THE EXPERT'S VOICE® IN WEB DEVELOPMENT

JSON Quick Syntax Reference

Wallace Jackson

Apress®

JSON Quick Syntax Reference

Wallace Jackson

Apress®

JSON Quick Syntax Reference

Wallace Jackson
Lompoc, California, USA

ISBN-13 (pbk): 978-1-4842-1862-4 ISBN-13 (electronic): 978-1-4842-1863-1
DOI 10.1007/978-1-4842-1863-1

Library of Congress Control Number: 2016941347

Managing Director: Welmoed Spahr
Lead Editor: Steve Anglin
Technical Reviewer: Chád Darby
Editorial Board: Steve Anglin, Pramila Balan, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Michelle Lowman, James Markham, Susan McDermott, Matthew Moodie, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing
Coordinating Editor: Mark Powers
Copy Editor: Tiffany Taylor
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com/9781484218624. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

This JSON syntax *book is dedicated to everyone in the open source community who is working diligently to make professional new media application development software and content-development tools freely available for application developers to use to achieve our creative dreams and financial goals. I also dedicate this book to my father, Parker Jackson, my family, my life-long friends, my content production ranch neighbors, and my business partners, for their continual assistance and those relaxing, beautiful sunset BBQs under pink clouds here on the Point Conception Peninsula.*

Contents at a Glance

About the Author xiii

About the Technical Reviewer xv

Acknowledgments xvii

Introduction xix

■ Chapter 1: Exploring a JSON Integrated Development Environment 1

■ Chapter 2: An Introduction to JSON: Concepts and Terminology 15

■ Chapter 3: The JSON Schema: JSON Structure Validation 21

■ Chapter 4: Objects and Object-Oriented Programming: OOP Primer 31

■ Chapter 5: JSON Arrays: Serialized Data Store Structures........... 51

■ Chapter 6: JSON Data Values: Types of Data that JSON Supports.... 59

■ Chapter 7: JSON Character Values: Defining Strings in JSON 65

■ Chapter 8: JSON Numeric Values: Defining Numbers in JSON 71

■ Appendix A: NetBeans 8.1: Setting Up a JSON Integrated Development IDE 83

■ Appendix B: Eclipse Mars: Setting Up a JSON Integrated Development IDE 99

■ Appendix C: IntelliJ IDEA: Setting Up a JSON Integrated Development IDE 117

Index 139

Contents

About the Author xiii

About the Technical Reviewer xv

Acknowledgments xvii

Introduction xix

■Chapter 1: Exploring a JSON Integrated Development Environment 1

NetBeans 8.1: The Intelligent JSON IDE 1

NetBeans 8.1 Is Smart: Code Editing in Hyper-Drive 2

NetBeans 8.1 Is Extensible: Coding in All Languages 3

NetBeans 8.1 Is Efficient: Project-Management Tool 3

NetBeans 8.1 UI Design: Responsive Web Design 4

NetBeans 8.1 Styling: CSS3 Style Editing Support 4

NetBeans 8.1 Debugging: Squash Those Bugs! 5

NetBeans 8.1 Optimization: Program Code Profiler 5

Creating a JSON Project: A Bootstrap Project 5

Summary 13

■Chapter 2: An Introduction to JSON: Concepts and Terminology 15

What Is JSON? 15

JSON Is Lightweight: Simpler Is Usually Superior 15

JSON Is Text-Based: 100% Language Independent 16

JSON Structures: Data Objects and Data Arrays.......16
JSON Is JavaScript Compatible: Easy Integration.......17
JSON Open Licensing: Free for Commercial Use.......20
Summary.......20
■Chapter 3: The JSON Schema: JSON Structure Validation.......21
JSON Schema: Concepts and Definitions.......21
JSON Schema Advantage: Clear JSON Description.......22
JSON Hyper-Schema Advantage: Links and Forms.......22
JSON Schema Core: Language Definition.......22
Summary.......29
■Chapter 4: Objects and Object-Oriented Programming: OOP Primer.......31
Object-Oriented Programming: Overview.......31
Java OOP Concepts: Hard Object Construction.......32
Java Objects: Virtual Reality Using OOP with Java.......39
JavaScript OOP Concepts: Hard and Soft Objects.......47
Summary.......50
■Chapter 5: JSON Arrays: Serialized Data Store Structures.......51
An Overview of Arrays: Data Structures.......51
Declaring a JavaScript Array: Variable Declaration.......52
Accessing a JavaScript Array: Using the Index.......53
Defining a JSON Array: Using the Colon Operator.......54
Java JSON Support: JSON Utility Classes.......55
JSON Object Model: Java Object and Array Builder.......55
The JSON Streaming Model: Parser and Generator.......57
Summary.......57

■Chapter 6: JSON Data Values: Types of Data that JSON Supports 59

JSON Value: Supported JSON Data Types........ 59

String Value: The Sequence or Array of Characters 59

Number Value: Representing the World 60

Boolean Value: True or False, On or Off, Yes or No 61

Null Values: A Placeholder for Future Data Values 62

Java and JavaScript: Boolean and Null 63

Summary........ 64

■Chapter 7: JSON Character Values: Defining Strings in JSON 65

JSON String: Unicode Character Support 65

Background of String Values 67

Escaping Control Characters: JSON Examples 67

Java and JavaScript: Using String Values 69

Java String Values: Java's String Class and Object 69

JavaScript String Values: Primitives and Objects 69

Summary........ 70

■Chapter 8: JSON Numeric Values: Defining Numbers in JSON 71

Number Types: Integer, Real, Exponential........ 71

JSON Number: Wide Ranging Numerics........ 71

Positive Integers: Positive Whole Number Values 72

Negative Integers: Negative Whole Number Values 73

Positive Real Number Values: Positive Fractions 74

Negative Real Number Values: Negative Fractions 76

Exponential Number Values: Scientific Notation 78

Summary........ 82

Appendix A: NetBeans 8.1: Setting Up a JSON Integrated Development IDE 83

Creating a JSON Development Workstation 83

Hardware Foundation 84

Open Source Software 85

Java 8: Installing the Foundation for NetBeans 8.1 85

NetBeans 8.1: Downloading the NetBeans JSON IDE 91

GIMP 2.8: Digital Image Editing and Compositing 93

Blender: 3D Modeling, Rendering, and Animation 93

Inkscape: Digital Illustration and Digital Painting 94

Audacity: Digital Audio Editing and Special Effects 94

Visual Effects: BlackMagic Design Fusion 8.0 VFX 95

Digital Video Editing: EditShare Lightworks 12.6 96

Office Productivity Suite: Apache OpenOffice 4.1.2 97

Summary 97

Appendix B: Eclipse Mars: Setting Up a JSON Integrated Development IDE 99

Creating a JSON Development Workstation 99

Hardware Foundation 100

Open Source Software 101

Java 8: Installing the Foundation for Eclipse Mars 101

Eclipse 4.5: Installing the Eclipse Mars JSON IDE 107

GIMP 2.8: Digital Image Editing and Compositing 112

Blender: 3D Modeling, Rendering, and Animation 112

Inkscape: Digital Illustration and Digital Painting 113

Audacity: Digital Audio Editing and Special Effects 113

Visual Effects: BlackMagic Design Fusion 8.0 VFX 114

Digital Video Editing: EditShare Lightworks 12.6 115

Office Productivity Suite: Apache OpenOffice 4.1.2 115

Summary 116

Appendix C: IntelliJ IDEA: Setting Up a JSON Integrated Development IDE 117

Creating a JSON Development Workstation 117

Hardware Foundation 118

Open Source Software 119

Java 8: Installing the Foundation for IntelliJ IDEA 119

IntelliJ IDEA: Downloading the IntelliJ IDEA for JSON 125

GIMP 2.8: Digital Image Editing and Compositing 135

Blender: 3D Modeling, Rendering, and Animation 135

Inkscape: Digital Illustration and Digital Painting 136

Audacity: Digital Audio Editing and Special Effects 136

Visual Effects: BlackMagic Design Fusion 8.0 VFX 137

Digital Video Editing: Editshare Lightworks 12.6 137

Office Productivity Suite: Apache OpenOffice 4.1.2 138

Summary 138

Index 139

About the Author

Wallace Jackson has written for several leading multimedia publications about production for the media content development industry, beginning with an article about advanced computer processing architectures for the centerfold (a removable "mini issue" insert) of the original issue of *AV Video Multimedia Producer* magazine distributed at the SIGGRAPH trade show.

Wallace has written for a number of popular publications regarding his work in interactive 3D and new media advertising campaign design, including *3D Artist* magazine, *Desktop Publisher Journal*, *CrossMedia*, *Kiosk*, *AV Video Multimedia Producer*, and *Digital Signage* magazine**,** as well as many other publications.

Wallace has authored more than 20 Apress book titles, including several in the ever-popular Apress *Pro Android* series, Java and JavaFX game engine development titles, digital image compositing titles, digital audio editing titles, digital video editing titles, digital illustration titles, VFX special effects titles, digital painting titles, Android 6 new media content production titles, and JSON and HTML5 titles.

In his current book on digital video editing and effects, Wallace focuses on Corel VideoStudio Ultimate X9 digital video software and uses it to demonstrate digital video editing, as well as digital video effects and compositing fundamentals, to beginners who want to become more digital video editing savvy.

Wallace is currently the CEO of MindTaffy Design, a new media advertising agency that specializes in new media content production and digital campaign design and development. The company is located by La Purisima State Park in Northern Santa Barbara County, on the Point Conception Peninsula, halfway between the clientele in Silicon Valley to the north and Hollywood, The OC, West Los Angeles, and San Diego to the south.

MindTaffy Design has created open source, technology-based (HTML5, JavaScript, Java 8, JavaFX 8, and Android 6.0) digital new media i3D content deliverables for more than a quarter century, since January 1991.

The company's clients consist of a significant number of international brand manufacturers, including IBM, Sony, Tyco, Samsung, Dell, Epson, Nokia, TEAC, Sun Microsystems (Oracle), Micron, SGI, KDS USA, EIZO, CTX International, KFC, Nanao USA, Techmedia, EZC, and Mitsubishi Electronics.

Wallace received his undergraduate BA degree in business economics from the University of California at Los Angeles (UCLA) and his graduate degrees in MIS/IT and business information systems design and implementation from University of Southern California located in South Central Los Angeles (USC).

Wallace also received post-graduate degrees from USC in entrepreneurship and marketing strategy, and he completed the USC Graduate Entrepreneurship Program. Wallace earned his two USC degrees while at USC's night-time Marshall School of Business MBA Program, which allowed him to work full-time as a COBOL and RPG-II programmer while completing his business and IT degrees.

You can visit Wallace's blog at `www.wallacejackson.com` to view his multimedia production content. You can also follow him on Twitter at `@wallacejackson` or connect with him on LinkedIn.

About the Technical Reviewer

Chád ("Shod") Darby is an author, instructor, and speaker in the Java development world. As a recognized authority on Java applications and architectures, he has presented technical sessions at software development conferences worldwide (in the United States, UK, India, Russia, and Australia). In his 15 years as a professional software architect, he's had the opportunity to work for Blue Cross/Blue Shield, Merck, Boeing, Red Hat, and a handful of startup companies.

Chád is a contributing author to several Java books, including *Professional Java E-Commerce* (Wrox Press), *Beginning Java Networking* (Wrox Press), and *XML and Web Services Unleashed* (Sams Publishing). Chád has Java certifications from Sun Microsystems and IBM. He holds a BS in computer science from Carnegie Mellon University.

You can visit Chád's blog at `www.luv2code.com` to view his free video tutorials on Java. You can also follow him on Twitter at `@darbyluvs2code`.

Acknowledgments

I would like to acknowledge all my fantastic editors and their support staff at Apress, who worked long hours and toiled diligently on this book to make it the very best JSON syntax title currently on the market.

I would like to thank the following people:

Steve Anglin, for his work as the acquisitions editor for the book and for recruiting me to write development titles at Apress covering widely popular open source content-development platforms (Android, Java, JavaFX, HTML5, CSS3, JS, JSON, and so on).

Matthew Moodie, for his work as development editor and for his experience and guidance during the process of making this a fantastic JSON title.

Mark Powers, for his work as the coordinating editor and for his constant diligence in making sure I either hit my chapter-delivery deadlines or far surpassed them.

Chád Darby, for his work as the technical reviewer and for making sure I didn't make technical mistakes.

Introduction

JSON Quick Syntax Reference is intended for individuals who wish to add JSON to their programming quiver. These include application developers, web site developers, user interface design architects, user experience designers, social media application developers, and just about anyone who's interested in generating superior-quality applications that use JSON to talk to the server from their client-side applications.

This book covers JSON syntax and related concepts; JSON editing; integrated development environments (IDEs) such as NetBeans, IntelliJ, and Eclipse; and how these relate to open source OOP languages such as Java and JavaScript. This equates to JSON syntax, JSON IDE, and OOP fundamentals combined in one book, including technical terms, topics, concepts, and definitions.

Each chapter builds on the knowledge learned in the previous chapter. Thus, later chapters in the book have you creating JSON objects, arrays, and data values using diagrams from the `json.org` web site, which guide you in making valid data constructs.

At the end of this book, three appendixes explain how to assemble a complete JSON development environment and workstation for each of the three major IDEs, including all related new media content production software packages.

In Chapter 1, you take a tour of the NetBeans 8.1 IDE, which supports HTML5, JavaScript, CSS, and JSON development inherently in its feature set.

