessageSink()
  calling on client-side channels, 242
  role in implementing client channels, 349
CreateSink() method, implementing, 258–260
CrossContextChannel, explanation of, 220
csc.exe command-line compiler, advisory about, 94
custom proxies, using, 308–315
Customer Class, creating for MarshalByRefObject in Singleton mode, 16–17
Customer data, validating, 22–25

D
D units for <lifetime> tag attributes, explanation of, 99, 137
DATA code, role in SMTP, 319
data objects, defining for MarshalByRefObject in Singleton mode, 16–17
data serialization, role in .NET Remoting, 13
DateTime instance variables, benefits of adding to server-side sponsors, 156
DCE/RPC (Distributed Computing Environment)/Remote Procedure Calls remoting framework, evolution of, 4
DCOM (Distributed Component Object Model) remoting framework, evolution of, 5
DELE *message_number* command, role in POP3, 321
delegates
  advisory about implementation of, 180–181
  advisory about using events with, 187, 190
  declaring for asynchronous calls, 58–59
  example of, 60–61
  explanation of, 58
  invoking without [OneWay] methods, 197–198
  managing with .NET Remoting messages, 340–341
  shipping destination assemblies to callers for, 191–196
  using for asynchronous calls, 179
  using with new clients, 61–63
  using with SoapSuds-generated DLLs, 180–181
  using with wrapper methods, 184–185
DeleteMessage() method, using with POP3, 330
dictionary keys for sample method call message, list of, 209
DispatchChannelSink, explanation of, 220
displayName attribute
  role in <channels> tag for configuration files, 101
  using with <client> property for configuration files, 109
  using with <wellknown> property for configuration files, 107, 109–110
distributed reference counting, explanation of, 135
.dll extension, advisory regarding configuration files, 97
DLLs (dynamic link libraries), using with SAOs, 29
DomainSpecificRemotingData, using with ServerContextTerminatorSin, 221
dynamic sinks
  functionality of, 215
  using, 246–247

E
-ERR messages, using with POP3, 320
e-mail addresses, parsing for incoming requests, 345
e-mail headers, creating, 322–323
e-mail, types of, 341
ease of implementation, advantage of .NET remoting, 9–10
encrypted HTTP traffic, displaying TCP trace of, 285
encrypted use, deploying for, 133–134
encryption helpers, creating, 272–274
EncryptionClientSink versus EncryptionServerSink, 274–280
EncryptionClientSinkProvider code, 282–283
EncryptionServerSinkProvider code, 283–285
EndInvoke() method in asynchronous calls
  advisory about, 60, 63
  using with delegates, 179
event handling, refactoring, 191–196

EventInitiator, calling BroadcastMessage() method with, 195
events
    ensuring optimal performance with, 198
    explanation of, 185–187
    problems with, 187–191
exception handling, performing during IClientChannelSink processing, 233
.exe extension, advisory regarding configuration files, 97
expired objects, calling methods for, 150–155
ExtendedMBRObject, role in lifetime management example, 145–148, 158
extensible architecture of .NET remoting, advantage of, 10–11

F
factory design pattern, using, 42–48
formatters
    role in configuration files, 103–104
    role in messages, 207
    serializing Message objects through, 211–212
Freeze() method, role in IContextProperty interface, 380

G
GAC (global assembly cache)
    contents after installing assemblies for versioned SAOs, 170
    manipulating contents of, 164
    role in exposing MarshalByRefObject in Singleton mode, 17, 19
general assemblies
    explanation of, 15
group memberships, verifying for users, 128–130

H
H units for <lifetime> tag attributes, explanation of, 99, 137
HashTables in SMTPHelper class, types of, 336–337
HELO *localhost* code, role in SMTP, 319
HTTP Basic Authentication scheme
    advisory about, 125
    enabling, 132
    explanation of, 131
HTTP channels
    attributes for, 101
    creating for MarshalByRefObject in Singleton mode, 18, 20
    specifying for port 1234, 102
    wrapping channels with, 366
HTTP headers for SOAP remoting call, example of, 212–213
HTTP requests, compressing, 268
HTTP responses, compressing, 268
HTTP/SOAP connection, example of TCP trace without using sinks, 262
HttpClientChannel, functionality of, 216
HttpServerChannel
    default sink chain for, 244
    explanation of, 218
HttpServerTransportSink, explanation of, 218

