Network Query Language (NQL)

David Pallmann

Wiley Computer Publishing

John Wiley & Sons, Inc.

Network Query Language (NQL)

David Pallmann

Wiley Computer Publishing

John Wiley & Sons, Inc.

To my dear wife, Rebekah

Publisher: Robert Ipsen
Editor: Ben Ryan
Development Editor: Kathryn A. Malm
Managing Editor: Angela Smith
New Media Editor: Brian Snapp
Text Design & Composition: D&G Limited, LLC

This book is printed on acid-free paper. ∞

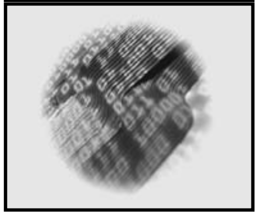
Library of Congress Cataloging-in-Publication Data:

Pallmann, David.
Network query language (NQL) / David Pallmann.
p. cm.
ISBN: 0-471-20766-7 (acid-free paper)
1. Internet programming. 2. Network Query Language (Computer program language) I. Title.
QA76.625 .P35 2002
005.2'762--dc21

2001006061

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1



Contents

Foreword

Information is a company's greatest asset. Within the maze of corporate networks and back office systems that asset is typically locked in place, a prisoner of the system that produced it, network that carried it, the client that displayed it. Reusing information presented by an existing system in a new application is tantalizingly easy to talk about but difficult in practice to do. Certainly newer open system architectures make this easier, but there is always some legacy system which is either not open or inaccessible through programmable interfaces. These are the villains of the system integrator's world.

The situation that is so frustrating is that you can see the data that you want to use on the screen—it's right there in front of you, perhaps in three different systems—and you could describe to a six-year-old what steps to take to come up with the desired result. Why can't computers connected to networks do the same?

I first met David Pallmann in 1998 when he came to pitch his ideas about standard ways to access data on network-connected computers. David was spellbinding with his rapid-fire presentation on how the Network Query Language (NQL) could harvest in situ data and repurpose it in new applications. Because combining information from multiple separate systems together to produce larger results has been my passion and profession for many years, David and I hit it off.

David's idea with NQL was to produce a common approach for accessing networked data, much like SQL made access to relational databases uniform. Initially David and his company NQLi focused on building *agents* to access data from World Wide Web pages using NQL. They built a variety of systems for their clients using NQL, including Stock-Vue, a product that uses NQL to harvest and reassemble information about investments.

As they built these applications, David and his team added new capabilities to the language—features that were motivated by the needs of real-world application developers.

When I first encountered NQL, it had primitives supporting Web access, as well as normal control structures, pattern scanning, and output commands for taking harvested information and presenting it in HTML pages. Over the next two years, NQL blossomed into a very complete programming system, adding support for such essential topics as XML, filling out support for networked data facilities such as news, mail, ftp, telnet, and Web site crawling, adding access to non-networked data in local databases and file systems, and finishing off the system with lots of niche, but important, capabilities such as support for genetic algorithms, fuzzy logic, and parallel processing.

Ho, hum, you say. YAPS: Yet Another Programming System. We've all seen similar capabilities in a variety of different systems—perhaps not as well integrated together, but they exist. Wait, the best is yet to come . . . .The NQL Programming Environment.

I don't know about you, but I never get things right the first time I try something. I am constantly trying things out, refining them, trying other approaches. I hate the delay and brittleness of the edit, compile, link, debug cycle. I much prefer the edit, run cycle that an interpreted system allows. Furthermore, although a programmer at heart, I'd much rather tell a computer to watch carefully what I am doing and then make a procedure based on those actions. (For the hard core: I love EMACS keyboard macros).

One of the strongest features of NQL is that it is interpreted. As such, it earns all of the honors that go to such systems. As a developer, you interact with NQL through the NQL Client, a full featured programming environment for creating, modifying, and debugging NQL programs. You can use the editor to write your programs, but also there are a number of wizards which will build NQL program fragments for you.

One such wizard, the browser recorder, allows you to visit a Web site, drill down to the page that contains the information you are interested in and then identify the specific area of that page that contains the information, putting that information in an NQL variable for subsequent processing. With this program-by-example wizard, you identify the place to visit for subsequent information harvesting of that moment's version of the information—the lead story in the living/arts section of a newspaper, the current price of soybeans, or the front page picture on your favorite sports Web site. Further programming in NQL would allow you to build and export an HTML page that combined these items into one consolidated presentation—or deliver an XML representation of these items, plus XML representations of other information retrieved from local SQL databases to another application.

One last observation: By staying at a high level in the types of objects and operations on those objects, NQL allows you to stand on the shoulders of other successful system developers, take information components from those systems, and deliver your audience the value of selective combinations, decisions, and derivations based on those components.

Let me stop now and let you read the book.

Harry Forsdick
VP Advanced Technology
Genuity, Inc.

Introduction

This book describes a new programming language, Network Query Language. Network Query Language is a kind of shorthand for Internet and network programming that lets you easily combine and leverage technologies such as XML and the Web. If you like Stuctured Query Language (SQL) for database programming, you'll love NQL for network programming. If you have a need to create commercial-grade connected applications such as bots, intelligent agents, middleware, and Web applications, you'll find NQL well worth looking at. For investing a small amount of time in getting to know the language, you'll receive a startling savings in development time.

Network Query Language runs on all major computing platforms, and you'll find a copy of it on the CD at the back of this book. You can also download the language from www.nqli.com.

Overview of the Book and Technology

Welcome to a language that breaks some of the rules and isn't afraid to venture into new territory. Where else can you crawl a Web site and download files with a single statement? Or create, train, and run a neural network with a five-line script? Network Query Language equips you with extensive capabilities in four functional areas that are usually lacking in other programming languages: Communications, conversion, distributed processing, and artificial intelligence.