Chapter 2 looks at basic JSON terminology, concepts, and requirements.

Chapter 3 examines at the JSON Schema, which provides a roadmap regarding what JSON supports and how to validate JSON data definitions.

In Chapter 4, you learn about JSON `Objects` and the syntax used to create them. The chapter also reviews OOP languages, in an object-oriented programming primer.

In Chapter 5, you explore JSON arrays and their syntax and learn how to create arrays of data for JSON `Objects` and JSON `Array` constructs.

Chapter 6 introduces the supported types of JSON data values you can use to construct JSON `Objects` and JSON `Arrays`, along with background information regarding these data types and how they are used in popular computer programming languages.

Chapter 7 covers the important JSON `String` data type and how it is used to create text-based data constructs for JSON `Objects` and JSON `Arrays`.

Chapter 8 covers the important JSON `Number` data type and how it is used to represent a wide range of numeric data for JSON `Objects` and JSON `Array` constructs.

If you're interested in creating JSON-compatible computer applications, and you want to learn all the basic JSON syntax and OOP fundamentals as well as how everything works together in the IDE, this is the reference book for you to read to start your journey to JSON mastery. The book is overflowing with tips, tricks, tools, topics, terminology, techniques, concepts, and syntax. *JSON Quick Syntax Reference* will give you a boost to transition from being a JSON neophyte to being the knowledgeable JSON data definition professional you seek to become, at least where a JSON content production pipeline is concerned.

CHAPTER 1

Exploring a JSON Integrated Development Environment

Let's get started in Chapter 1 by learning everything you can about the NetBeans 8.1 JSON integrated development environment (IDE), because that is the primary piece of software you use in this book to create JSON projects. I chose the NetBeans 8.1 IDE to use for the book, even though I cover all the major IDE software in the appendixes, because it has native JSON project-creation workflows that you see in this chapter. The other IDEs currently require plug-ins to support JSON development. You start by learning about the NetBeans 8 IDE is because it is the front end, or window, through which you look at JSON development projects. Each chapter builds on information from previous chapters, so a logical progression is from how an IDE works, to JSON concepts and terminology, to objects, data structures, applications, and so forth.

NetBeans 8.1 is the official IDE for the Java 8 JDK, and it's rapidly growing in popularity for HTML5, CSS3, and JavaScript development as well; as such, this is the IDE you should use for this book. Also, a major player (Oracle) is behind the software, and it is freely available for commercial usage (open source). That is not to say you cannot use another IDE, such as Eclipse or IntelliJ. These are the official IDEs for Android 4.x (32-bit) and Android 6.x (64-bit), respectively; but I prefer to use NetBeans 8.1 for my new media apps and game development for Java, JavaFX, HTML5, CSS3, and JavaScript software development, as well as for HTML5, XML, JSON markup, and JS programming.

The first thing you do in this chapter is look at what has been added in NetBeans 8.1. Next you examine the various attributes of the NetBeans 8.1 IDE that make it an invaluable tool for JSON Development. You'll see all the cool things JSON will do for you during the course of this book. Finally, you learn how to create JSON projects by using the NetBeans 8.1 New Project dialogs, which specifically support the creation of JSON projects in NetBeans.

NetBeans 8.1: The Intelligent JSON IDE

I assume that you already have a professional-level workstation in place for new media content development and JSON development, as outlined in detail in Appendix A. IDE installations are covered in the appendixes, including each of the three open source IDEs. I also assume that you have removed all your outdated JDKs and IDEs and made sure you have the latest Java and IDE software installed on your PC.

W. Jackson, *JSON Quick Syntax Reference*, DOI 10.1007/978-1-4842-1863-1_1

If you're new to coding and do not have the appropriate workstation, go to Walmart or PriceWatch.com, and purchase an affordable multicore (4-, 6-, or 8-core) 64-bit computer running Windows 10. It should have 4, 6, or 8 GB of DDR3 (1333 or 1600 memory-access speeds) system memory at the very least, and at least a 500 GB or even 1 TB hard disk drive.

NetBeans 8.1 Is Smart: Code Editing in Hyper-Drive

Although it is true that an IDE is essentially like a word processor, it is geared toward writing text for coding or markup rather than business documents. IDEs such as NetBeans 8.1 can lend a lot more power to your programming work process than a word processor brings to the document-authoring work process.

For instance, a word processor does not make suggestions in real time regarding the content you're writing for your business, whereas the NetBeans 8 IDE actually looks at what you are coding while you're actually writing the code and helps you write programming statements and constructs. One of the things NetBeans 8.1 will do is finish lines of code for you, as well as apply colors to your code statements to highlight different types of constructs.

NetBeans 8 also applies industry standards to your code, indenting the code to make it easier to read for both you and for the members of your JSON application development team.

NetBeans 8 can provide matching code-structure brackets, colons, and semicolons, so you don't get lost when you are creating complex, deeply nested, or dense programming constructs found in modern programming languages, such as those supported by NetBeans 8.1. You create such constructs as this book progresses in complexity and you go from JSON beginner to JSON developer; I point out JSON code that is dense, complex, or deeply nested as you encounter it.

NetBeans 8.1 can also provide bootstrap code such as the JSON application bootstrap code you create a bit later in the chapter. (I know you're eager to get started creating JSON projects and structures as soon as possible.)

As your code becomes more complex, it also becomes a far better candidate for *code refactoring*, which can make your code easier to understand, easier to upgrade, and far more efficient. NetBeans 8 can refactor your program's code automatically. Code refactoring is the process of changing the structure of existing computer code so it is more efficient or scalable without changing its external behavior—that is, what the programming logic (your code) accomplishes. For instance, you could take Java 6 or Java 7 code and make it more efficient by implementing lambda expressions using Java 8, or even Java 9 when it is released later this year.

NetBeans includes *pop-up helper* dialogs containing methods, constants, asset references, and suggestions regarding how to construct programming statements. For instance, NetBeans 8.1 can suggest when it might be appropriate to use the powerful new Java 8 lambda expression features to make your code more streamlined and multithread compatible.

NetBeans 8.1 Is Extensible: Coding in All Languages

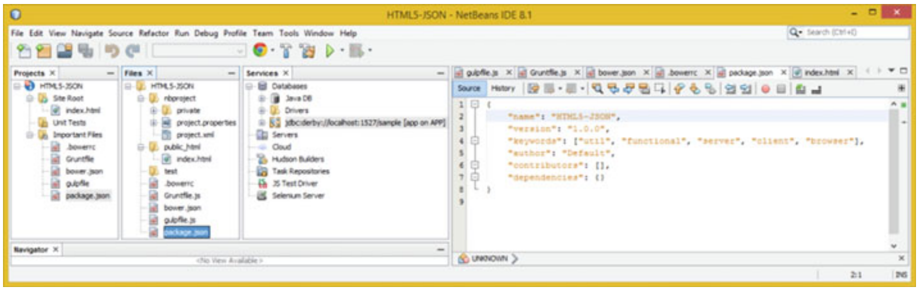
Another thing a word processor doesn't do is allow you to add features to it. NetBeans does, using its plug-in architecture. This type of architecture is said to be *extensible*, which means if needed, it can be extended to include additional features. For instance, if you wanted to extend NetBeans 8.1 to allow you to program using Python, you could. NetBeans 8.1 also supports older languages such as COBOL and BASIC in this fashion, although the majority of popular consumer electronic devices today use Java, XML, JavaScript, and HTML5. I googled this to be sure, and there are people coding in both Python and COBOL in NetBeans 8; this is real-world proof that the NetBeans 8.1 IDE is indeed *completely extensible*.

Probably due to its extensibility, the NetBeans 8.1 IDE supports a number of powerful programming languages, including C, C++, Java SE, JavaDoc, JavaScript, XML, HTML5, and CSS on the client side; and PHP, Groovy, Java EE, and Java Server Pages (JSP) on the server side. *Client-side* software runs on the device the end user is holding or using (in the case of an iTV set), and *server-side* software runs remotely on a server somewhere and talks to the end user over the Internet or similar network. Client-side software is more efficient, because it is local to the device it is running on and thus is more scalable: no server is involved that might experience overload as more people use the software at any given point in time.

NetBeans 8.1 Is Efficient: Project-Management Tool

Your programming IDE needs to be able to manage projects that can grow to become massive, involving over a million lines of code, contained in hundreds of folders in the project folder hierarchy, and involving thousands of files or new media assets. Clearly, project-management features should be extremely robust for any mainstream IDE, and NetBeans 8.1 contains a plethora of project-management features. These allow you to look at JSON development projects and the corresponding files and their interrelationships in several different ways.

NetBeans 8.1 provides four primary project-management views, or *panes*, that you can use to see the various types of interrelationships in your project. I jumped ahead to the end of the chapter, where the example JSON project has already been created, and took the screen shot shown in Figure 1-1. It shows the primary project-management panes open in this new project; you can see the types of project-related information they show you.



```
{
    "name": "HTML5-JSON",
    "version": "1.0.0",
    "keywords": ["util", "functional", "server", "client", "browser"],
    "author": "Default",
    "contributors": [],
    "dependencies": {}
}
```

Figure 1-1. *Projects, Files, and Services project-management panes*

The *Projects* pane shows HTML5 and JSON source files and important files that make up your JSON project; it's at far left in Figure 1-1. The next pane is *Files*, which shows the `nbproject` folder and file hierarchy on your hard disk drive. The *Services* pane, to the right of that, shows databases, servers, cloud, Hudson Builders, task repositories, JS Test Driver, and Selenium Server, if these are being used in the project. These are primarily server-side technologies, and technologies used with a development team, so this book doesn't get into them in much detail.

The Projects pane should always be left open on the left side of your IDE, as you see in all the figures in this chapter. This pane is the primary access point for all of your project source code and assets (content). The Files pane shows not only the project folder and its file hierarchy but also data, HTML5 files, JSON files, and all related project files.

The Navigator pane, shown at the bottom of your NetBeans 8 IDE under the Projects, Files, and Services panes, shows the relationships that exist in your HTML5, JavaScript, CSS3, XML, XSL, and JSON code structures.

NetBeans 8.1 UI Design: Responsive Web Design

NetBeans 8 also has Design a GUI drag-and-drop design tools for a plethora of platforms, including Java SE, Java EE, Java ME, JavaFX, and Java Swing, as well as C, C++, PHP, HTML5, JavaScript, and CSS3. NetBeans 8.1 provides *visual editors* that can write the application's UI code for you, so all you have to do is make what's on the screen look the way you want it to look in your JSON application.

NetBeans 8 IDE supports the use of *responsive web design* architecture, allowing you to select among various form factors for your web page. NetBeans 8.1 can quickly lay out visual elements in your web design using your browser of choice with any form factor: portrait, widescreen, or super-widescreen. You can select from form-factor presets ranging from smartphones to tablets to laptops to iTV sets, and using a portrait or landscape screen display topology.

NetBeans 8.1 Styling: CSS3 Style Editing Support

In addition to enhancing browser capabilities, NetBeans' CSS Style Editor is aware of any new media assets your browser is currently displaying and automatically places edits to the CSS markup referencing those new media assets. Editing support is included for LESS and SASS preprocessors is also provided, including syntactic and semantic coloring for CSS language constructs, automatic indentation, reformatting, intelligent code folding, and file templates.

Code-completion and refactoring tools are available with variables and mixins. The deep integration of WebKit browsers allows you to preview changes that you make in a CSS Style Edit window live in your browser as they happen. No more guessing what CSS3 changes will look like until you refresh the browser; the changes are real-time, taken directly from the source code.

You can also inspect using the browser, and the NetBeans CSS Style Editor window can automatically display CSS rules for each style element you select in the browser window. It does all this automatically. You can edit CSS3 from the CSS Style Editor window using properties fields, or you can go into the CSS3 source code and edit the code directly, using the NetBeans code-completion and inline documentation helpers.

CSS3 Style Editor previews are also available for mobile devices. These include the Google Chrome embedded WebKit browser, Chrome on Android, and Safari on iOS mobile devices. There's an excellent chance this feature will soon work on other WebKit browsers as well, such as in Opera OS.

NetBeans 8.1 Debugging: Squash Those Bugs!

There's a logical assumption across computer programming languages that the negative impact of a *bug* on a programming project—that is, code that does not do exactly what you want it to—increases in magnitude the longer it remains unfixed. For this reason, bugs must be squashed as soon as they are born, so to speak. NetBeans 8's bug-finding code-analysis tool, an integrated NetBeans debugger, and NetBeans 8.1 integration with the third-party FindBugs project on SourceForge.net (http://findbugs.sourceforge.net, if you want the stand-alone version) supplement NetBeans' code-correcting and efficiency tools, discussed at the beginning of this section.

Your JSON, HTML5, CSS3, and JavaScript code probably will not become complicated until later in the book, so later chapters cover how these advanced tools work when you need to use them and your knowledge is more advanced.

NetBeans 8.1 Optimization: Program Code Profiler

NetBeans has a *profiler* that looks at your JSON code while it is running and tells you how efficiently it uses memory and CPU cycles. This allows you to refine your code and make it more efficient in its use of key system resources, which is important for JSON development because it affects the smoothness of play on systems that are not as powerful (single-core and dual-core CPUs, for instance).