I
IChannel and IChannelReceiver code, role in implementing server channels, 356–357
IChannel and IChannelSender code, 347–349
IChannel, implementing server methods and properties for, 359
IChannel with populated sink providers, example of, 239–240
IChannelSinkBase and IServerChannelSink code, 361
IClientChannelSink interface for message links
    advisory about, 307
    example of, 210–211
    performing exception handling for, 233
    role in implementing compression sinks, 251–253
IContextProperty interface code, implementing, 380–381
IContributeDynamicSink property, displaying, 246
IContributeObjectSink interface code, 380
ICustomerManager
    extending for MarshalByRefObject in Singleton mode, 22–23
    implementing for MarshalByRefObject in Singleton mode, 17–18
IDE (integrated development environment), trying to start Windows services from, 115
IDictionary, role in server-side sinks, 244
IDynamicProperty interface
    displaying, 246
    registering with current context, 247

IIS (Internet Information Server)
deploying server applications with, 117–123
using authentication within, 126–127, 130
ILease interface
adding sponsors with, 150
location of, 136
IMessage, advisory about changing content in IClientChannelSink, 286
IMessageSink interface
checking parameters with, 387–391
connecting to TransparentProxy, 243
example of, 209–211
role in server-side asynchronous processing, 233–235
includeVersions attribute for formatters and providers in configuration files, using, 104–105
InfinitelyLivingSingleton, calling in lifetime example, 144
InitializeLifetimeService() method, overriding, 51–52, 138–139
InitialLeaseTime property, explanation of, 136
installutil.exe
installing Windows services with, 115
uninstalling Windows services with, 117
interface definitions for .NET Remoting, explanation of, 10–13
invocation, types of, 52
IPrincipal interface, verifying group memberships for users with, 128–130
ISerializable interface, examining, 176
IServerChannelSink interface
displaying, 218
role in compression sinks, 255–257
role in server-side asynchronous processing, 233–235
ISponsor interface, implementing, 149–150
IV (initialization vectors), using with symmetric encryption, 272

J

Java EJB (Enterprise Java Beans) remoting framework, evolution of, 6
Java RMI (Remote Method Invocation) remoting framework, evolution of, 6

K

KeepAlive() method, calling for server-side sponsors, 161–162
key pairs
advisory about, 163
generating for assemblies, 163
KeyGenerator.exe, role in encrypting network traffic, 271

L

lastKeepAlive, comparing for server-side sponsors, 158
lease-based object lifetime service, explanation of, 48–50, 135. *See also* lifetime management
Lease class, location of, 136
lease times
changing defaults for, 137–138
changing on per-class basis, 138–139
LeaseManager
advisory about calling server-side sponsors from, 156, 158
role in lifetime management, 135–136, 137
leaseManagerPollTime attribute, role in <lifetime> tag for configuration files, 99
leases, understanding, 136
LeaseSink, explanation of, 221
leaseTime attribute, role in <lifetime> tag for configuration files, 99
lifetime management, 13. *See also* lease entries
example of, 48–52, 139–144
extending example of, 145–148
introduction to, 135–136
for Singletons, 33
<lifetime> tag, using with configuration files, 99
LifetimeService, getting for objects, 150
LIST command, role in POP3, 321
listen attribute, role in HTTP channels for configuration files, 102
listening clients
broadcasting messages to, 194–195
role in refactored event handling, 193–194
Locals window, locating ObjRef in, 207