Network Query Language is intended for both experienced developers and beginners. If you're a seasoned developer, you'll find NQL saving you oodles of time. Tasks that used to take you a month can be accomplished in a week, and what used to take a week can be accomplished in a day. For the junior programmer, you'll find NQL very easy to pick up. The language avoids cryptic punctuation, does not assume a computer science background, and uses simple, easy-to-remember keywords. The three hallmarks of NQL are unparalleled ease of use, lightning-fast development, and sheer power. It's not uncommon for developers to tell us they've created solutions and put them into production within 24 hours of their first exposure to NQL.

Network Query Language has interesting roots. My colleagues and I didn't set out to create a new programming language, but we ended up creating one out of necessity. After five years of developing custom solutions—such as intelligent agents, corporate portals, content servers, shopping bots, search engines, and middleware—we had used many different languages and found all of them unsatisfying. A pattern was emerging: No matter what language we used for a project, the building blocks we needed to use on a daily basis were missing. It offended us that programming languages don't contain built-in statements for tasks such as interacting with Web sites, sending and receiving e-mail, working with XML, and applying artificial intelligence techniques. We decided to do something about it. Network Query Language was born when we developed over 500 powerful building blocks and wrapped an SQL-like language around them.

If NQL sounds similar to SQL, that's no accident: SQL is our hero, one of the great success stories of the computer industry. It turned long, proprietary database programs into short, generic queries. There were numerous benefits. Database programs could be written in significantly less time. Short queries were less likely to contain bugs than large database programs, and they could be developed far more rapidly. A more junior level person could perform database programming. Structured Query Language was such an elegant idea that although 25 years have gone by, it is still with us and is used pervasively. We believe today's Internet and network programming situation is remarkably similar to the database problems of the past and cries out for an SQL-like remedy. Our intention with NQL is to appeal to the same paradigm. The things that are true of SQL are also true of NQL: Large programs become short, simple scripts. Extremely rapid development becomes the norm. Junior people can do powerful things.

Best of all, you don't have to choose between NQL and your favorite programming language. Although NQL can be used as a stand-alone solution, it can also be called as a component from another language. If you prefer, you can continue using languages such as Visual Basic, C++, or Java and call NQL when needed to be that missing puzzle piece in a project. Be forewarned, however, that NQL is addictive and you may find yourself using it more and more.

This book's aim is to serve as a handbook: Your field guide to NQL, if you will. It introduces the language, takes you on a guided tour, teaches through the use of examples, and provides reference information.

How This Book Is Organized

The organization of this book is such that it can be read straight through or consulted topically. Readers who are new to the language will find introductory material at the beginning and a smooth progression from chapter to chapter. Each chapter stands on its own, however; readers already familiar with NQL can go directly to an individual chapter for assistance in a particular subject.

Chapter 1, *Presenting NQL*, gives you a first look at Network Query Language without getting into too much of the nuts and bolts. The chapter covers the appropriate uses for NQL, its design philosophies, and the unique features that distinguish it from other languages.

Chapter 2, *Developing in NQL*, guides you through the use of the NQL development environment. The chapter describes how to enter and edit code, run scripts, use the debugger, and deploy solutions.

Chapter 3, *Language Fundamentals*, covers the syntactical rules of NQL, including statements, functions, comments, code blocks, literals, and variables. The chapter also describes the stack and the success/failure condition, two constructs which allow statements to work together efficiently.

Chapter 4, *Flow of Control*, describes how program flow is governed in NQL. The chapter discusses loops, functions, parallel processing, and timing.

Chapter 5, *Expressions, Operators, and Functions*, explains the rules for combining constants, variables, operators, and functions together in expressions. It also documents the built-in functions of NQL and how user-defined functions are specified.

Chapter 6, *Working with Files*, visits the subject of file input/output in NQL. The ability to create, open, append, read, and write files is discussed. File and directory operations are also covered.

Chapter 7, *Transferring Files*, describes how files can be transferred to and from remote systems using File Transfer Protocol (FTP). It also covers file and directory operations.

Chapter 8, *Working with Databases*, discusses how to query and modify databases. Several database protocols are supported, permitting access to the vast majority of databases.

Chapter 9, *Working with XML*, explores the retrieval, parsing, and generation of XML documents. One of NQL's most attractive features is its ability to convert data to and from XML.

Chapter 10, *Accessing Web Sites*, focuses on interfacing with the World Wide Web. Material covered includes accessing Web pages, submitting forms, crawling Web sites, and downloading files. Web services are also covered.

Chapter 11, *Processing HTML*, attacks the problem of how to programmatically mine data from Web pages. Multiple methods of extracting information from HTML documents are covered.

Chapter 12, *Sending and Receiving E-mail*, examines NQL's messaging capabilities. The mechanics of sending e-mail messages, receiving them, and acting upon them are covered.

Chapter 13, *Accessing Newsgroups*, shows how to access newsgroups, also known as forums or discussion boards. Reading newsgroup postings allows your programs to gather information on public perceptions of a subject or who the interested parties are.

Chapter 14, *Searching Directory Servers*, covers how to access directory servers with the LDAP protocol. Searching directory servers allows your programs to connect better with their local environment and locate people, systems, and resources.

Chapter 15, *Terminal Emulation*, enters the world of connecting to UNIX systems, minicomputers, and legacy systems. Your programs can mimic terminal-based interactive users, sending commands, waiting for responses, and capturing screens of output.

Chapter 16, *Interacting with Applications*, is about cooperation between your programs and other software. OLE Automation permits you to exchange information with other programs or control them. Exporting allows your scripts to create documents, spreadsheets, and presentations.

Chapter 17, *Socket Communications*, describes TCP/IP socket programming in NQL, which you would use to implement a protocol NQL does not already support, or your own custom protocol.

Chapter 18, *Serial Communications*, discusses how to perform serial port communications in NQL, which you would use to read and write data to and from a local device connected to your computer.

Chapter 19, *Synchronization*, is concerned with creating distributed applications, where multiple agents must cooperate to work as a team. Methods of communications and synchronization are covered.

Chapter 20, *E-Commerce*, explores electronic transactions, which NQL supports for several major credit card providers. Open Financial Exchange is also covered, which makes automated interaction with financial institutions possible.