The profiler is a *dynamic* software-analysis tool. It looks at your Java code while it's running, whereas the FindBugs code-analysis tool is *static*: it looks at your code in the editor, when it is not compiled and running in system memory. The NetBeans debugger lets you step through your code while it is running; this tool can be viewed as a hybrid of static (editing) and dynamic (executing) code-analysis modes.

Creating a JSON Project: A Bootstrap Project

Let's get down to business and see how to create a bootstrap JSON project foundation automatically using NetBeans 8.1. This takes you toward the ultimate goal of being up to speed regarding JSON development tool options, terms, principles, structures, and projects. This example shows you how to create an empty JSON and HTML5 project using the NetBeans New Project dialog.

Click the Quick Launch icon on your taskbar (or double-click the NetBeans icon on your desktop), and launch NetBeans 8. You see the NetBeans startup screen, shown in Figure A-11 in Appendix A. Close the Welcome startup screen using the X in the tab. In the new (empty) IDE, click the New Project icon at upper left; it's shown selected in Figure 1-2. Doing so opens the NetBeans New Project dialog.

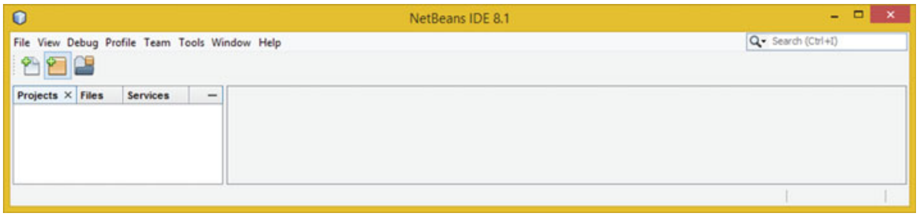
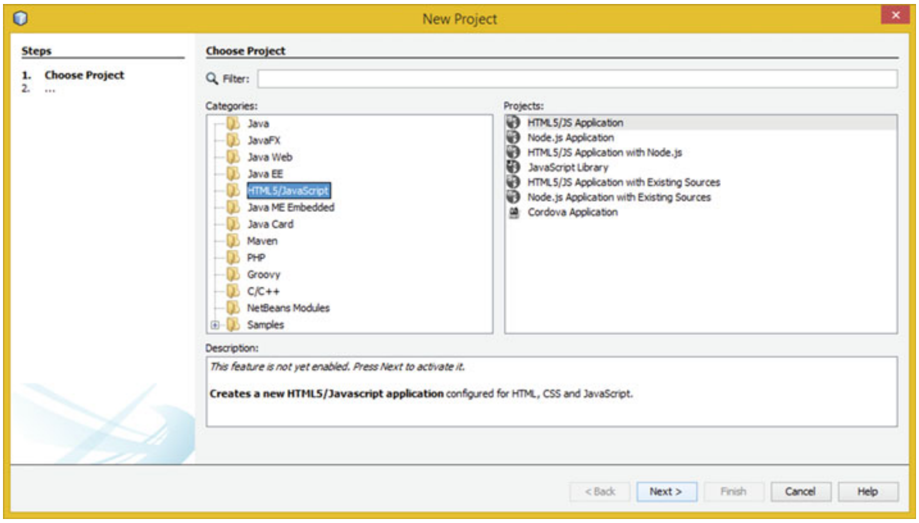


Figure 1-2. *Launch NetBeans 8.1, and click the New Project icon*

In case you're wondering, you can access the start page and its tutorials at any time! To do this, in the NetBeans 8.1 IDE, choose Help ➤ Start Menu.

To create an empty skeleton or *bootstrap* NetBeans 8 JSON project, you can use the NetBeans 8.1 New Project series of dialogs. This is one of those helpful JSON programming features that I talked about in the previous section; it creates a starting-point project with all the correct JSON files (ending in `.json`), HTML5 files (ending in `.html`), and JavaScript files (ending in `.js`), as you saw in Figure 1-1. You learn about these components in the rest of this book.

The first dialog in the series is Choose Project, shown in Figure 1-3, where you select the HTML5/JavaScript option. The *JS* in JSON stands for JavaScript, so this is the logical place to start. Click the Next button to proceed to the next in the series of New Project dialogs, where you can name your project and set its location on your JSON development workstation.

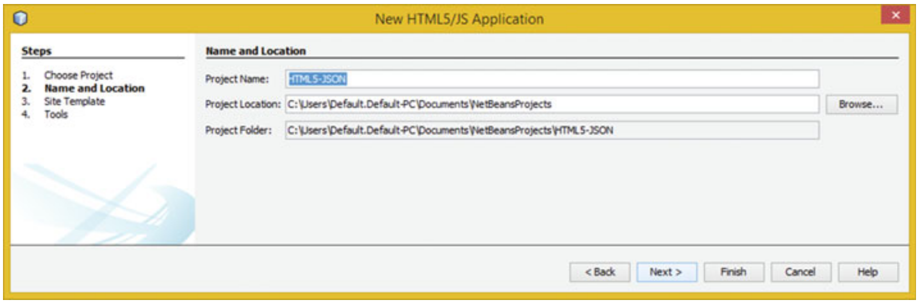


This feature is not yet enabled. Press Next to activate it.

Creates a new HTML5/Javascript application configured for HTML, CSS and JavaScript.

Figure 1-3. *Select HTML/JavaScript in the Choose Project dialog*

In the Name and Location dialog, shown in Figure 1-4, enter a Project Name of **HTML5+JSON**. Leave the Project Location and Project Folder fields set with the default data specified by NetBeans 8.1. Once everything is configured, click Next to advance.


New HTML5/JS Application

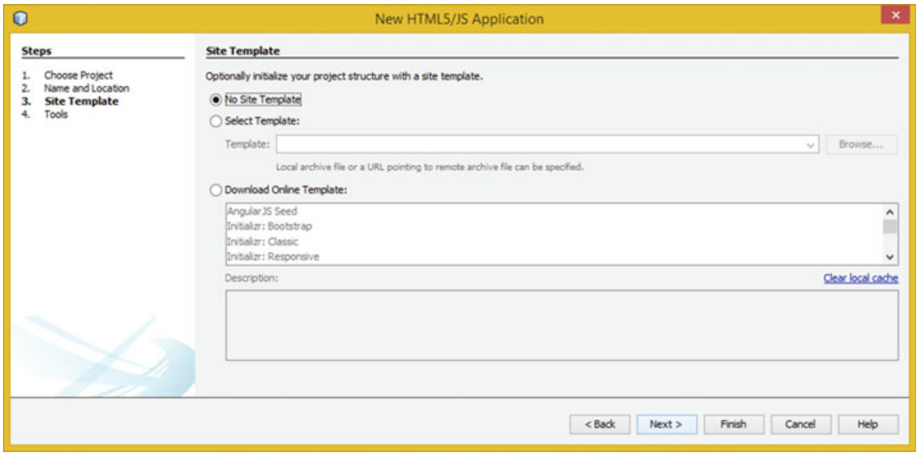
Steps
1. Choose Project
2. **Name and Location**
3. Site Template
4. Tools

Name and Location

Project Name: HTML5-JSON
Project Location: C:\Users\Default.Default-PC\Documents\NetBeansProjects
Project Folder: C:\Users\Default.Default-PC\Documents\NetBeansProjects\HTML5-JSON

Figure 1-4. *Name the project HTML5-JSON, and click the Next button*

In the Site Template dialog, shown in Figure 1-5, select the first radio button, labeled No Site Template. I recommend that you shy away from using site templates—they make your site look like everyone else's, when your objective may be to have a uniquely branded web site design (look and feel).


New HTML5/JS Application

Steps
1. Choose Project
2. Name and Location
3. **Site Template**
4. Tools

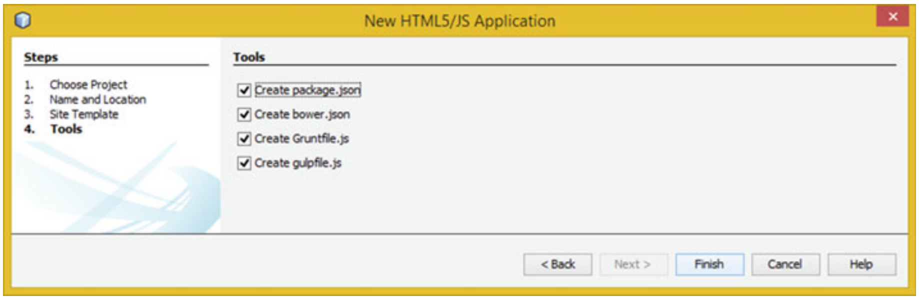
Site Template

Optionally initialize your project structure with a site template.

● No Site Template
○ Select Template:
Template:
Local archive file or a URL pointing to remote archive file can be specified.
○ Download Online Template:
AngularJS Seed
Initializr: Bootstrap
Initializr: Classic
Initializr: Responsive
Description:

Figure 1-5. *Select No Site Template, and click Next*

Click the Next button to advance to the fourth dialog, which allows you to select tools you wish to use for your JSON and JavaScript development. Select the JSON and JavaScript tools that NetBeans 8.1 suggests, as shown in Figure 1-6. I will be sure to cover these so you know what they do for your HTML5, JavaScript, and JSON application project-development workflow, although I prefer to code JS from scratch.



☑ Create package.json
☑ Create bower.json
☑ Create Gruntfile.js
☑ Create gulpfile.js

Figure 1-6. *The Tools dialog in the New Project series*

`package.json` is for the Node.JS Package Manager (NPM); this file holds various metadata relevant to your project. You can use the file to give information to NPM that allows it to identify the project as well handle project dependencies. `project.json` can contain metadata such as the project description, the project's versioning, licensing information, configuration data, and so on, all of which is important to NPM and its end users.

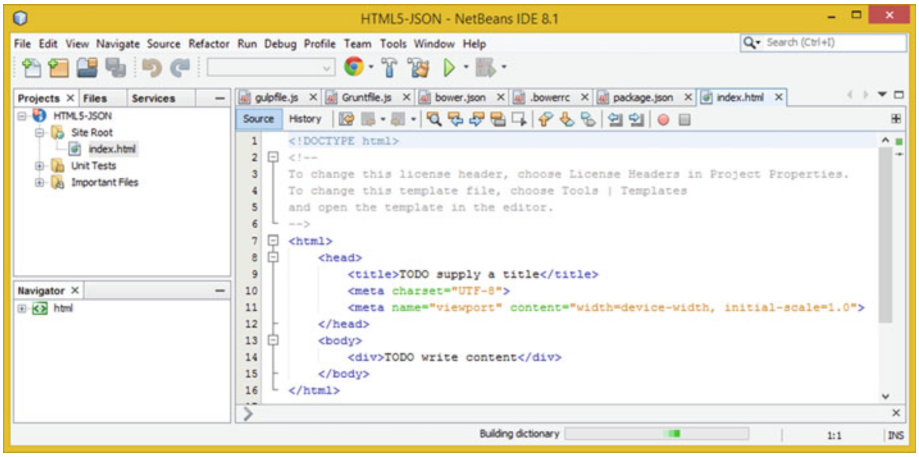
Here's a sample `project.json` file with lots of goodies:

```
{
  "name"         : "jsonproject",
  "description"  : "Sample JSON Project NPM description format using JSON",
  "homepage"     : "example://cloudy.github.com/fakejsonprojectexample/wj/",
  "keywords"     : ["util", "functional", "server", "client", "browser"],
  "author"       : "Apress Author Wallace Jackson <wallacejack@apress.com>",
  "contributors" : [],
  "dependencies" : [],
  "repository"   : {"type": "git", "url": "git://github.com/cloudy/wj.git"},
  "main"         : "jsonproject.js",
  "version"      : "1.1.6"
}
```

This `package.json` file would normally be located in the root directory for your NetBeans 8 JSON project, as highlighted in Figure 1-1. Bower is another package-management system you can use to define project packages, much like the NPM `project.json` file you just learned about.

`Gruntfile.js` is for Grunt, which also works with NPM and is a task-running utility. Grunt and its plug-ins are installed and managed via NPM. A Grunt 0.4.x installation requires stable Node.js versions greater than 0.8. Odd version numbering for Node.js is considered to be unstable. `gulpfile.js` is used for the Gulp streaming build system, which is a Grunt alternate. (I selected these to show you the tool options.)

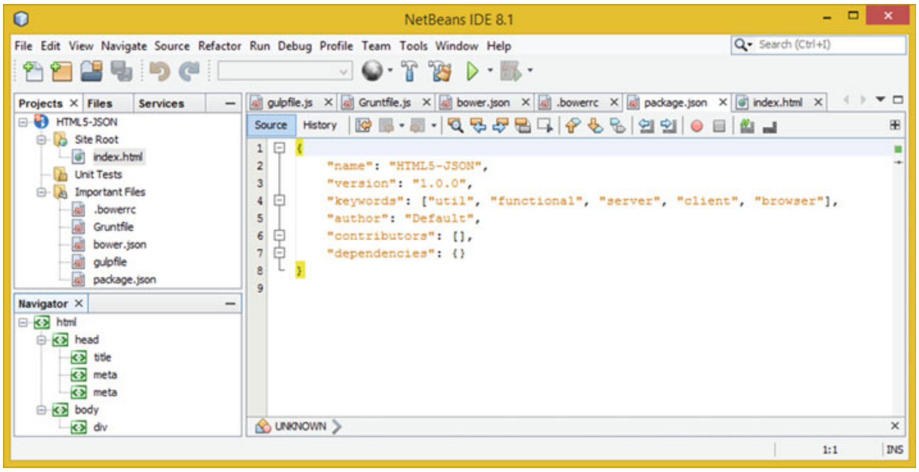
Click the Finish button in the last New Project dialog, and NetBeans 8.1 creates the entire project as shown in Figure 1-7, with folders, subfolders, and files. Now this JSON project is ready for you to populate with your own code and new media assets that the code references.