M

M units for <lifetime> tag attributes, explanation of, 99, 137
machineName attribute, role in HTTP channels for configuration files, 102
MAIL FROM code, role in SMTP, 319
MarshalByRefObjects
defining data object for (in Singleton mode), 16–17
defining remote interface for, 15–16

executing remote method calls on server side with, 14
exposing in Singleton mode, 15–22
functionality of, 28
implementing server for, 17–19
refactoring event handling with, 191–196
role in basic lifetime example, 139–144
role in multiserver configuration, 68
marshalling objects by value, explanation of, 27
Mercury/31 e-mail server, downloading, 371
Mercury core configuration, changing to, 372
Message objects, serializing through formatters, 211–212
message sinks
examining, 209–211
role in .NET Remoting Framework layers, 103
messages. *See also* client-side messaging, server-side messaging
contents of, 208–209
creating, 203–204
moving through transport channels, 212–214
understanding role of, 207–209
metadata
advisory regarding configuration files, 89–95
extracting with SoapSuds command, 92
metadata-assembly generation, role in .NET Remoting, 12
metadata level, working at, 377–378
MethodCall messages
versus ConstructionCall, 207
partial definition of, 208–209
methods
calling for expired objects, 150–155
calling on remote objects, 52
executing asynchronously, 179
MMC (Microsoft Management Console)
creating groups and assigning users with, 129
creating virtual roots with, 119
mobile objects versus remote objects, 1
mode attribute, using with <wellknown> property, 107
MS units for <lifetime> tag attributes, explanation of, 99–100, 137
MTS/COM+ (Microsoft Transaction Server)/Component Object Model) remoting framework, evolution of, 5
MulticastDelegate, basing server-side events on, 194
multiserver configuration
example of, 69–77
fundamentals of, 68
multiserver/multiclient management, performing with .NET Remoting, 14

N
name attribute, role in HTTP channels for configuration files, 101
name parameter
for SMTPClientChannel, 346
for SMTPServerChannel, 356
nested objects, using with ISerializable, 177
.NET console applications, deploying servers a, 111
.NET Remoting
advantages of, 9–14
binding to custom transfer protocols, 321
ease of implementation of, 9–10
extensible architecture of, 10–11
five elements of, 201–202
functionality of, 7
handling reply messages with, 341–343
handling responses with, 340
interface definitions for, 11–12
managing object encoding/decoding with, 13
mapping POP3Msg objects to, 343–344
performing lifetime management with, 13
performing multiserver/multiclient management with, 14
role generated metadata assembly in, 12
role of base objects in, 12
role of message sinks in, 103
role of shared assembly in, 11–12
role of shared interfaces in, 12
scenarios for, 2
serialization of data in, 13
.NET Remoting connections
handling reply messages with, 341–342
handling responses for, 340
making, 336–345
role of asynchronous delegates in, 340–341
network traffic, encrypting, 269
new operator, using with configuration files, 89

nonwrapped proxy metadata, generating, 84–86

O

+OK messages, using with POP3, 320–321
object encoding/decoding, management of, 13
objects
  registering as Singletons, 33–34
  serializing their state, 27–28
  types of, 14
objectUri attribute, using with <well-known> property, 107
ObjRef object
  explanation of, 14
  functionality of, 204, 206–207
  role in multiserver configuration, 68
one-way asynchronous calls, example of, 63–67
one-way relaxed versioning schema, enabling, 175
[OneWay] attribute, removing from asynchronous one-way calls, 66
[OneWay] events, advisory about, 196–198
ORBs (object request brokers), role in CORBA, 4
organization names, establishing character limitations for, 376–377
Organization object, first version of, 375–376