Chapter 21, *Graphics*, explains how to work with graphic images in NQL. Images can be dynamically created, modified, saved, and converted to other formats. The chapter also covers reading images from digital devices.

Chapter 22, *Agent Characters*, discusses how to create animated agent characters that speak, move, and perform actions.

Chapter 23, *Fuzzy Logic*, describes the use of fuzzy logic, a form of soft computing. Fuzzy logic allows your programs to represent real-world quantities more accurately and make better decisions.

Chapter 24, *Neural Networks*, covers how to create, train, and run neural networks. Neural networks are powerful due to their ability to learn, and are useful for applications that require pattern matching or prediction.

Chapter 25, *Bayesian Inference*, focuses on Bayes' Theorem and how it can be applied to compute probabilities. Bayesian inference allows causes to be inferred from effects, rather than the usual inferring of effects from causes.

Chapter 26, *Interacting with the Desktop*, enumerates the options NQL offers for interacting with the desktop. This includes opening documents, displaying console windows, rendering text-to-speech, and playing multimedia files.

Chapter 27, *Network Monitoring*, focuses on the use of the SNMP protocol to interact with network devices. Network devices can be discovered, monitored, and controlled.

Chapter 28, *Web Applications*, explores the use of NQL to drive solutions that can be accessed from Web browsers. Like ASP and Perl, NQL scripts can power Web applications.

Chapter 29, *Supporting Mobile Devices*, explains how to write Web applications that can be accessed by mobile devices. A variety of PDAs and phones can be supported.

Chapter 30, *System Actions and Information*, covers actions the operating system can perform, such as powering down the computer; and information the operating system can supply, such as the current user's privileges.

Chapter 31, *Calling NQL as a Component*, is about the use of NQL as a callable object from other development environments. It is possible to gain the benefits of NQL without giving up the use of your favorite programming language.

Chapter 32, *NQL Cookbook*, is just what it sounds like: a collection of recipes for putting NQL to work for a particular task. For each task listed in this chapter, a step-by-step procedure and sample script are included.

Appendix A, *NQL Versions and Editions*, is your ammunition against version and platform confusion. It describes editions of the language and platform differences. Past and present versions of NQL are also described.

Who Should Read This Book

If you need to develop connected applications quickly, this book (and Network Query Language) are for you. Since NQL appeals to both junior and experienced developers, the book doesn't assume that you have a deep background in computer science. It is assumed that you understand the rudiments of programming, however. Programmers, IT personnel, Web masters, and content engineers can all put NQL to good use in short order.

If you really want a firm grounding in NQL, then by all means read the book straight through and try each of the tutorials. If you're short on time or have a looming deadline, here are some recommendations:

If you're a junior programmer, read Chapters 1 through 5 carefully, which will give you a good grounding in essentials. Go through the tutorials in these chapters and make sure you understand them. The remaining chapters of the book are topical, so when you need to do something new, like sending mail messages, go to the appropriate chapter. You might want to briefly scan through all of the chapters to develop a feeling for what NQL is capable of. You will find many common tasks addressed in Chapter 32, including scripts that can serve as templates for your own solutions.

For experienced programmers, you'll probably want to skim through Chapters 1 through 5 to get a feel for the language, taking note of NQL's distinguishing characteristics and pausing for an extended read when you come across something interesting. The remaining topical chapters can be read in any order desired, and contain tutorials for getting up to speed quickly. If you plan to use NQL to drive Web applications or mobile applications, you'll want to read Chapters 28 and 29 carefully. If you plan on calling NQL as a component from another language, see Chapter 31. The cookbook in Chapter 32 provides many ready-to-run scripts for putting NQL to use right away.

Conventions

This book uses certain conventions for showing code examples and reference forms of statements and functions.

Names of Statements, Functions, and Parameters

In the body text of the book, names of statements, functions, and parameters are often italicized. For example, "the *replace* function requires two parameters, a *search-string* and a *replace-string*."

Code Examples

Code examples appear in monospaced type.

```
open filename
while read(line)
{
    show line
}
close
```

When part of a code example is left to the imagination, an italicized line beginning and ending with an ellipsis (...) appears.

```
open filename
while read(sLine)
{
    ...do something with sLine...
}
close
```

Small code examples are in-line with the body text; code examples that are larger or are especially significant are cited by a listing number, such as "Listing 8.1."

```
//movie-reviews - reads movie reviews out of sample Access database
and displays results

dbpath = "c:\\Program
Files\\NQL\\Samples\\Scripts\\Database\\movies.mdb;"

dbprotocol "ado"
if opendb("Driver=Microsoft Access Driver (*.mdb); DBQ=" dbpath)
{
    boolean bHaveData = select("select * from movies")
    while bHaveData
    {
        dumpvars
        bHaveData = nextrecord()
    }
}
else
{
    show "Unable to open the database " dbpath
}
closedb
```

Listing 8.1 Database example.

Statement and Function Reference Forms

When a statement or function is first introduced, its reference form is usually given. Reference forms appear in monospaced type. Although NQL will accept upper-, lower-, or mixed-case keywords, the official rendering of these keywords in the language is lower case, and that is how they appear in this book.

```
end
```

Parameters are italicized, indicating the places where you need to supply values. Statement reference forms indicate a keyword and some number of parameters. Function reference forms indicate a return type, a keyword, and some number of parameters in parentheses.

```
open filespec
boolean open(filespec)
```

Parameters that must be variables are indicated by the word *var*. When a variable list (one or more variables) is needed, the term *var-list* is used.

```
clear var
output var-list
```

Square brackets [ ] indicate optional elements. An ellipsis (...) within square brackets indicates elements that can repeat multiple times.

```
nextline [var]
for var = [initial, ] final [, increment]
sendmessage recipient, subject, message1, type1 [ ... , messageN, typeN]
```

If there are many statements or functions to be listed, they are arranged in a table.