HTML5-JSON - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
HTML5-JSON
Site Root
index.html
Unit Tests
Important Files
Navigator
html
gulpfile.js Gruntfile.js bower.json .bowerrc package.json index.html
Source History
Building dictionary
1:1 INS

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

Figure 1-7. *Your bootstrap JSON project, created by NetBeans 8.1*

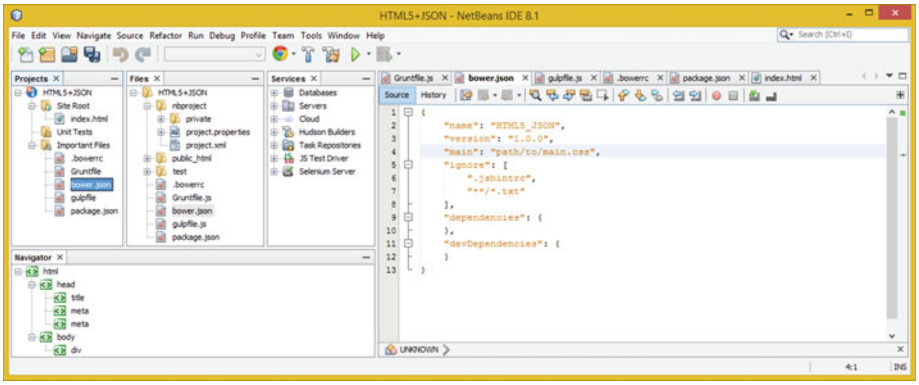
As you can see, NetBeans 8.1 created the project and opened the `index.html` file and the Bower, Grunt, Gulp, and NPM tool-related files in the IDE. Figure 1-8 shows `package.json` with `name`, `version`, `keywords`, `author`, `contributors`, and `dependencies` entries, which you can populate if you're going to use Node.js.



```
{
    "name": "HTML5-JSON",
    "version": "1.0.0",
    "keywords": ["util", "functional", "server", "client", "browser"],
    "author": "Default",
    "contributors": [],
    "dependencies": {}
}
```

Figure 1-8. *The* `package.json` *file defines package-level options*

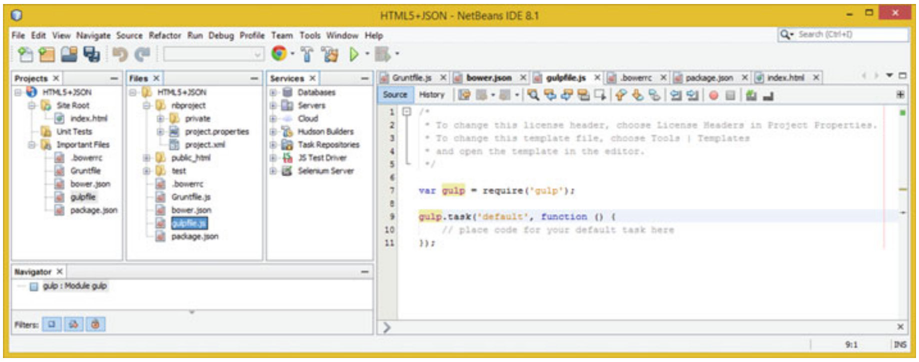
Figure 1-9 shows the `bower.json` file, featuring the `name`, `version`, `main`, `ignore`, `dependencies`, and `devDependencies` data entries. You can customize these if you are using Bower. It is important to note that you are not required to use Node, Bower, Grunt, or Gulp; I'm just showing you the NetBeans tools.



```
{
    "name": "HTML5_JSON",
    "version": "1.0.0",
    "main": "path/to/main.css",
    "ignore": [
        ".jshintrc",
        "**/*.txt"
    ],
    "dependencies": {
    },
    "devDependencies": {
    }
}
```

Figure 1-9. *The* `bower.json` *file defines a project-level CSS path*

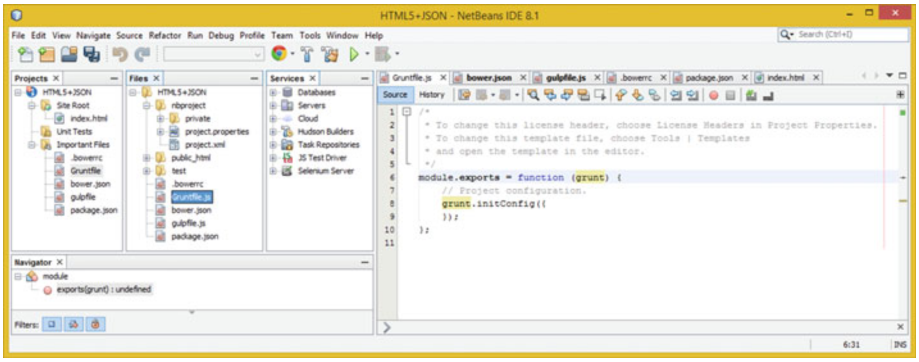
Figure 1-10 shows the `gulpfile.js` file, which defines a variable named `gulp` and then designates it as requiring the Gulp engine. It then calls the `.task()` method from this `gulp` object, defining a `'default'` string and an empty `function()` that you can fill with your own functional program logic.



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

var gulp = require('gulp');

gulp.task('default', function () {
    // place code for your default task here
});
```

Figure 1-10. *The* `gulpfile.js` *file defines project Gulp tasks*

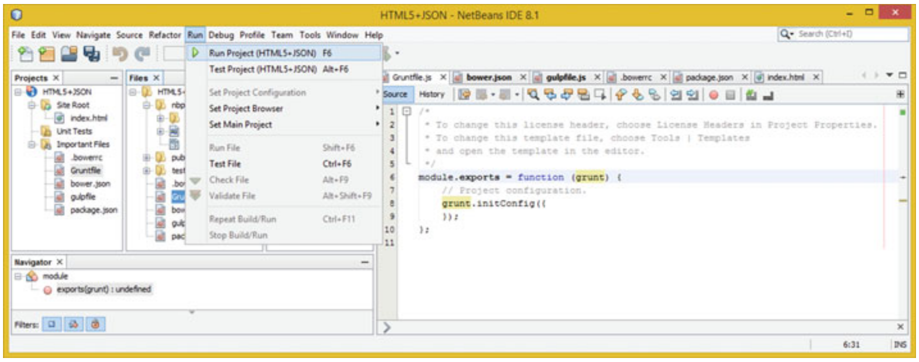
Figure 1-11 shows a `Gruntfile.js` file that instantiates an object named `grunt` in the `function()` parameter area, calls the `.initConfig()` method from the `grunt` object, and includes an empty `{}` code structure in an initialization configuration method parameter area body, which you can code.



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
module.exports = function (grunt) {
    // Project configuration.
    grunt.initConfig({
    });
};
```

Figure 1-11. *The `Gruntfile.js` file defines project Grunt tasks*

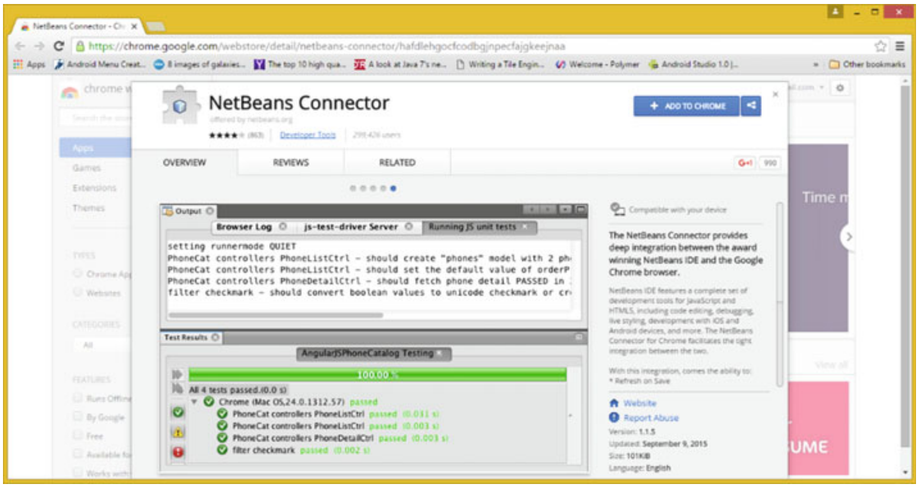
Now that you've seen how to create an empty JSON, JS, and HTML5 project that supports Node.js, Grunt, Gulp, NPM, and Bower, let's take a look at what happens to the NetBeans 8 IDE when you choose the Run ➤ Run Project menu sequence. This is shown at the top of Figure 1-12.



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
module.exports = function (grunt) {
    // Project configuration.
    grunt.initConfig({
    });
};
```

Figure 1-12. *Use the NetBeans Run ➤ Run Project menu sequence*

The first time you run any NetBeans 8 HTML5-related project, you need to establish a bridge to your Chrome browser using the NetBeans Connector plug-in, which you can get at the Google Chrome Web Store for free; see Figure 1-13.


https://chrome.google.com/webstore/detail/netbeans-connector/hafdlehgocfcodbgjnpecfajgkeejnaa
NetBeans Connector
offered by netbeans.org
Developer Tools
+ ADD TO CHROME
OVERVIEW
REVIEWS
RELATED

```
Output
Browser Log    js-test-driver Server    Running JS unit tests
setting runnermode QUIET
PhoneCat controllers PhoneListCtrl - should create "phones" model with 2 ph
PhoneCat controllers PhoneListCtrl - should set the default value of orderP
PhoneCat controllers PhoneDetailCtrl - should fetch phone detail PASSED in
filter checkmark - should convert boolean values to unicode checkmark or cr

Test Results
AngularJSPhoneCatalog Testing
100.00 %
All 4 tests passed.(0.0 s)
  Chrome (Mac OS,24.0.1312.57) passed
    PhoneCat controllers PhoneListCtrl passed (0.031 s)
    PhoneCat controllers PhoneListCtrl passed (0.003 s)
    PhoneCat controllers PhoneDetailCtrl passed (0.003 s)
    filter checkmark passed (0.002 s)
```

Compatible with your device

The NetBeans Connector provides deep integration between the award winning NetBeans IDE and the Google Chrome browser.

NetBeans IDE features a complete set of development tools for JavaScript and HTML5, including code editing, debugging, live styling, development with iOS and Android devices, and more. The NetBeans Connector for Chrome facilitates the tight integration between the two.

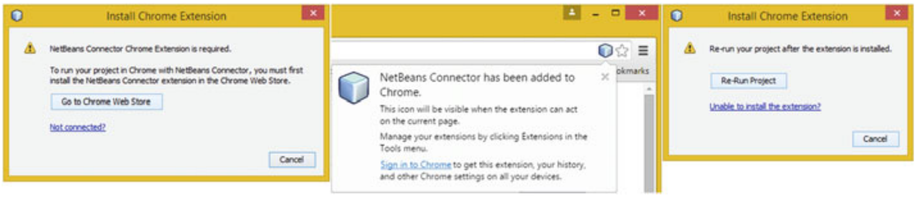
With this integration, comes the ability to:
* Refresh on Save

Website
Report Abuse
Version: 1.1.5
Updated: September 9, 2015
Size: 101KiB
Language: English

Figure 1-13. *Add your NetBeans Connector plug-In to Chrome*

Instead of trying to enter a cryptic URL like the one shown in Figure 1-13 (which, because it is algorithmically generated, could change at any time), you should let the NetBeans 8 IDE do the shopping for you, as shown at far left in Figure 1-14. Simply click the Go To Chrome Web Store button, and NetBeans will automatically open the Google Chrome Web Store page in Figure 1-13.

Once you click the blue Add To Chrome button, shown at upper right in Figure 1-13, you see the pop-up dialog in the middle of Figure 1-14, advising you of the install completion. When you return to NetBeans, you see the dialog shown at right in Figure 1-14; click the Re-Run Project button to run your (empty) bootstrap code. Running empty projects doesn't produce an application, but NetBeans opens a few more (CSS Styles, Output, Browser DOM) panes. You have now previewed the JSON-capable IDE from Oracle!



NetBeans Connector Chrome Extension is required.

To run your project in Chrome with NetBeans Connector, you must first install the NetBeans Connector extension in the Chrome Web Store.

NetBeans Connector has been added to Chrome.

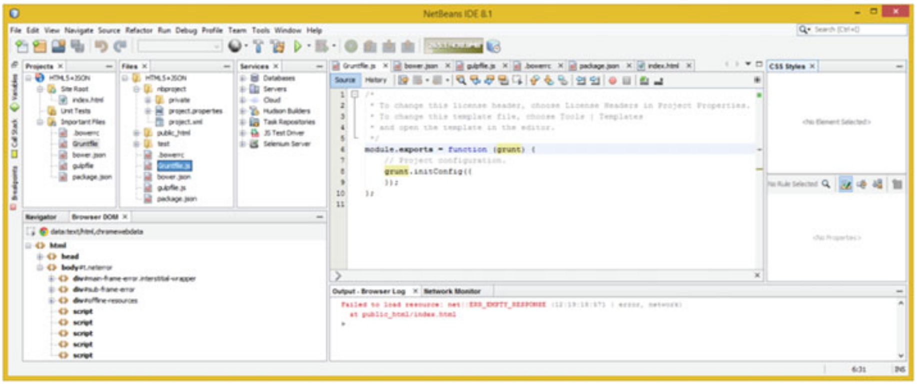
This icon will be visible when the extension can act on the current page.

Manage your extensions by clicking Extensions in the Tools menu.

Sign in to Chrome to get this extension, your history, and other Chrome settings on all your devices.

Re-run your project after the extension is installed.

Figure 1-14. *Install Chrome Extension dialog and work process*



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
module.exports = function (grunt) {
    // Project configuration.
    grunt.initConfig({
    });
};
```

```
Failed to load resource: net::ERR_EMPTY_RESPONSE (12:13:18:57) | error, network)
  at public_html/index.html
```

Figure 1-15. *Several new panes have been opened by NetBeans 8.1*

Summary

In this first chapter, you looked at the NetBeans 8.1 JSON-capable IDE. You saw NetBeans' powerful, intuitive features and learned how to have NetBeans create JSON projects. I wanted to make sure you understood your software tools before proceeding!