P

PASS *password* code, role in POP3, 320
playback attacks, explanation of, 131
POP (Post Office Protocol)
  encapsulating for channel source, 325–327
  RFC for, 318
POP3
  checking for new mail with, 331–333
  using DeleteMessage() method with, 330
  using Disconnect() method with, 330–331
  using GetMessage() method with, 328–330
  using MessageCount property with, 328
  using SendCommand() method with, 327–328
port 1234, configuring HTTP channel on, 18, 95, 102, 244
port 1235, opening HTTP channel on, 74–75
port attribute, role in <channels> tag for configuration files, 101
port numbers
  advisory when running two servers on one machine, 75
  specifying for server applications, 100–102
port= attributes, creating in channels of configuration files, 150–151
priority attribute, role in HTTP channels for configuration files, 101
ProcessMessage() function
  role in creating sinks for clients, 352–353
  sample implementation for server-side asynchronous processing, 235
  using with compression sinks, 257–258, 264–265
properties, setting on per-host basis, 299–301
protocol considerations for transport channels, 317–323
provider chains
  creating sinks from, 241
  role in client-side sinks, 239
provider chains, role in client-side sinks, 239
providers
  creating for clients, 350–355
  creating for encryption, 280–285
  role in configuration files, 103–104
proxies
  creating, 202–203
  customizing, 308–315
  role in .NET Remoting architecture, 202
proxy assemblies, role in versioning client-activated objects, 172–173
proxy objects
  default combination of, 308
  role in MarshalByRefObject in Singleton mode, 20
  role in remoting applications, 14
proxyName attribute, role in HTTP channels for configuration files, 101
proxyPort attribute, role in HTTP channels for configuration files, 101
published objects, example of, 36–37

Q

QUIT command
  role in POP3, 321, 330–331
  role in SMTP, 319

R

RCPT TO: <*e-mail address*> command, role in SMTP, 319

RealProxy, extending, 309
- role in .NET Remoting architecture, 203–204

ref attribute
- for formatters and providers in configuration files, 103
- role in <channels> tag for configuration files, 101

reference counting, explanation of, 135

RegisterServer() method, role in .NET Remoting connections, 344

rejectRemoteRequests attribute, role in HTTP channels for configuration files, 102

remote components, hosting with Windows services, 113–114

remote events, troubleshooting, 195–196

remote interface, defining for MarshalByRefObject in Singleton mode, 15–16

remote objects
- activation example, 40–41
- advisory about, 391
- calling methods on, 52
- creating with RemotingHelper class, 185–186
- instantiating, 186–187
- managing with .NET remoting, 11
- versus mobile objects, 1

remote SAOs, creating references to, 187

remote sponsors, explanation of, 156

remoteable types, sharing information about, 78

remoting
- evolution of, 3–7
- explanation of, 1
- five elements of, 201–202
- scenarios for implementation of, 2

remoting applications, preparing for development of, 14–15

remoting behavior, reasons for changing of, 249

remoting configuration files, structure of, 99

remoting events
- explanation of, 185–187
- problems with, 187–191

remoting proxies, customizing, 309–310

Remotingonfiguration.Configure() method, calling, 88–89

RemotingHelper class, creating remote objects with, 185–186

RemotingServices.Marshal(), using with published objects, 36–37

renewOnCallTime
- property, explanation of, 136

reply messages, handling with .NET Remoting Framework, 341

replySink, role in asynchronous IMessageSink processing, 223

request and response message contents, dumping with custom proxies, 312–314

request streams, verifying compression of, 263

responses, handling with .NET Remoting Framework, 340

RETR *message_number* command, role in POP3, 321

RFCs (Requests for Comment), 318
- for APOP command, 320
- for Base64 character encoding, 321
- for SMTP protocol, 318

roles, checking, 128–131

runtime information, passing, 286–299

S

S units for <lifetime> tag attributes, explanation of, 99, 137

SAOs (server-activated objects)
- accessing versions of, 170–172
- calling with different delays for clients, 143–144
- versus CAOs, 206
- explanation of, 28–30
- passing runtime information with, 296–297
- presenting with servers, 80–81
- registering on clients with <wellknown> property, 109–110
- server-side implementation for IIS, 118–119
- using asynchronous calls with, 183