Planned Features

Some NQL language features described in this book are described as *planned features*. This means that these features were on the drawing board for Network Query Language and were considered likely to be included in the 2.0 release, but a final decision had not been committed to as of the time of this writing. In tables of statements and functions, an asterisk signifies a planned feature. In body text, the phrase *Planned feature:* precedes discussion of planned features.

Names of Web Sites

Many of the sample scripts in this book access Web sites. Rather than using actual Web site names, this book often uses generic placeholder names such as www.my-site.com, www-my-news-site.com, and www.my-retail-site.com.

Tools You Will Need

To get the most out of this book, you'll need a copy of Network Query Language so that you can try things out. Even if you don't own a copy, you can obtain an evaluation edition, either from the book's companion CD-ROM or the Internet. The version of NQL reflected in this book is Version 2.0, so you'll do best to have NQL 2.0 or later installed. If you can only get NQL 1.1 for your platform, this book will still be useful to you. Appendix A will help you keep your sanity if you happen to be using a different version of NQL.

What's on the CD-ROM

The companion CD-ROM contains software and code examples.

The software on the CD-ROM includes three versions of Network Query Language:

- Network Query Language for Windows version 2.0 (60-day evaluation)
- Network Query Language for Windows version 1.1 (60-day evaluation)
- Network Query Language for Java version 1.1 (60-day evaluation)

As of this writing, the Windows edition of NQL 2.0 was just nearing readiness. If you're running on a non-Windows platform, visit www.nqli.com or www.networkquerylanguage.com to determine what the latest version of NQL is, and what platforms it is available for. If you can't get your hands on NQL 2.0 or later for your type of computer, take heart. NQL 1.1 is available in both a Windows edition and a Java edition that runs on all major computing platforms. If you have to use NQL 1.1 instead of NQL 2.0, see Appendix A for some helpful information.

The CD-ROM also contains all of the NQL script code cited in the book as tutorials. When you go through each chapter's tutorial, you have the choice of entering script code by hand or copying the script from the CD-ROM. If a tutorial script requires supporting files, these are included as well.

Lastly, the scripts from Chapter 32, the NQL cookbook, are also included on the CD-ROM. These ready-made scripts can reduce your learning curve significantly.

Where To Go from Here

It's my sincere desire to see you and the rest of the world's software developers do more powerful things in a shorter timeframe. Network Query Language can be instrumental in making that happen. If it becomes part of your standard collection of programming tools you will find yourself being a programming superstar more often, and with less effort! In this book, I've sought to justify the reasons for NQL's existence; entice you with its power, brevity, and ease of use; educate you through tutorials; and equip you with essential reference information. The rest is up to you and your imagination.

I am happy to hear from readers, and can be reached at nqlbook@hotmail.com.

David Pallmann
Mission Viejo, California

CHAPTER

Acknowledgments

Although I am the inventor of Network Query Language, I can hardly take sole credit for it! Many talented people have contributed to the NQL effort over the years. Christopher Hunter and John Ross have been integral to the language all along, from its earliest inception to present day. In version 2.0, Brandon Bethke and Jed Stumpf also served major architectural roles, taking charge of the compiler and runtime systems. The language also benefited from the ongoing engineering efforts of Robert Apodaca, Scott Bishop, Mike Campbell, Brian Connolly, Matt Grofsky, Ashur Novick, Peter Tran, and Bill Walker. Gentlemen, I salute you all.

It takes more than engineers to see a new product developed. I would like to thank Denny Michael and Doug Tullio, both of whom believed in NQL enough to dedicate years to seeing it advance from an interesting concept into a commercial-grade product. Their tireless efforts provided the conditions under which NQL could be developed.

To my wife Becky and daughters Susan and Debra, I must express my heartfelt thanks for your support, without which it would have been impossible to write a book. Putting up with a husband and dad with his nose in a laptop every waking moment can't have been enjoyable. The good news is, I'm back!

I am indebted to the fine folks at John Wiley & Sons for making the writing and editing process easy for me, and for their dedication to professionalism.

CHAPTER

1

Presenting Network Query Language (NQL)

This chapter gives you a first look at Network Query Language (NQL). It is often instructive to gaze at something new from 30,000 feet before zooming in on the minute detail, and that is certainly true of computer languages. Before diving into the nuts and bolts of NQL, we want to consider the big picture: Its proper uses, its design philosophies, and its unique features. This chapter offers a first look at NQL, discusses appropriate uses for the language, and examines its unique qualities. The chapter ends with a tutorial in which you will enter, save, and run your first NQL program.

A First Look

It is traditional when introducing new languages to show a "Hello, World" program. So let's get that out of the way. The one-line script that follows displays "Hello, World" on the screen.

```
show "Hello, World!"
```

That's hardly very interesting, so let's look at something slightly more ambitious (and useful). The following script is a complete NQL program.

```
get "http://www.my-news-site.com"
match '<li><a href="{link}">{headline}</a>'
```

```
while
{
     output headline, link
     nextmatch
}
```

Even if you've never seen NQL before, you can probably make an educated guess about the purpose of this program. It accesses a news Web site, locates the news stories, and outputs the headlines and links as eXtensible Markup Language (XML). A few things should strike you about this program:

- It is very small.
- It is very clear.
- It does something that is not easy (or even possible) in many mainstream programming languages.

Appropriate Uses for NQL

Network Query Language is ideally suited for creating connected applications such as bots, agents, spiders, middleware, Web applications, and mobile applications.

Bots are automated programs (the term bot comes from the word robot). Bots are essential because of infoglut. The amount of information on the Internet is constantly growing, and the same is true of electronic information within corporate networks. Yet a human being's ability to read, digest, and work with information is essentially fixed. Thus, there is an *information gap* that grows worse each every day. Bots come to the rescue by automating tasks for people and making life manageable again.

Agents are bots that serve an individual. In the real world, we employ human agents for all kinds of purposes: Travel agents, insurance agents, real estate agents, accountants, attorneys, and so on. Agents specialize in an area and represent our personal interests. The same is true of software agents: They are very good at a specific task, and they have a clear understanding of their master's wishes.