In the next chapter, you start learning about JSON: what it is, what it's used for, and similar foundational information you need throughout the rest of this reference book.

CHAPTER 2

An Introduction to JSON: Concepts and Terminology

Now that you have had an overview of the integrated development environment (IDE) you use to work with JSON and compatible OOP languages—the most common of which is Oracle NetBeans 8.1 for Java, JavaFX, and HTML5 development—this chapter takes a high-level look at what defines JSON, including the concepts behind it, advantages of using JSON, terms used to describe it, and the various rules that dictate its data-object definition design approach. JSON is well on its way to replacing XML as a definition format, at least where object-oriented data is concerned. Chapter 3 looks at object-oriented concepts and design; here you focus on learning other JSON-related information, so later you can focus on the definition format and what it supports and allows you to do with objects. After these core chapters, the book examines real-world applications for JSON as well.

What Is JSON?

JSON stands for *JavaScript Object Notation*. It is based on the *ECMA-262 3rd Edition JavaScript Standard* established in December 1999. This subset of the JavaScript programming language, often called *ECMAScript*, is what I use for JavaScript programming. This is so my JavaScript code is sure to run across all WebKit-based HTML5 browsers and HTML5 operating systems. This is why I use ECMAScript in my HTML5 with CSS3 and JavaScript projects, rather than one of the proprietary JS package solutions (Node.JS, for instance), which might not have native support on any given OS, platform, or device. To have native support, the Node.JS library would have to be part of an OS, browser, or platform API, as ECMAScript-262 is. More about this later on in the chapter. Let's look at JSON's advantages.

JSON Is Lightweight: Simpler Is Usually Superior

The JSON object structure definition language originally was designed and developed for use as a lightweight, JavaScript object-oriented data-interchange programming structure. This JSON format was specifically designed to be as simple as possible so that programmers can read JSON data structures effortlessly as well as encode JSON data structures optimally, concisely, quickly, and efficiently.

W. Jackson, *JSON Quick Syntax Reference*, DOI 10.1007/978-1-4842-1863-1_2

Obviously, this also makes JSON easy for programming algorithms to parse, process, and generate; but then again, what streamlined algorithm can't a 2 GHz, multicore processor handle these days? Still, the less processing power is used to parse and process JSON data structures, the more is left over for your application to use.

No matter how powerful a server, network, or client-side device may be, data-footprint optimization is always a great thing. You want scalability to be maximized in case your application takes off and millions of users flock to it as it becomes the "in thing." This is exactly what JSON has been optimized for in its design, structure, language, and implementation. XML, for instance, uses far more characters to design data structures than a JSON structure would.

JSON Is Text-Based: 100% Language Independent

JSON uses a text-based format that's completely language independent. The specification uses conventions familiar to programmers of OOP languages, including C, C+, C++, Objective-C, C#, Java, JavaFX, JavaScript, Ruby, Visual Basic .NET, Object COBOL, Object Pascal, Scala, Smalltalk, ADA, Perl, PHP5, Python, Prolog, and dozens of others. (In case you are wondering, Simula was the first OOP language; it was created in Norway by Ole-Johan Dahl and Kristen Nygaard back in the 1960s. The creator of C++, Bjarne Stroustrup, was influenced by the object-based approach of Simula. The Java programming language was also influenced by this object-oriented approach.)

This object-based, text-scripted approach makes JSON the perfect data-interchange definition language for use in today's popular OOP languages, operating systems, devices, and platforms. Let's take a closer look at the types of data structures that can be used in a JSON data-definition language construct. After that, you explore the close relationship JSON has with ECMAScript-262 JavaScript as its primary parsing OOP language.

JSON Structures: Data Objects and Data Arrays

Only two primary types of data structures can be used in JSON, and they're covered in this section. This doesn't mean JSON data constructs are not powerful; these two types of data structures can be used in conjunction to build complex data representations, as you see throughout this book.

It is important to note that the structures used in JSON are what are called *universal* data structures, because they are used in virtually all current programming languages and platforms in one format or another. It is therefore logical that JSON is based on these two types of data structures, in order for each language that supports using JSON to be able to parse, or deconstruct, JSON representations into objects in memory.

Data Objects: Collections of Name:Value Pairs for Data Structures

A collection of name:value pairings is known across programming languages by a many terms. The most common is *object*, due to C++ and Java, but terms such as *record*, from database languages; *struct*, for structure (which I prefer); and *hash table* are also common, as you know. Other terms include *associative array*, *data dictionary*, and *keyed list*.

Thanks to the *curly brace* `{ ... }` encapsulation construct and the ability to nest these constructs, objects can become complex and powerful for defining precise data-structure organization and object-design hierarchies. You look at this in Chapter 3, which reviews OOP concepts, structure, and terminology using the popular Java and JavaScript languages.

Data Arrays: An Ordered List Defining Sequential Data Values

In addition to data objects, you can also define data arrays, which are ordered lists of data values that can be accessed by positional pointers or references in that sequence. This is a database or datastore in its simplest format. In conjunction with the allowed object structure, your database record structure can be simulated handily using JSON.

An ordered list of sequential data values is also recognized across programming languages by a number of terms. This construct is most commonly referred to as a *data array*. Other terms, such as *vector*, from 2D and 3D mathematics; *sequence*, also from mathematics; and *list* are also common. Other terms may include *data collection*, *datastore*, and *ordered data*.

JSON Is JavaScript Compatible: Easy Integration

Because JavaScript is half of what JSON stands for, it is logical that JSON works seamlessly with JavaScript (JS). JavaScript is a general-purpose programming language originally introduced with the Netscape Navigator browser, as its document scripting language. JavaScript works in conjunction with HTML5 and CSS3. One of the popular myths regarding JavaScript is that it is somehow related to Oracle's Java, but it is in fact not related to Java other than being another popular OOP language. JavaScript has been standardized in ECMAScript using the *ECMAScript-262 Language Specification, 3rd Edition*.

JavaScript is similar to the Scheme language and uses a syntax much like the original C programming language. It uses *soft objects*, where new object members can be added to the soft object via simple assignment. A *hard object*, on the other hand, would require a new class to instantiate it. You look at this difference in more detail in Chapter 3.

JSON is a subset of JavaScript's object literal notation and is therefore seamlessly compatible using that OOP language. Because JSON is a subset of JavaScript, it can be used in that language with no alteration of format, structure, or code design. This means less work for the JavaScript (HTML5) developer.

JSON object members can be retrieved in JavaScript using dot notation (`object.member`) or subscript operators, so you can reference members in the JSON defined object right in the JavaScript code. You can also convert JSON object definitions to JavaScript objects using the JS `eval()` function.

Converting a JSON Object Definition to an Object: Using eval()

To convert JSON text to an object in JavaScript, you can use a JavaScript `eval()` function. The `eval()` function invokes the JavaScript compiler. Because JSON is a subset of JavaScript, your compiler will correctly parse the JSON object definition text, produce that object's structure, and load it in system memory. The JSON object definition must be wrapped in parentheses to avoid triggering any code ambiguities when you are compiling JavaScript syntax. The JavaScript syntax looks like this:

```
var yourJSONobject = eval(' (' + yourJSONobjectDefinition + ') ');
```

The algorithm used by JavaScript's `eval()` function is well optimized, so it executes very rapidly. The `eval()` function can also compile and execute other JavaScript programs, so security issues may arise. Therefore, if you're going to use an `eval()` function, make sure your source is trusted and the programmers who wrote the source were competent.

If your web application is using an `XMLHttpRequest` call, communication is allowed only via the same server providing the page. This determines that it's trusted. However, that source JSON object definition may still be incompetent. If your server is not robust in its JSON encoding, or if it does not meticulously validate all of its data input, then it could deliver invalid JSON object definition text that may carry dangerous script. An `eval()` function might execute this scripting and unleash malware.

Parsing JSON Object Definitions into Objects: Using JSON.parse()

Because of the security issue with the `eval()` function, it is safer to use the *JSON parser*. The JSON parser recognizes only JSON object definitions, rejecting other scripts that do not define properly formed JavaScript Object Notation.

In HTML5 browsers that provide native JSON support, the JSON parser approach should be significantly faster than using the `eval()` function. Native JSON support is included in the sixth ECMAScript standard, also called the ECMAScript 2015 Language Specification. A PDF of the specification can be found at `http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf`.

Using `JSON.parse()` on the JSON definition looks like this:

```
var yourJSONobject = JSON.parse(yourJSONobjectDefinition, reviver);
```

The (optional) `reviver` parameter can contain a *function*, which is called for every key:value pair at every level of the final object. Each value is replaced by the result of this `reviver` function. The `reviver` function may be used to reform generic objects to pseudoclass instances or, for instance, to transform any date strings into actual `Date` objects with a `Date` function.

There is also a `JSON.stringify()` function, if you want to go in the other direction and turn your JavaScript objects into JSON definitions. This function, like `JSON.parse()`, is included in the ECMAScript 2015 Language Specification. Let's take a look at this JavaScript function next.

Stringifying JSON Objects into Definitions: Using JSON.stringify()

The `JSON.stringify()` function lets you process in the opposite direction and convert JavaScript object structures into JSON definitions. Because JSON does not support cyclical data structures, be careful not to give any cyclical data structures to a JSON stringifier. Using `JSON.stringify()` on your JSON object looks like this:

```
var yourJSONobjectDefinition = JSON.stringify(yourJSONobject, replacer);
```

If the `stringify()` method encounters any JavaScript object that contains a `toJSON()` method, it calls that method. This stringifies the value that is returned, whch should allow the object to determine its own JSON representation.

The `JSON.stringify()` method can take an (optional) array of strings. These strings are used to select the properties that are included in the JSON object definition's text.

The `JSON.stringify()` method can also take the (optional) `replacer` function. This is called only after the `toJSON()` method, if there is one present, on each of the values in your object data structure. This is passed each key:value pair as parameters and is bound to the object holding the key. The returned value is stringified.

Values that don't have any representation in JSON, such as functions and undefined, are excluded. Nonfinite numbers are replaced with a *null value*. To substitute other values, you should use the `replacer` function.

A customized `replacer(key, value)` function is coded something like this in JavaScript:

```
function replacer(key, value) {
    if (typeof value === 'number' && !isFinite(value)) {
        return String(value);
    }
    return value;
}
```

Giving your corresponding `reviver` to `JSON.parse()` can undo the `replacer` function, so these are analogs (opposites) of each other.

JSON.parse() and JSON.stringify(): Open Source Code Available

The open source code for a JSON parser or a JSON stringifier is available at creator Douglas Crockford's GitHub account `https://github.com/douglascrockford/JSON-js`. There are `cycle.js` utilities for JSLint, JSON_Parse, and JSON2.js functions available. When minified, these are less than 2.5 KB.

JSON became a built-in feature of JavaScript when the *ECMAScript-262 5th Edition* was adopted in 2009.Most of the files on this web site are for applications that are expected to run in deprecated (obsolete) web browsers. For most purposes, you should use the `JSON2.js` version or functions that are native to the latest ECMAScript version.

JSON Open Licensing: Free for Commercial Use

JSON has an exceptionally open license, which I copied from the popular www.json.org website and included here for review. As you see, you must always include this licensing text.

Summary

This chapter defined what JSON is, its advantages and attributes, concepts and terminology, and so forth, to give you a broad foundational understanding of JSON before the rest of the book starts breaking it down into its logical components. In the Chapter 3, you learn about OOP languages with an object-orientation primer (review), before the book gets into defining JSON objects and taking a look at how to create a JSON object definition.

CHAPTER 3

The JSON Schema: JSON Structure Validation

Now that you have had an overview of the JSON data-definition language, this chapter looks at the schema that defines JSON, including the concepts behind the JSON Schema, advantages of using it, terms used to describe it, and the various rules that dictate its definition and design approach. The chapter also looks at the JSON Hyper-Schema definition and design approach.

Just as the JSON language can be found at: `www.json.org`, the JSON Schema is at `www.json-schema.org`. The web site has four primary sections: About, Docs, Examples, and Software. This chapter outlines the core JSON Schema information found on this web site so you understand the full scope of what can be done using JSON on both the client side and the server side of your asynchronous processing connection. JSON allows you to implement both client-side and server-side processing and components for your application's JSON data objects. After the book covers the JSON Schema Core, Chapters 4–8 get into how to define JSON definition structures.