SDLChannelSink, explanation of, 219

serializable objects, generating first version of, 177–178

[Serializable] objects
- using in shared assemblies, 90
- versioning, 174–179

serialization
- benefits to versioning, 178
- implementing, 14–15

serializing objects, 27–28

server-activated objects, versioning, 162–164

server applications, deploying, 111

server assemblies, explanation of, 15

server channels, implementing, 355–361

server-side asynchronous processing, performing, 233–235

server-side HTTP channel's sink stack, diagram of, 245
server-side messaging, examples of, 216–218. *See also* client-side messaging, messages
server-side objects
  designing and developing for IIS, 118–119
  using ObjRefs with, 68
server-side PriorityChangerSink code, 293–296
server-side sink chain with compression sink, diagram of, 250
server-side sink providers, creating, 258–260
server-side sinks. *See also* client-side sinks, sinks
  creating, 244–245, 361–366
  implementing for compression sinks, 255–258
  initializing for encryption, 281
server-side sponsors
  creating, 160
  keeping alive, 156
  pinging with clients, 157–158
  using, 156–162
server-side transport sinks versus conventional channel sinks, 362
ServerContextTerminatorSink, explanation of, 220–221
server.cs file, generating with SoapSuds, 94, 183
ServerObjectTerminatorSink, explanation of, 221
ServerProviders and ClientProviders properties, role in configuration files, 102–106
servers
  advisory when running on same machine, 75
  advisory when using events with, 187
  creating for synchronous calls, 53–57
  enhancing for versioned SAOs, 168–170
  example for SingleCall SAO, 30–33
  example with client-activated object, 39–40
  implementing for MarshalByRefObject in Singleton mode, 17–19, 22–25
  implementing for SoapSuds-generated metadata, 80–81
  implementing in sample multiserver configuration, 73–76
  implementing IServerChannelSink on, 287–288
  role in configuration files, 90–91, 95
  role in sample multiserver configuration, 70
  sample factory design pattern implementation for, 44–46
  sample implementation for lifetime management, 140–142, 145
  using binary encoding via HTTP with, 105–106
ServerStartup class, role in Singleton-object example, 34, 36
<service> property, role in configuration files, 106
shared assemblies
  advantage of, 11–12
  creating abstract base classes in, 78
  implementing for sample multiserver configuration, 72–73
  management of, 77–86
shared base classes, functionality of, 78
shared implementations, functionality of, 78
shared interfaces
  for factory design pattern, 44
  functionality of, 78
  role in .NET Remoting, 12
sign-on process, securing, 131–133
single sign-on, enabling, 132–133
SingleCall SAOs
  example of, 30–32
  explanation of, 28
  role in sample multiserver configuration, 74–75
Singleton mode, defining MarshalByReftObject in, 15–22
Singleton SAOs
  example of, 33–36
  explanation of, 28
  tip about, 35
sink chains
  defining in configuration files, 237–238
  role in implementing client channels, 349–350
sink providers
  behavior after invoking CreateSink() method, 242–243
  creating, 258–260
  examining <url> entries with, 305
  understanding, 237–238
sinks. *See also* client-side sinks, server-side sinks
  building for encryption helpers, 272
  creating for clients, 350–355
  creating for encryption, 274–280
  using, 261–263