Spiders are bots that crawl the Internet, indexing Web pages and constructing the databases that power search engines. Spiders can be used for other purposes beside search engines, such as creating site maps.

Middleware applications connect two or more systems or applications, helping them to work in harmony as part of a larger system. Middleware is valuable because it avoids having to change major systems, instead taking the approach of interfacing between them.

Web applications are the modern gateway to interfacing with companies and other organizations. Web applications perform such diverse functions as conveying information, searching databases, initiating actions, and effecting transactions.

Mobile applications are Web applications that support Personal Digital Assistants (PDAs) and cellular phones rather than desktop Web browsers. Mobile applications have to be concerned with supporting a variety of devices and accommodating a small display size, different output formats, and a simple user interface.

Why a New Language?

Network Query Language came about after repeated disappointment with mainstream programming languages. It wasn't that these languages were especially deficient; rather, what has changed is the focus of modern software development. Since the rise of networking and the Internet, there has been a shift toward new kinds of applications that stress communication, content transformation, distributed processing, and smarter software. That means the statements and functions that have been standard in languages for decades no longer serve us so well. In an age where programmers are performing new tasks and meeting new challenges, we need some new tools in our toolbox.

Design Philosophies

It is no accident that NQL scripts are powerful yet remain small and elegant—that is a reflection of the design philosophies that went into the language. The benefits to you are more power, clarity of code, speedier development, and the ability to extend the language further.

Power

NQL is an extremely powerful language. It has far more built-in capabilities than other programming languages. There are over 500 statements and functions in NQL, many of which aren't found elsewhere. This unrivaled collection of building blocks is at your immediate disposal. Many of the one-liners in NQL accomplish things that would normally require large, exceedingly complex programs in their own right.

Many difficult tasks become simple in NQL. It might be beyond you to directly write programs that interact with Web sites, send and receive email, export data to applications, or perform artificial intelligence—but these kinds of activities are ridiculously easy in NQL.

Clarity

NQL is intended not only for professional programmers, but also for anyone who needs to quickly create a connected application, such as IT personnel, content engineers, and Webmasters. With this in mind, NQL makes no assumptions about computer science background or prior programming experience. The result is a language that has a simpler syntax than Visual Basic yet retains its power.

The names of statements and functions in NQL have been selected to be as straightforward as possible. Whereas some languages use ornate names for historical reasons, NQL's keywords tend to be one or more real words found in the dictionary. It is easy to discern the meanings of statements with intuitive names like *openmail*, *sendmessage*, and *firstunreadmessage*.

Likewise, the use of cryptic punctuation has been strenuously avoided. If an NQL statement requires parameters, they simply appear to the right of the statement. If there is more than one parameter, they are separated by commas. A newline separates statements, so there's no need for an end-of-statement character such as the semicolon used in C++ and Java.

Rapid Development

NQL promotes rapid development. The aim is to turn months into weeks, weeks into days, and days into hours. Features that facilitate rapid development include the following:

- A fast, friendly development environment
- A scripting language design that does not require a separate compilation process
- Powerful building blocks built directly into the language
- A clear, compact syntax
- A safe, forgiving runtime environment

When you craft a program in a standard programming language, you typically combine hundreds, thousands, or tens of thousands of instructions. Naturally this takes time to write and debug. When programming in NQL, programs are composed of far fewer statements because of the powerful building blocks built into the language. Sometimes a complete NQL script is just a handful of statements. Tiny programs mean faster development, a smaller likelihood of errors, and faster debugging.

Let's consider a case in point. Say you want to crawl a Web site and download files, a task that is easy to talk about conceptually but is not trivial to program. Site crawling requires a careful understanding of Web technologies in order to be done well and normally an intermediate to senior level developer would be needed to implement such a program. In a language like C++ or Java, crawling a Web site requires 20 or so pages of code. In NQL, the task can be accomplished in just one statement, because site crawling and file downloading are concepts built into the language.

Familiarity

NQL also stresses familiarity. Some languages make the error of being unique to the point of alienating developers. Although NQL has many unique features and a unique grammar, it also provides plenty of things programmers will find familiar. Among these are the following:

- Typed variables
- Operators and expressions
- Multi-statement code blocks

```
// Retrieve news headlines from a news site main page,
// output to a text file (news.text), and
// open the results on the desktop.

String headline
String link
Integer stories

Create "news.txt"

Get "http://www.my-news-site.com"

stories = 0

Match '&#149;&nbsp;<A href="{link}">{headline}</A>'
While
{
    Write headline & ","
    Write link & "\r\n"
    stories = stories + 1
    NextMatch
}
Close

OpenDoc "news.txt"
```

Listing 1.1 Visual Basic-like phrasing.

- Standard flow of control statements
- User-defined functions

To illustrate the point, take a look at the script code in Listings 1.1, 1.2, and 1.3. All three scripts perform identically, but they don't look all that similar. If you're from a Visual Basic background, Listing 1.1 probably looks most familiar to you.

If you come from a C++ or Java background, you're probably more at-home with Listing 1.2. NQL supports a number of conventions from Visual Basic, C++, and Java, possessing aliases in many cases. Most statements can also be treated as functions.

If you've used assembly language or a stack-oriented language such as Forth or PostScript, you may relate best to Listing 1.3. The use of NQL statements and the stack allows one to program in terms of instructions that work together, just like the instructions in a microprocessor.

As you can see, there is a great deal of freedom in how programs are expressed in NQL. Capitalization is immaterial, as is indenting and the use of white space (that is, spaces, tabs, and blank lines).

```
/***********************************************
* Retrieve news headlines from a news site page, *
* output to a text file (news.text), and         *
* open the results on the desktop.               *
***********************************************/

string headline, link
int stories = 0

if create("news.txt")
{
    get "http://www.my-news-site.com"
    while match('&#149;&nbsp;<A href="{link}">{headline}</A>')
    {
        write headline ","
        write link "\r\n"
        stories=stories+1
    }
    close
    opendoc "news.txt"
}
```

Listing 1.2 C++/Java-like phrasing.