JSON Schema: Concepts and Definitions

The JSON Schema lets you describe the JSON object definition data format. It is based on the *JSON Schema Definition, Version 4,* which is located at `www.json-schema.org` and referenced in of the JSON Schema definition file, as you see later in this chapter. The JSON Schema specification is split into three primary working components: *JSON Schema Core*, *JSON Schema Validation*, and *JSON Hyper-Schema*.

This chapter covers these and how they work together. Let's get started by looking at the advantages of JSON Schema and JSON Hyper-Schema. You see examples in this chapter and those that follow of how curly braces ({}), colons, and commas are used to define JSON's syntax, so you know how to define objects and their (nested) sub-objects. For now, just follow along; the next chapter gets into this in more detail when you see how to define and encode JSON data objects using these curly-braces structures.

JSON Schema Advantage: Clear JSON Description

The JSON Schema definition language was designed and developed for use in describing an existing JSON data format. Like JSON, the JSON Schema uses a lightweight, JavaScript object-oriented data-interchange programming structure. In fact, JSON Schema uses the JSON data-structure format, because it was specifically designed to be as simple as possible.

Obviously, this makes JSON Schema very clear and concise, so it is easy for programmers and programming algorithms to parse, process, and validate. It provides both human-readable and machine-readable documentation for JSON definitions.

JSON Schema definitions provide you with complete JSON structure-validation roadmap capability, allowing you to define your client-server JSON relationship so that JSON data-object definitions can be synchronized on both ends of the client-server transaction (work process). This lets you implement things such as automated testing and validate client-submitted JSON data objects, because JSON Schema can make sure data types and ranges match correctly with what your server-side programming logic is able to (or looking to) process.

Next, let's look at the JSON Hyper-Schema. As you may have guessed, it handles more advanced JSON constructs and multimedia assets.

JSON Hyper-Schema Advantage: Links and Forms

The JSON Hyper-Schema definition language was designed and developed for use in describing more advanced capabilities for existing JSON data formats such as links, URLs, URIs, and forms, and multimedia assets such as digital images, digital video, and digital audio. Like JSON, JSON Hyper-Schema uses a lightweight, JavaScript object-oriented data-interchange programming structure. JSON Hyper-Schema also uses the JSON data-definition structure format, because it was specifically designed to be as simple as possible.

JSON Hyper-Schema adds the capability to define *links* (also known as *hyper-links*) in JSON objects, using the URI Template data format. More information regarding this data format can be found at `http://tools.ietf.org/html/rfc6570`. Obviously, links in JSON Hyper-Schema let you craft a more powerful and flexible user experience.

JSON Hyper-Schema also makes it possible to define *forms*: you can define a form's data structure using JSON Schema for the form's JSON data-object definition format. Forms in JSON Hyper-Schema allow a more useful, powerful, flexible user experience: users can fill out and submit forms that are inherently validated by JSON Schema.

Now, let's look at JSON Schema Core. It provides the foundational definitions and terminology for the JSON Schema and JSON Hyper-Schema components.

JSON Schema Core: Language Definition

JSON Schema Core provides a *contract* (a rigid syntax-processing specification for programmers to follow) for what JSON data-definition format is required for any given application, and how to interact with that data-object structure. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control for your JSON data. The Core specification defines JSON Schema Core terminology and mechanisms.

Related JSON specifications are built on this specification and define different applications. Current examples are JSON Schema Validation and JSON Hyper-Schema.

JSON uses conventions and terminology that are described in RFC2119 and RFC4627. JSON Schema Core uses RFC2119 to define the following keywords: `MUST`, `MUST NOT`, `REQUIRED`, `SHALL`, `SHALL NOT`, `SHOULD`, `SHOULD NOT`, `RECOMMENDED`, `MAY`, and `OPTIONAL`. It uses RFC 4627 to define the following terms: *JSON*, *JSON text*, *JSON value*, *member*, *element*, *object*, *array*, *number*, *string*, *boolean*, *true*, *false*, and *null*.

Section 3 of JSON Scheme Core defines the core terminology used. Section 3.1 defines the terms *property* and *item*. When referring to a JSON object, as defined by RFC4627, the terms *member* and *property* can be used interchangeably. When you refer to JSON arrays, the terms *element* and *item*, again as defined in RFC4627, can be used interchangeably.

Section 3.2 defines a JSON Schema document and its keywords. A JSON Schema document is a JSON document, and that document *must* be a JSON object. Object members or properties defined by this JSON Schema (this JSON Schema Core specification or related specifications) are called *keywords* or *schema keywords*. A JSON Schema *may* contain properties that are not schema keywords as well, if you desire for your JSON applications.

Section 3.3 defines an empty schema. An *empty schema* is a JSON Schema using no properties, or with properties that are not schema keywords.

Section 3.4 defines a *root schema* and a *subschema*. Here is an example of a JSON root schema that has no subschemas:

```
{ "title" : "root" }
```

JSON Schemas can also be nested, in which case they are subschemas. Here is an example, where `nested` and `alsoNested` are subschemas, and `root` is a root schema:

```
{ "title" : "root",
     "otherSchema": {
     "title": "nested",
          "anotherSchema": {
          "title": "alsoNested"
                           }
                    }
}
```

In Section 3.5, the JSON Schema defines seven distinct primitive types for JSON values: `array`, which is a JSON array; `boolean`, which is a JSON boolean; `integer`, which is a JSON number without a fractional or exponent portion; `number`, which is any JSON number (`number` includes `integer`); `null`, which is the JSON null value; `object`, which is a JSON object; and `string`, which is a JSON string. The next several chapters cover these in detail; they are core to the JSON definition.

Section 3.6 defines JSON *value equality*. Two JSON values are said to be equal if and only if both are nulls or both are booleans, and they are set to the same value; or if both are strings and contain the same value; or if both are numbers and have the same mathematical value. If both are arrays, to be considered equal, they must have the same number of items, and items at the same index must also be equal, according to the previous definitions. If both are objects, they must have the same set of property names and values for each property.

Section 3.7 defines an *instance* as being any JSON value. An instance may be described by one or more schemas. An instance may also be referred to as a JSON instance or JSON data.

Section 4 of the JSON Schema Core contains an overview of the JSON Schema definition. This document proposes a new media type, also called the MIME type, denoted as follows:

```
"application/schema+json"
```

This identifies the JSON Schema for describing JSON data. A JSON Schema is itself defined using JSON. The JSON Schema Core and related specifications define keywords allowing these JSON Schemas to describe JSON data in terms of allowable values and textual descriptions, and allowing the interpretation of relations with other types of data resources. The following Section 4 subsections contain a summary of features that are defined by related specifications.

Section 4.1 defines *JSON validation*. JSON Schema allows applications to *validate* instances, either non-interactively or interactively. For instance, an application may collect JSON data and check that this data matches a given set of constraints; another application may use a JSON Schema to build an interactive interface, in order to collect user input according to constraints described by the JSON Schema.

Section 4.2 defines *hypermedia* and *linking*. This JSON Schema provides a method for extracting link relations to other resources from a JSON instance, as well as describing the interpretations of instances as multimedia data. This allows JSON data to be interpreted as rich hypermedia documents, placed in the context of the larger collection of related JSON data resources.

Section 5 of the JSON Schema Core defines all general considerations. Section 5.1 defines the JSON Schema's applicability to all JSON data values. It is acknowledged that an instance may be any valid JSON value, as defined by RFC4627. It is interesting to note that the JSON Schema does not mandate that any instance be of any particular type; a JSON Schema can describe any JSON value, including the null value.

Section 5.2 defines the JSON Schema's programming language independence. The JSON Schema inherently needs to be programming-language agnostic, as covered in Chapter 2. The only limitations to this programming-language independence are those expressed in RFC4627 or those of a host programming language.

Section 5.3 defines the JSON Schema Core as it relates to HTTP. This specification acknowledges the role of HTTP in document RFC2616 as being the dominant protocol in use on the Internet, along with the large number of official specifications related to HTTP. This specification uses a subset of the HTTP specifications to recommend a set of mechanisms usable by this protocol to associate JSON instances to one or more schemas.

Section 5.4 defines the JSON Schema Core's relation to other protocols. A JSON Schema doesn't define any semantics for JSON client-server interfaces for any other protocols other than HTTP. These semantics are application dependent or subject to agreement between parties involved in use of the JSON Schema for their own private development requirements.

Section 5.5 defines mathematical integers. It is acknowledged by this specification that some programming languages, and their associated numeric parsers, use different internal representations for floating-point numbers and integers, whereas other languages do not. For this reason, for interoperability purposes, JSON data values used in the context of a JSON Schema—whether that JSON data is a JSON Schema or a JSON instance—should ensure that mathematical integers are represented using the `integer` type defined in this specification.

Section 5.6 defines extending the JSON Schema. Implementations *may* choose to define additional proprietary keywords for the JSON Schema. In the absence of an explicit agreement, schema authors *must not* expect that these additional keywords are supported by peer implementations. Implementations *should* ignore keywords that they do not support and that are not a part of these public JSON Schema definitions.

Section 5.7 defines JSON security considerations. Both schemas and instances are JSON values. As such, all security considerations defined in document RFC4627 apply.

Section 6 defines the JSON Schema `$schema` keyword, and Section 6.1 defines the purpose of this `$schema`. The $schema keyword can be used as the JSON Schema *version identifier* as well as the *location of the resource* that in and of itself is the JSON Schema definition that describes any JSON schema written for this particular version.

The `$schema` keyword *must* be located at the root of the JSON Schema. The value of the keyword *must* be a URI as defined in document RFC3986 and must also be a valid JSON Reference, and the URI *must* be both absolute and normalized. The resource located at this URI *must* successfully describe itself. It's *recommended* that JSON schema authors include this keyword at the beginning of their schema definition, like this:

```
{ "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "A JSON Schema",
  "description": "example",
  "type": "object",
  "properties":
  { remainder of JSON Schema definition properties are nested in this area }
}
```

The following JSON Schema `$schema` HTTP values are currently predefined:

```
http://json-schema.org/schema# (Written against the current Schema version)
http://json-schema.org/hyper-schema# (Written against the current version)
http://json-schema.org/draft-04/schema# (Written against Draft 4 version)
http://json-schema.org/draft-04/hyper-schema# (Written against Draft 4 ver.)
```

Section 6.2 defines JSON Schema customization. When extending the JSON Schema with custom keywords, schema authors *should* define a custom URI for `$schema`. This custom URI *must not* be one of the previous predefined (publicly used) values.

Section 7 defines URI resolution scopes and dereferencing mechanisms for JSON. Section 7.1 contains the definition of the JSON Schema, and it uses JSON Reference as the mechanism for the schema addressing. It extends the JSON Schema specification in two ways: JSON Schema offers facilities to alter the base URI against which a reference must resolve by the means of the `id` keyword; and it defines a specific dereferencing mechanism, extending JSON Reference to accept arbitrary fragment parts. Altering this URI in the schema is called *defining a new resolution scope*. The initial resolution scope of a schema is the URI of the schema itself, if any, or the empty URI if the schema was not loaded by using the URI.

Section 7.2 defines URI resolution-scope alteration by using an `id` keyword. Section 7.2.1 defines valid values for this `id` keyword. The value for this keyword *must* be a string and *must* be a valid URI. This URI *must* be normalized and *should not* be an empty fragment (#) or the empty URI.

Section 7.2.2 defines the usage of the `id` keyword. The `id` keyword is used to alter the URI resolution scope. When `id` is encountered, an implementation *must* resolve it against the *most immediate parent scope*. The resolved URI is the new resolution scope for this subschema and all of its children, until another `id` is encountered. When using `id` to alter resolution scopes, schema authors *should* ensure that resolution scopes are unique within the schema. Here is an example from the `www.json-schema.org` web site:

```
{
    "id": "http://x.y.z/rootschema.json#",
    "schema1": {
        "id": "#foo"
    },
    "schema2": {
        "id": "otherschema.json",
        "nested": {
            "id": "#bar"
        },
        "alsonested": {
            "id": "t/inner.json#a"
        }
    },
    "schema3": {
        "id": "some://where.else/completely#"
    }
}
```

Subschemas at the following URI-encoded JSON Pointers starting from the root schema define the following resolution scopes:

```
# (document root)     http://x.y.z/rootschema.json#
#/schema1             http://x.y.z/rootschema.json#foo
#/schema2             http://x.y.z/otherschema.json#
#/schema2/nested      http://x.y.z/otherschema.json#bar
#/schema2/alsonested  http://x.y.z/t/inner.json#a
#/schema3             some://where.else/completely#
```

Section 7.2.3 defines canonical dereferencing and inline dereferencing for JSON Schemas. When resolving a URI against a resolution scope, an implementation may choose two modes of operation: *canonical dereferencing*, where the implementation dereferences all resolved URIs, or *inline dereferencing*, where the implementation chooses to dereference URI values within the schema. Implementations *must* support canonical dereferencing and *may* support inline dereferencing. For example, consider this schema found at `www.json-schema.org`:

```
{
    "id": "http://my.site/myschema#",
    "definitions": {
        "schema1": {
            "id": "schema1",
            "type": "integer"
        },
        "schema2", {
            "type": "array",
            "items": { "$ref": "schema1" }
        }
    }
}
```

When the implementation encounters a `schema1` reference, it resolves the reference against the most immediate parent scope, which references URI `http://my.site/schema1`#. The way to process this URI value differs according to your chosen dereferencing mode. If canonical dereferencing is used, the implementation dereferences this URI and fetches your content at this URI. If inline dereferencing is used, the implementation notices that URI scope `http://my.site/schema1`# is already defined in your schema and uses the appropriate subschema.