SinkStack, contents in asynchronous requests, 228–230
SMTP conversion, example of, 318–319
SMTP sessions, starting, 319
SMTP (Simple Mail Transfer Protocol)
  encapsulating for channel source, 323–331
  RFC for, 318
SMTPChannel
  code for, 366–369
  using, 369–371
SMTPClientChannel, parameters for, 346
SMTPClientTransportSinkProvider code, 350–351
SMTPHelperClass, connecting to .NET Remoting with, 336–339, 344
SMTPHelper.MessageReceived() static method, calling, 332–333
smtpServer parameter for SMTPClientChannel, 346, 356
SMTPServerChannel
  parameters for, 356
  versus SMTPClientChannel, 356–357
SMTPServerTransportSink, implementing, 362–364
sn.exe, generating key pairs with, 163
SOAP formatter, role in handling asynchronous responses, 233
SOAP (Simple Object Access Protocol) remoting framework, evolution of, 6–7
SoapClientFormatterSink, explanation of, 215
SoapFormatter, introduction to, 211
SoapServerFormatterSink, explanation of, 219
SoapSuds
  alternatives for starting of, 83
  calling, 79
  extracting metadata with, 167
  generating nonwrapped proxy source with, 172–173
  generating source code with, 93–95
SoapSuds -gc output for SAO, example of, 181–183
SoapSuds-generated metadata, examples of, 79–86
SoapSuds-generated proxies, using delegates with, 180–181
SoapSuds, role in .NET Remoting generated metadata assembly, 12
SoapSuds.exe, role in SAO example, 38–40
SoapType attribute, removing from server.cs, 183
source code
  generating with SoapSuds, 93–95
  simplifying for clients with wrapped proxies, 81–82
sponsors
  role in lifetime management, 49, 135–136
  working with, 149
sponsorshipTimeout attribute
  explanation of, 136
  role in <lifetime> tag for configuration files, 99
SSL (Secure Socket Layers) encryption, advisory about, 134
StackBuilderSink, explanation of, 221
StartListening() method, calling for server channels, 357–358, 360
STAT command, role in POP3, 321
StopKeepAlive() method, calling for server-side sponsors, 159
strictBinding attribute for formatters and providers in configuration files, using, 104–105
strong naming
  displaying for assemblies, 166
  role in versioning, 162–164
suppressChannelData attribute, role in HTTP channels for configuration files, 101
symmetric encryption, example of, 269–274
synchronous calls
  example of, 53–57
  explanation of, 52
  using RealProxy with, 203
synchronous messages, examples of, 214–216
SyncProcessMessage() method, implementing, 286–287

T
TCP channel, role in configuration files, 102
TCP trace from client/server connection, example without using sinks, 262
time units for <lifetime> tag attributes, 99
TimeSpans, returning for sponsors, 149–150
transport channels
  moving messages through, 212–214
  preparing machines for, 371
  protocol considerations for, 317–323
  role in messages, 207–208
transport sinks, creating, 361–366

try/catch blocks, adding to clients for asynchronous one-way calls, 65–66
TTL (time-to-live)
  changing defaults for, 137–138
  role in lease-based object lifetime service, 135
type attribute
  for formatters and providers in configuration files, 103
  role in <channels> tag for configuration files, 101
  using with <activated> property for configuration files, 108, 110
  using with <wellknown> property, 107
  using with <wellknown> property for configuration files, 109–110
TypeLoadExceptions, generating with delegates, 180
<Typename> properties in configuration files for lifetime example, 147
typos, effect on configuration files, 97

U
/u [<assembly>] parameter, role in versioning, 164
uncompressed streams, creating class for, 253–254
Unicode, explanation of, 321
url attribute
  using with <client> property for configuration files, 109
  using with <wellknown> property for configuration files, 109–110
UrlAuthenticationSink, using, 306–308
UrlAuthenticator code, storing usernames and passwords with, 301–305
URLs (uniform resource locators)
  parsing, 344
  specifying username/password combinations for, 307
useDefaultCredentials property, enabling single sign-on with, 132–133
useIpAddress attribute, role in HTTP channels for configuration files, 102
user accounts, defining for transport-channel examples, 371–372
USER *username* code, role in POP3, 320

V
versioned SAOs, lifecycles of, 165–172
versioning
  ins and outs of, 162
  purpose of, 175–176
  role in configuration files, 104–105
virtual roots, creating for IIS as container for server-side objects, 119

W
Web services remoting framework, evolution of, 6–7
Web sites
  for compression library, 251
  for Mercury/32 e-mail server, 371
  for SOAP binding to SMTP, 321
  tcpTrace tool, 262
web.config, role in lifecycle of versioned SAOs, 166
<wellknown> property
  role in configuration files, 106–107, 109–110
  role in lifecycle of versioned SAOs, 166–167, 170
Windows authentication, enabling, 131–132
Windows services, deploying servers with, 111–117
wrapped proxies
  creating with SoapSuds, 79–84
  internals of, 83–84
wrapper methods, using delegates with, 184–185
wrappers, event handling with, 192
?WSDL parameter, advisory when creating server-side sinks, 245

X
XML (eXtensible Markup Language), serializing objects to, 27–28
XML-RPC (eXtensible Markup Language-Remote Procedure Calls) remoting framework, evolution of, 6