Customizable and Extensible

NQL is a customizable language. Every keyword in the language can be redefined. The same is true for every error message. This level of customization makes it easy to handle internationalization. All it takes to change keywords or messages is editing and compiling a message file.

As powerful as NQL is, it can be extended. Anyone can write new statements for the language, which become seamless additions indistinguishable from the core statements in the language. In the Windows edition of NQL, a Dynamic Link Library (DLL) (created in any language) can implement new statements. In the Java edition of NQL, a Java class can implement new statements. Third parties with experience in specific areas of technology or industries can leverage their expertise and make contributions to NQL.

The Right Building Blocks

What are the building blocks that make NQL special? They fall into four main categories: *communication*, *data conversion*, *automation/distributed processing*, and *intelligent behavior*. Connected applications frequently need to draw on building blocks in these categories. For example, consider an agent that retrieves news stories from a Web site every 24 hours and stores them as XML in a local database. The agent utilizes commu-

```
; Retrieve news headlines from a news site page,
; output to a text file (news.text), and
; open the results on the desktop.

        STRING  HEADLINE
        STRING  LINK
        INT     STORIES

        CREATE  "news.txt"
        GET     "http://www.my-news-site.com"

        STORIES =0

        MATCH   '&#149;&nbsp;<A href="{LINK}">{HEADLINE}</A>'

        WHILE
        {
                WRITE   HEADLINE | ","
                WRITE   LINK | "\r\n"
                STORIES =STORIES+1
                NEXTMATCH
        }
        CLOSE

        OPENDOC "news.txt"

        END
```

Listing 1.3 Assembler-like phrasing.

nication to retrieve Web pages, conversion to generate XML, automation to run every 24 hours, and intelligent behavior to identify news articles on Web pages.

Communication

NQL is full of support for Internet and network protocols. The communication facilities allow you to retrieve information from just about anywhere, whether on the Internet or corporate network. They also allow you to deliver information, results, or notifications to people and systems, whether nearby or remote. The communication facilities include the following:

Web sites. The ability to read Web pages, crawl Web sites, and interact with Web sites.

File input/output. Reading and writing disk files.

File Transfer. Sending and receiving files between computers.

Databases. Querying databases, and adding, updating, and deleting database records.

Email. Sending and receiving email.

Newsgroups. Scanning newsgroups.

Directory Servers. Querying directory servers.

Terminal Emulation. Accessing UNIX systems and legacy systems.

Interaction with Applications. Controlling applications under program control, and exporting.

Sockets. TCP/IP input/output.

Serial Communications. Serial port input/output.

e-Commerce. Processing credit card transactions and accessing financial servers.

Network Monitoring. Detecting network devices, reading their status, and sending them commands.

Mobile Devices. Recognizing mobile devices and outputting data in their content languages.

Web Services. Accessing XML-based services over the Internet.

Text-to-speech. Speaking content on the desktop.

Audio. Playing sound and music files on the desktop.

Video. Playing movie files on the desktop.

Data Conversion and Analysis

Because NQL can work with data in different ways and perform conversions between common data types, it is possible for scripts to transform content. Coupled with NQL's communication capabilities, scripts are able to acquire content, analyze and transform it, and deliver the results to users in just the format they desire. Some of NQL's data-related capabilities include the following:

XML. Parsing, validating, creating, importing, and exporting eXtensible Markup Language, the standardized data language for representing all kinds of content.

HTML. Parsing, generating, importing, and exporting HyperText Markup Language, the data language of Web pages.

Delimited data. Reading, writing, importing, and exporting delimited data files.

Images. Creating, manipulating, and converting graphic images.

Date/Time. Obtaining current date and time, converting to and from various formats.

Filtering. Removing unwanted data.

Ranking. Analyzing and reordering information so that the most relevant items appear first.

Conversion. Transforming data into other formats.

Strings. Manipulating strings and text, and generating formatted output.

Automation and Distributed Processing

Connected applications are frequently automated and/or distributed. Automated solutions run all by themselves, either on a schedule or in response to events. Distributed solutions require cooperation between multiple programs working together as a team.

Scheduling. Performing tasks at set dates and times, and repeating tasks at regular intervals.

Inter-process communication. Communication between agents for cooperative processing on a large scale.

Synchronization. Locking and unlocking to manage access to shared resources and to coordinate processing.

Application control. Launching applications, on local or remote systems, and passing them command line arguments, data, or keystrokes.

Intelligent Behavior and Analysis

An essential ingredient in constructing reliable software with good decision-making capabilities is intelligence. NQL includes the following features that can be used to make software smarter:

Neural Networks. Adaptive software that learns and can be applied to pattern recognition, prediction, and organizational tasks.

Bayesian Inference. Predictive statistics that determine causes from effects.

Fuzzy Logic. Algorithms dealing with fine degrees of belief and probability.

Pattern Matching. Ability to recognize objects based on patterns or landmarks.

Categorization. Procedure of examining text and Web pages and assigning them categories.

Miscellaneous

In addition to the four main categories just covered, there are some additional NQL capabilities worth mentioning. They include the following:

Agent characters. Desktop characters that can display information, speak, and perform actions.

Registry access. Reading and writing information to the system registry.

System information. Determining operating system, current directory, available memory and resources, and other information.

System actions. Powering down, shutting down, or rebooting the computer, and logging off the current user.

Role Independence

Network Query Language stresses role independence, allowing you to utilize NQL wherever convenient. NQL scripts can execute on the desktop, on a server, as a Web application, and as a component. This means you can use NQL to be that missing puzzle piece in your enterprise, taking on the size, shape, and location needed.