Section 7.2.4 defines JSON Schema in-line dereferencing and fragment support. When using inline dereferencing, a resolution scope may lead to a URI that has a non-empty fragment part, which is not a JSON Pointer, as in this example from `www.json-schema.org`:

```
{
    "id": "http://some.site/schema#",
    "not": { "$ref": "#inner" },
    "definitions": {
        "schema1": {
            "id": "#inner",
            "type": "boolean"
        }
    }
}
```

An implementation choosing to support in-line dereferencing *should* be capable of using this kind of referencing. Implementations choosing to use canonical dereferencing, however, are not required to support this.

Section 7.3 defines security considerations for the JSON Schema. Inline dereferencing can produce canonical URIs that differ from the canonical URI for the root schema. Schema authors *should* ensure that implementations that use canonical dereferencing obtain the same content as implementations using inline dereferencing. Extended JSON References that use fragments that are not JSON Pointers are not dereferenceable using implementations choosing not to support inline dereferencing. This kind of referencing is defined for backward compatibility and *should not* be used in new JSON schemas.

Section 8 defines recommended correlation mechanisms for use with the HTTP protocol. It is acknowledged by the specification that the majority of interactive JSON Schema processing is over HTTP. Section 8 therefore provides recommendations for materializing an instance/schema correlation using mechanisms that are currently available for this protocol. An instance is said to be *described* by one or more schemas.

Section 8.1 defines HTTP correlation by means of the `Content-Type` header. It is *recommended* that a MIME type parameter by the name of `profile` be appended to the `Content-Type` header of the instance being processed. If present, the value of this parameter *must* be a valid URI, and this URI *should* resolve to a valid JSON Schema. The MIME type *must* be `application/json` or any other subtype.

An example of a header from the `www.json-schema.org` web site is as follows:

```
Content-Type: application/my-media-type+json;
  profile=http://example.com/my-hyper-schema#
```

Section 8.2 defines the correlation by means of the `Link` header. When using the `Link` header, the relation type used *must* be `describedBy`, as defined by document RFC5988, Section 5.3. The target URI of the `Link` header *must* be a valid JSON Schema. Here is an example of a header from `www.json-schema.org`:

```
Link: <http://example.com/my-hyper-schema#>; rel="describedBy"
```

Section 9 describes IANA considerations. The proposed MIME media type for JSON Schema is defined as follows:

```
type name:    application;
subtype name: schema+json.
```

Before getting into the more advanced JSON Schema (Validate and Hyper-Schema) definitions, I want to make sure you understand how to define JSON data objects using all of their different types, from a syntactic perspective. Chapters 4–8 present some hands-on examples. The book revisits more advanced topics such as validation and hypermedia once you begin looking at real-world examples of how JSON syntax works.

Summary

This chapter defined what the JSON Schema is, its advantages and attributes, concepts and terminology, and so forth, to give you a broad foundational understanding. The next five chapters break down JSON data objects and their definitions into logical types and components.

You learned that you can use JSON Schema(s) to define how your JSON data objects are processed when you are using JSON as an asynchronous data communications protocol between server-side and client-side applications. You looked at the JSON Schema Core in detail, including all nine sections and their subsections, and learned how this provides a foundation for JSON Schema Validate and JSON Hyper-Schema, which define more complex JSON processing.

In Chapter 4, you learn about two popular OOP languages. You then get into the definition of JSON objects and look at how to create JSON object definitions.

CHAPTER 4

Objects and Object-Oriented Programming: OOP Primer

Now that you have had an overview of JSON in Chapter 2 and the JSON Schema Core and its components in Chapter 3, this chapter examines what defines object-oriented programming (OOP), including the concepts behind it, advantages of using OOP, terms used to describe OOP, and the difference between hard objects used in Java and soft objects used in JavaScript. Remember that JSON is used with JavaScript, Java, C++, and a great many other OOP languages, but it is best integrated with JavaScript and also integrates seamlessly with ECMAScript-262.

First you look at OOP concepts via Java, which is used on every OS as well as in the popular Android operating system. Java uses a more complex, hard object paradigm, because it requires classes and constructors to create an object. Then you examine OOP in JavaScript, which uses a soft object paradigm: objects can be created without using special classes and constructor methods. (By the way: *methods* are called *functions* in JavaScript.)

Object-Oriented Programming: Overview

The initial programming languages were *top down* or *linear* and either had line numbers, as in BASIC, or processed lines of code in the order that they appeared. As you learned in Chapter 2, the first OOP was Simula, and it introduced some very advanced concepts that made programming an order of magnitude more flexible, extensible, and powerful, all at the same time. These faculties also made OOP more difficult to learn and comprehend, which is why this book spends this chapter making sure you understand OOP and the two types of objects created in the two most popular programming languages today: Java and JavaScript.

This chapter starts with Java because chances are you use it for application development or mobile development; the chapter finishes with JavaScript and at that point also introduces the JSON object value and how to define a JSON object. JavaScript has fewer complexities than Java, so covering Java first gets those complexities out of the way. Then you can look at how JavaScript differs; it is less rigid regarding rules, which is why it is called a *soft object model* whereas Java is called a *hard object model*. As far as JSON goes, it is as widely used with Java as it is with JavaScript, so all of this chapter

© Wallace Jackson 2016
W. Jackson, *JSON Quick Syntax Reference*, DOI 10.1007/978-1-4842-1863-1_4

is directly relevant to JSON no matter how you slice it. Java has extensive JSON class libraries, just as JavaScript has a plethora of JSON functions. Some of these, such as `JSON.parse()`, you saw in Chapter 2.

Java OOP Concepts: Hard Object Construction

Let's make sure you and I are on the same page by reviewing the core concepts and principles behind the Java programming language. This chapter gives you a primer or comprehensive overview of an entire OOP language. The Java JDK (and JRE) that you install in the appendixes of this book are the foundation for the JSON IDE and Java applications as well as for the NetBeans 8.1 IDE, which you saw in Chapter 1. You also learned the basics of how the IDE you are using to code JSON using Java or JavaScript (HTML5) applications functions as a code editor or application-testing tool.

Many of the core Java OOP constructs and principles covered in this chapter go back quite far in the Java programming language—most of them as far back as Java 1 (known as 1.02). The most widely used version of Java SE currently is Java 6 (1.6); Java EE uses Java 8. Android 4.4 and earlier use Java 6, until the advent of Android 5 and 6, which use Java 7 (1.7). This chapter also covers the features added in Java 8 (1.8), which is the most recent release, as well as the new features planned for Java 9 (1.9), which will be released in the fourth quarter of 2016. All these versions of Java are used on billions of devices, including Java 6, which is used in the 32-bit Android 2.x, 3.x, and 4.x OS, and applications; Java 7, used in the 64-bit Android 5.x and 6.x OS and applications; Java 8, used across all popular personal computer operating systems, such as Microsoft Windows, Apple, Open Solaris, and a plethora of popular Linux distributions (custom Linux OS versions) such as: SUSE, Ubuntu, Mint, Fedora, and Debian; and Java 9. This chapter covers the most foundational Java programming language concepts, techniques, principles, and structures that span these four major versions of Java that are currently in widespread use today on personal computers, iTV sets, and handheld devices, such as tablets and phones.

You start out with the easiest concepts and progress to the more difficult ones. The chapter begins at the highest level of Java—the Java API and its package—and progresses to the hands-on Java programming constructs contained in the Java packages, which are called Java *classes*. You learn about methods, as well as the variables or constants that classes contain, and what superclasses, subclasses, and nested classes are. Finally, you learn about Java `Objects` and how they form the foundation of OOP. You see what a constructor method is and how it creates a Java `Object` using a special kind of method that has the same name as the class in which it is contained.

Java Packages: Organizing a Java API Using Functional Classes

At the highest level of a programming platform—such as Google's 32-bit Android 4 or earlier, which uses Java SE 6; or 64-bit Android 5 or later, which uses Java SE 7; or the current Oracle Java SE platform, which was released as Java SE 8—there is a collection of packages that contain classes, interfaces, methods, and constants, which collectively form the *application programming interface (API)*. This collection of Java code can be used by application developers to create professional-level software across many OSs, platforms, and consumer electronics devices, such as desktops, laptops, netbooks, tablets, HD iTV sets, UHD iTV sets, eBook readers, and smartphones.

To install any given version of a Java API, you install the software development kit (SDK) as described in the book's appendixes. The Java SDK has a special name: the *Java development kit (JDK)*. If you're familiar with Android 5/6, which is actually Java 7 on top of Linux OS, you know that a new API level is released every time a few new features are added.

In addition to the API level defined by the SDK you install and use, the highest-level construct in the Java programming language is the *package*. You always use the `package` keyword to declare an application package at the top of your Java code. This needs to be the first line of code declared other than comments, which are not processed.

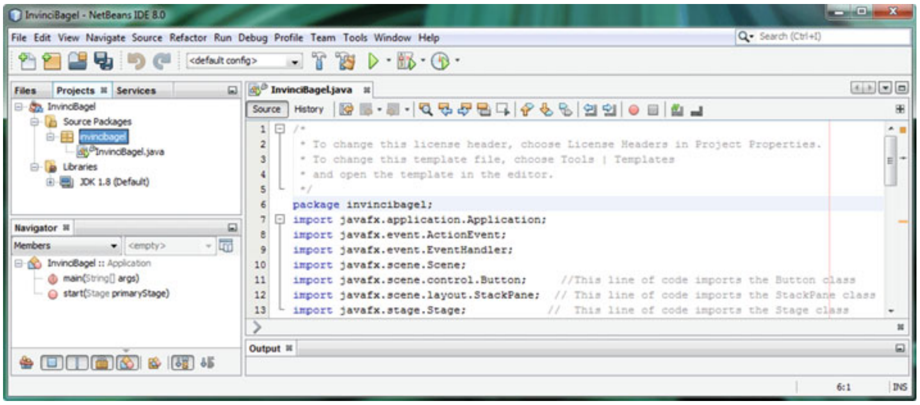
As you may have ascertained from the name, a Java package packages together all of your Java programming constructs. These include classes and methods (functions) that relate to the application: for example, a board game package contains all of your code, as well as the code you import to create, compile, and run the 3D board game. You take a look at the concept of importing and the Java `import` keyword next; they are closely related to the package concept.

A Java package is useful for organizing and containing your application code. But it is even more useful for organizing and containing the SDK's (API's) Java code that you use along with your own Java programming logic to create Java games or IoT applications.

You can use any of the classes that are part of the API packages you are developing with, by using the Java `import` keyword in conjunction with your package and the classes that you wish to use. This is called an *import statement*.

An import statement begins with `import`, followed by the package name and class reference path (full proper name); the statement needs to be terminated using a semicolon. For example, an import statement used to import the JavaFX `EventHandler` class from the `javafx.event` package should look like the following (also see Figure 4-1):

```
package invincibagel;             // custom invincibagel package for game
import javafx.event.EventHandler; // imports EventHandler into invincibagel
```



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package invincibagel;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;     //This line of code imports the Button class
import javafx.scene.layout.StackPane;  // This line of code imports the StackPane class
import javafx.stage.Stage;            //  This line of code imports the Stage class
```

***Figure 4-1.** The `package` and `import` keywords used in the Java code for the `InvinciBagel` game*

You can see the Java single-line comment // and multiline comment convention in this example, as well as in Figure 4-1. If you are interested in learning more about Java game programming, I took this example and screenshot from my book *Beginning Java 8 Game Programming* (Apress 2015), which covers the JavaFX new media engine.

An import statement informs the Java compiler that it needs to bring a specified external package into your custom package (in this example, import it into the `invincibagel` package), because you are using methods (and constants) from the class that is referenced using the `import` keyword, as well as specifying what package the class you are importing is stored in. If you use a class, method, or constants in your own Java class, such as the `BoardGame` class, and you haven't declared the class for use by using an import statement, the Java compiler will throw an error because it can't find the class it needs to use in your package.

Java Classes: OOP Modular Structures

The next-largest programming structure beneath the package level is the Java *class*; as you just saw, the import statement references both the package that contains the class and the class itself. Just as a package organizes all the related classes, a class organizes all of its related methods, data variables or data constants, and sometimes other nested classes as well (discussed in the next section). Classes let you make your code more modular, so it's not as structured (linear) as top-down programming languages. A Java class can be used to organize your Java code at the next logical level of functional organization, and therefore your classes contain Java code constructs that add specific functionality. These include methods, variables, constants, and nested classes, all of which are covered in this chapter.

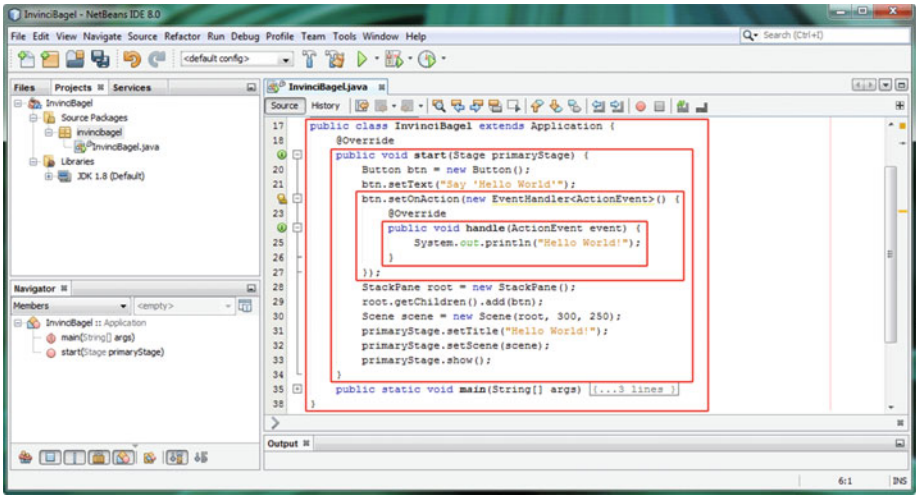
Java classes can also be used to create Java hard objects, which are discussed after you learn about classes, nested classes, methods, and data fields. Java `Objects` are constructed using a Java class and have the same name as the Java class and the exact same name as the class's constructor method, which you also see later in this chapter.

You can preface the declaration with Java modifier keywords that declare the class as being public or private or with other designators regarding what the class can and will do and for whom. Java modifier keywords are always placed before the Java `class` keyword, using the following format:

```
<modifier keywords> class <your custom classname goes here>
```

One of the powerful features of Java classes is that they can be used to modularize Java code. Your core application features can be a part of a high-level class, which can be subclassed to create more specialized versions of that class. This is a core feature of OOP. Once a Java class has been used to create a subclass, it becomes the *superclass*. A class always subclasses another superclass using the Java `extends` keyword.

As you can see in Figure 4-2, you declare a class using a Java `public` access modifier, then the Java `class` keyword, and a name for the class (in this case, `InvinciBagel`), which then uses the `extends` keyword to become a JavaFX `Application`. The import statement for `Application` is shown in Figure 4-1, under the `package` statement, naming the project (custom) package `invincibagel`. Notice that I am using NetBeans 8, which can be used for both Java and HTML5 development, supporting JSON development across all environments.