Desktop

On workstations, NQL scripts may run as desktop agents. The agents can be out of sight or can take on a console appearance that resembles an MP3 player. The desktop is a logical choice for these kinds of applications:

- Applications that need to speak, play audio, or play video on the desktop
- Applications that need to interact with users
- Applications that need to be on the desktop for permission/security content reasons, such as an agent that reads incoming email messages
- Applications that need to perform on-screen notification, such as displaying messages or animating agent characters
- Applications that need to launch, export to, or otherwise interact with other desktop applications

Servers

On servers, NQL scripts can perform actions that service entire departments or organizations rather than individuals. Servers are logical places for these kinds of applications:

- Applications that need to respond to clients
- Applications that need to process large amounts of data based on queues, lists, or databases, such as shopping bots
- Middleware that interconnects two or more systems
- Scheduled applications that run in the background

Web Servers

On Web servers, NQL applications can operate as CGI applications, just as applications in languages like ASP, JSP, PHP, and Perl do. Web servers are a logical place for the following kinds of applications:

- Web applications, accessible from desktop Web browsers
- Mobile applications, accessible from PDAs and phones
- Applications that need to be accessible from anywhere in the world

Components

NQL can be called as a component, which gives you the luxury of continuing to use other languages and development environments but still tap into NQL's power when you need it. Some instances where you might want to call NQL as a component include the following:

- When you are extending existing projects already developed in another language
- When you have programmers already comfortable in working with a different language
- When you want to add communication, conversion, and intelligence features easily to extensible applications, such as Microsoft Office

Tutorial: Directory

Now that you know something about NQL's capabilities, let's develop and run a complete application. In this tutorial, you will create a script that examines a disk directory, creates a Web page that lists the files, and opens the Web page in a browser on the desktop. The result will be a Web page like the one shown in Figure 1.1.

There are four steps in this tutorial:

1. Launch the NQL development environment.
2. Enter the Directory script.
3. Run the Directory script.
4. Understand the Directory script.

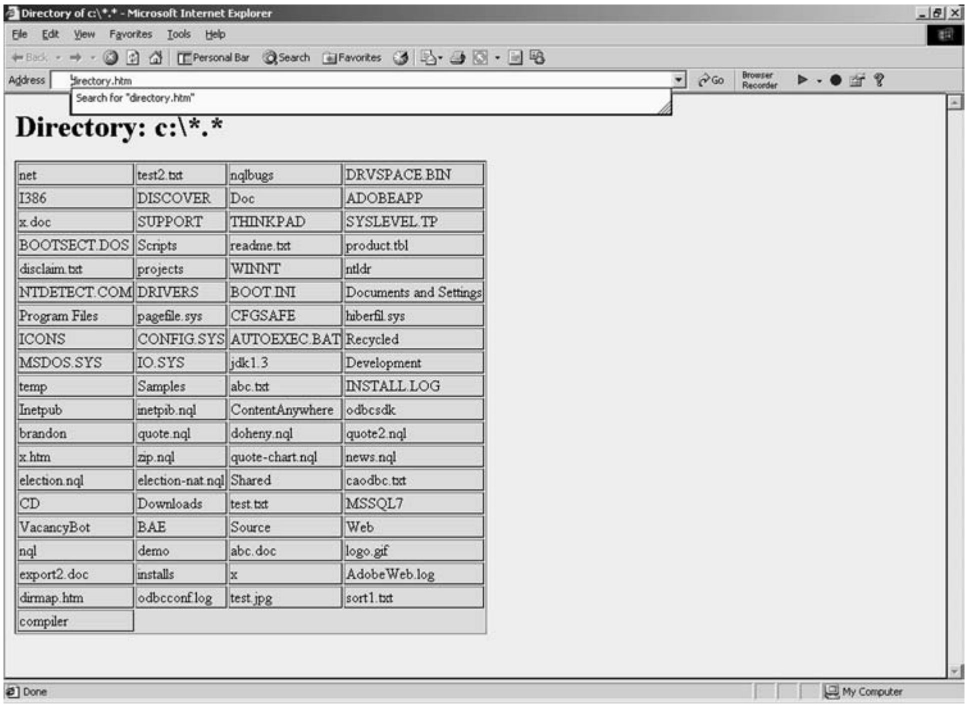
When you are finished with this tutorial, you will have seen how to do the following in NQL:

- Retrieve the names of files in a disk directory.
- Create a Web page and write HTML to it.
- Open a Web page in a browser on the desktop.

Let's begin!

Step 1: Launch the NQL Development Environment

Launch the NQL development environment. On a Windows system, this can be accomplished by clicking on the NQL Client desktop icon, or by selecting *Program Files, Network Query Language, NQL Client* from the Start Menu. On other platforms, you should have a desktop icon and/or command line method of launching the NQL Client.


Directory of c:\*.* - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Personal Bar Search Favorites
Address directory.htm
Search for "directory.htm"
Go
Browser Recorder
Done
My Computer

Directory: c:\*.*

net	test2.txt	nqlbugs	DRVSPACE.BIN
I386	DISCOVER	Doc	ADOBEAPP
x.doc	SUPPORT	THINKPAD	SYSLEVEL.TP
BOOTSECT.DOS	Scripts	readme.txt	product.tbl
disclaim.txt	projects	WINNT	ntldr
NTDETECT.COM	DRIVERS	BOOT.INI	Documents and Settings
Program Files	pagefile.sys	CFGSAFE	hiberfil.sys
ICONS	CONFIG.SYS	AUTOEXEC.BAT	Recycled
MSDOS.SYS	IO.SYS	jdk1.3	Development
temp	Samples	abc.txt	INSTALL.LOG
Inetpub	inetpib.nql	ContentAnywhere	odbcsdk
brandon	quote.nql	doheny.nql	quote2.nql
x.htm	zip.nql	quote-chart.nql	news.nql
election.nql	election-nat.nql	Shared	caodbc.txt
CD	Downloads	test.txt	MSSQL7
VacancyBot	BAE	Source	Web
nql	demo	abc.doc	logo.gif
export2.doc	installs	x	AdobeWeb.log
dirmap.htm	odbcconf.log	test.jpg	sort1.txt
compiler			

Figure 1.1 Directory Web page.

At this point, you should have the NQL development environment active on your desktop, with an empty code window. Now you're ready to enter your first NQL script.

Step 2: Enter the Directory Script

In the NQL development environment, enter the script shown in Listing 1.4 and save it under the name directory.nql. Enter the script, then save it by clicking the *Save* toolbar button (which has a disk icon).

If you prefer, you may copy directory.nql from the companion CD. If you have installed the companion CD on your system, you will have all of the book's tutorial materials in a \Tutorials directory on your hard drive. Click the *Open* toolbar button (folder icon), and select directory.nql from the Tutorials\ch01 folder.

At this point, the script in Listing 1.4 should be in your code window, either because you entered it by hand or because you opened the script from the companion CD. You are now ready to run the script.

Step 3: Run the Directory Script

Now, take a deep breath and run the script. You can do this by selecting *Build, Run* from the menu, clicking the Run toolbar button, or pressing F5. Almost immediately, a

```
//directory - creates a Web page that shows the contents of a disk
directory

string path = "c:\\"

int size = length(path)

path = path & "*.*"

if !dir(path) { end }

create "directory.htm"

write "<html><head><title>Directory of {path}</title></head>"
write "<body bgcolor=#000088 text=#ffffff>"
write "<h1>Directory: {path}</h1>"
write "<table border=1 bgcolor=#0000FF><tr>"

while nextline(file)
{
    file = mid(file, size+1)
    write "<td>" file "</td>"
    files = files+1
    if files>=4
    {
        write "</tr><tr>"
        files = 0
    }
}

write "</tr></table>"
write "</body></html>"

close

opendoc "directory.htm"

end
```

Listing 1.4 Directory script.

Web page should open up on the desktop that shows a directory of the files in c:\. If this does not happen, check the following:

1. Make sure you have a c: drive. If you don't, change the path in the script.
2. Make sure there are files in the c:\ root directory. If there aren't, change the path in the script.

3. Ensure that you are running directory.nql from a disk directory that you can write to. If not, copy the script to a different location, such as your primary hard drive.
4. Ensure that you have a default application for displaying .htm files (most systems assign a Web browser for this purpose).
5. Check the NQL client's Errors tab for error messages.

At this point you should have seen the directory script run, generating a Web page that lists the files in a disk directory. Although we haven't discussed how it works yet, you've seen the results of an NQL script first-hand, in this case exercising NQL's abilities to read disk directories, generate Web pages, and open files on the desktop. In the next step, we'll dissect the script and explain exactly how it works.

Step 4: Understand the Directory Script

We now want to make sure we understand every part of the directory script. The first line is a comment line.

```
//directory - creates a Web page that shows the contents of a disk
directory
```

Next, a variable named *path* is declared and initialized. The word string indicates that *path* is a string variable. The path is set to c:\, the directory to be displayed in the Web page.

```
string path = "c:\\"
```

A second variable named *size* is now declared and initialized. The data type is integer, and the value is the length of the path string. The variable *size* will be used to separate the filename from the path prefix later in the script.

```
int size = length(path)
```

The path is appended with *.* so that it is now a full wildcard directory specification of c:\*.*.

```
path = path & "*.*"
```

A directory of the path is requested using the *dir* function. If a directory cannot be obtained, *dir* returns false and the script ends. If the directory is obtained, it is stored on the stack as a list of filenames separated by newlines.

```
if !dir(path) { end }
```

It is now time to create the Web page and write some initial HTML to it. The *create* statement creates a new file and opens it, overriding any previous file of the same name. We call the Web page directory.htm. The *write* statements output some initial HTML, ending in the beginnings of a table.

```
create "directory.htm"

write "<html><head><title>Directory of {path}</title></head>"
write "<body bgcolor=#000088 text=#ffffff>"
write "<h1>Directory: {path}</h1>"
write "<table border=1 bgcolor=#0000FF><tr>"
```

Now to loop through the filenames returned by the *dir* function earlier. A *while* loop uses *nextline* to get the next line from the data on the stack. The loop runs as long as there are files left to process, leaving the current file to work with in the variable named *file*.

```
while nextline(file)
{
```

Within the loop, the filename is reduced to eliminate the path (drive and folder) portion, using the *mid* function and the *size* variable calculated earlier. The file name is then written out as an HTML table element. If more than four files across have been written, additional HTML is written out to move to the next table row.

```
    file = mid(file, size+1)
    write "<td>" file "</td>"
    files = files+1
    if files>=4
    {
        write "</tr><tr>"
        files = 0
    }
}
```

Once all of the files have been written to the HTML, the table needs to be finished and the final HTML suffix code needs to be output. More *write* statements take care of this.

```
write "</tr></table>"
write "</body></html>"
```

The HTML file is now finished and can be closed with *close*.

```
close
```

Lastly, we want to open the newly created Web page on the desktop. The *opendoc* statement accomplishes this.

```
opendoc "directory.htm"
```

An *end* statement terminates the script.

```
end
```

At this point, you've not only seen an NQL script in action, you've also had a look under the hood as well. The remaining chapters of this book go into more detail about the families of NQL statements and functions.

Further Exercises

You could amend this example in a number of ways. Once you've learned more about NQL from the chapters that follow, you may want to come back to this example and try your hand at extending it. Here are some interesting modifications that can be made:

- Show different types of files in different colors. For example, .doc files in blue and .xls files in green.
- Show more information about each file than just its name, such as the size of the file.
- For text files, display an excerpt from the file.

Chapter Summary

Network Query Language is oriented toward the development of connected applications such as bots, agents, spiders, middleware, Web applications, and mobile applications. The design philosophies behind NQL yield power, clarity, and rapid development while retaining familiarity to programmers and permitting extensibility.

NQL contains over 500 building blocks that fall into four main categories: communications, data conversion, automation/distributed processing, and intelligent behavior. Powerful scripts can be created quickly from just a handful of these building blocks.

NQL features role independence, which means it can be put to use anywhere it is needed. Scripts can run on the desktop, on servers, on Web servers, and as components. Although complete applications can be developed in NQL, it is also possible to combine NQL with other languages.

You've had your first look at NQL. Hopefully you've seen a lot to get excited about. The chapters that follow explore the language's rules and capabilities in detail, with plenty of examples along the way.