```
public class InvinciBagel extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {...3 lines }
}
```

Figure 4-2. *Public class* `InvinciBagel` *extends* `Application`*, creating a JavaFX game application*

Using the Java `extends` keyword tells a Java compiler that you want the superclass's capabilities and functionality added (extended) to your class, which, once it uses `extends`, becomes a *subclass*. A subclass extends the core functionality that is provided by the superclass it is extending. To extend your class definition to include a superclass, you add to (or extend, no pun intended) your existing class declaration using the following format:

```
<modifier keywords> class <your custom classname> extends <superclass>
```

When you extend a superclass using your class, which becomes the subclass of that superclass, you can use all of that superclass's features (nested classes, inner classes, methods, constructors, variables, and constants) in your subclass. You can use this code without having to explicitly rewrite (recode) these Java constructs in the body of your class, which would be redundant (and disorganized), because your class extends this superclass, making it a part of itself.

The *body* of a class is coded in curly braces, shown in the outer red box in Figure 4-2, which follow your class declaration. In Figure 4-2, the `InvinciBagel` class extends the Application superclass from the JavaFX application package. Doing this gives `InvinciBagel` everything it needs to host, or run, the JavaFX application. The JavaFX `Application` class constructs this `Application` object so that it can use system memory and call an `.init()` method to initialize anything that may need initializing, and also call a `.start()` method, which you can see in Figure 4-2 in the second red box. A `.start()` method is where you put Java code statements that will ultimately be needed to fire up (start) any JavaFX application. When an end user finishes using your InvinciBagel Java application, the `Application` (object) created by this `Application` class, using the `.Application()` constructor method, will call its `.stop()` method and remove your application from system memory. This will free up system memory space for other applications used by your end users.

The chapter explores Java methods, constructors, and objects next, as you progress from the higher-level package and class constructs, to lower-level constructs.

Java Methods: Code Constructs Providing Core Logic Functions

In Java classes, you generally have *methods* and the *data fields* (variables or constants) that the methods use for data they operate on. Because you are going from outer structures to inner structures, or top-level structures to lower-level structures, this chapter covers methods next. Methods are sometimes called functions in other programming languages, such as JavaScript.

You can see an example of the `.start()` method in Figure 4-2; this method holds the programming logic that creates the basic Hello World application. The programming logic in this `.start()` method uses Java programming statements to create a stage and a scene, place a button on the screen in the `StackPane`, and define event-handling logic so that when the button is clicked, the bootstrap Java code writes the "Hello World" text to the NetBeans IDE output area.

Declaring a Method: Modifier, Return Type, and Method Name

A method declaration starts with an *access-control modifier* keyword: `public`, `protected`, `private`, or package private. Package private is designated by not using any access-control modifier keyword. In Figure 4-2, the `.start()` method is declared using the `public` access-control modifier.

After the access-control modifier, you declare the method's *return type*. This is the *type of data* that the method will return after it is called (invoked). Because this `.start()` method performs setup operations but doesn't return any specific type of value, it uses the `void` return type, which signifies that the method performs tasks but does not return any data to the calling entity. In this case, the calling entity is the JavaFX `Application` class, because the `.start()` method is one of the key methods; the others are the `.stop()` and `.init()` methods provided by the `Application` superclass that the `InvinciBagel` class extends. This class controls the *application lifecycle stages* for this JavaFX application.

After the return type, you supply your method's name, which, by convention (or programming rules) should start with a *lowercase letter* (or word, preferably a verb). Any subsequent (internal) words (nouns or adjectives) start with a capital letter.

For instance, the method to display the `SplashScreen` should logically be named `.displaySplashScreen()`. Because it does something but does not return a value, it is `void` and is therefore declared using this empty Java code structure:

```
public void displaySplashScreen() { Java code to display splashscreen here }
```

You may need to pass *parameters*, which are named data values that need to be passed in and operated on in the *body* of the method, which is the part in curly braces. These parameters go in the parentheses attached to the end of the method name. In Figure 4-2, the `.start()` method receives the `Stage` object named `primaryStage` as its parameter, using the following Java method declaration programming syntax:

```
public void start(Stage primaryStage) { code to start your Application }
```

You can provide as many parameters as you like, using data type /parameter name pairs, with each pair separated by a comma. Methods are not required to have any parameters. If a method has no parameters, the parentheses are empty (right next to each other). This is how method names are written in this book, so you know they are methods: dot notation before and parentheses characters after the method name, like this: `.start()`, `.stop()`, and so on.

The programming logic that defines this method is contained in the body of the method, which as you have already learned is in curly braces that define the beginning and the end of the method. The programming logic in a method includes variable declarations, programming logic statements, and iterative control structures (loops), among other things, all of which you use to create JavaFX applications.

Constructor Methods: Turning a Java Class into a Java Object

This section covers a specialized type of Java method: a *constructor method*. This can be used to create, or *construct*, Java `Objects`, as you see later in the chapter. A constructor method could be considered a hard object in contrast to a JSON soft object, because Java requires a constructor method to create a Java `Object. Objects` in Java are simply called *objects*; the hard versus soft distinction is in JavaScript, which has both types of object declarations, as you see soon.

Objects are the foundation of OOP, so it is important for you to have an understanding of constructor methods before you learn about the Java `Object` itself. Because this section covers methods, this is the most logical place to look at object constructors (as constructor methods are sometimes called by veteran Java developers).

Creating a Java Object: Invoking the Class Constructor Method

A Java class contains a constructor method that must use the exact same name as the class itself. This method can be used to create Java `Objects` using the class. constructor methods use the Java class that contains them as the blueprint to create an instance of that class in system memory, which creates the Java `Object`. This constructor method always returns a Java `Object` type and thus does not use any of the Java return types that other methods typically use (`void`, `String`, `float`, `int`, `byte`, and so on). A constructor method should always be invoked using the Java `new` keyword, because you are creating a new Java `Object`! If you do not create a constructor method, the Java compiler will auto-create one for each class.

You can see an example of this in the bootstrap JavaFX code in Figure 4-2, in lines 20, 28, and 30. The new `Button`, `StackPane`, and `Scene` objects are created, respectively, using the following object-declaration, object-naming, and object-creation Java code structure:

```
<class name> <object instance name> = new <constructor method name> ;
```

A Java `Object` is declared in this fashion, using the class name, the name of the object you are constructing, the Java `new` keyword, and that class's constructor method name (and parameters, if any) in a single Java statement terminated with a semicolon character because each Java `Object` is an *instance* of a Java class. For example, on line 20 of the Java code in Figure 4-2, the portion to the left of the equals operator tells the Java language compiler that you want to create the `Button` object named `btn` using the JavaFX `Button` class as that object's blueprint. This declares the `Button` class (object type) and gives it a unique name.

The first part of creating the object is thus called the *object declaration*. The second part of creating a Java `Object` is called the *object instantiation*; it takes place on the right side of the equals operator and involves the object (class) constructor method along with the Java `new` keyword.

To instantiate the Java `Object`, you invoke, or use, the Java `new` keyword in conjunction with the object constructor method call. Because this takes place to the right of the equals operator, the result of the object instantiation is placed into the declared object, which is on the left side of the Java statement.

This completes the process of declaring (class name), naming (object name), creating (using the `new` keyword), configuring (using the constructor method), and loading (using the equals operator) your very own custom Java `Object`.

It's important to note that the declaration and instantiation parts of this process can be coded using *separate lines of Java code* as well. For instance, the `Button` object instantiation (Figure 4-2, line 20) could be coded as follows:

```
Button btn;           // Declare a Button object named btn
btn = new Button();   // Instantiate btn object using the Java new keyword
```

This is significant because coding object creation this way allows you to declare the object at the top of your class, where each method in the class that uses or accesses the objects can see it. In Java, objects or data fields are only visible in the Java programming construct (class or method) that they are declared in. So if methods are nested in a class, you should declare anything you want the methods to be able to access (see) at the top of the class construct and before the method constructs.

If you declare an object in a class, and therefore outside all the methods contained in the class, the methods in the class can then access (see or use) that object. Similarly, anything declared in a method is local to that method and is only visible to other members of that method, meaning all Java statements in that method's scope (what is within the `{...}` delimiters). In the current example, if you wanted to implement this separate object declaration in the class, outside the methods and object instantiation in the `.start()` method, the first few lines of the `InvinciBagel` class would look like the following:

```
public class InvinciBagel extends Application {
     Button btn;  // Declared outside of your start() method construct
     @Override
     public void start(Stage primaryStage) {
        btn = new Button();  // Instantiated in your start() method
        btn.setText("Say 'Hello World'");  // Object can now be utilized
        // other programming statements could continue in here
     }
}
```

When the object declaration and instantiation are split up this way, the `Button` object can be used or accessed by methods in the class other than `.start()`. In the previous code, other methods of the `InvinciBagel` class could call `.btn.setText()` without the Java compiler throwing any errors. The way the `Button` object is declared in Figure 4-2, only the `.start()` method can see the object, so only `.start()` can implement the method call; thus the `btn Button` object belongs solely to `.start()`, using the single-statement declare and instantiate approach.

Java Objects: Virtual Reality Using OOP with Java

Objects are the foundation of OOP languages—in this use case, Java. Everything in Java is based on the Java `Object` superclass. I like to call this the *master class*. The `Object` class is in the `java.lang` package, so an import statement for it references `java.lang.Object` (the full pathname to the Java `Object` class). All other Java classes are created (subclassed) using this class, because everything in Java is ultimately an `Object`.

Java `Objects` can be used to virtualize reality by allowing objects you see in everyday life. In Java applications, you can create objects using your imagination, to be realistically simulated. You do so by using data fields (variables and constants) and methods. These Java programming constructs make up your object's *characteristics* or attributes (constants), *states* (variables), and *behaviors* (methods). Java class constructs organize each object definition (constants, variables, and methods) to give birth to an instance of that object, using the constructor method for the class, which designs and defines the object construction.

Designing a Java Object: Constants, Variables, and Methods

One way to think about Java `Objects` is that like they are nouns; things (objects) that exist in and of themselves. The object behaviour is created using methods like verbs: things the nouns can do. As an example, let's consider that very popular object in everyone's life: a car. Let's define the `Car` object's attributes to see how an object can be defined using Java. Some characteristics, or attributes that do not change, held in *constants*, might be defined as follows:

- `Paint Color (Candy Apple Red, Metallic Blue, Silver, White, Black)`
- `Engine Fuel (gas, diesel, biodiesel, hydrogen, propane, electric)`
- `Drive Train (2 Wheel Drive or 4 Wheel Drive)`

Some *states*, which change, and which define the car in real time, held in *variables*, could be defined as follows. They hold the current values for direction, speed, and what gear you are in:

- `Direction (N, S, E, W)`
- `Speed (15 miles per hour)`
- `Current Gear (1, 2, 3, 4, 5, Reverse, Park)`

The following are some things the Car object should be able do—its *behaviors*, or how it functions. In JavaScript, these are called functions; in Java, however, they are called methods. Java Object behaviors might be defined using these methods:

- `Accelerate!`
- `Shift Gears`
- `Apply Brake`
- `Turn the wheels`
- `Turn on the stereo`
- `Use the headlights`
- `Use the turn signals`

You get the idea. Figure 4-3 shows a simple diagram of this Java Object structure. It includes the characteristics, or attributes, of the car that are central to defining the Car object, and the behaviors that can be used with the Car object. These attributes and behaviors define the car to the outside world.

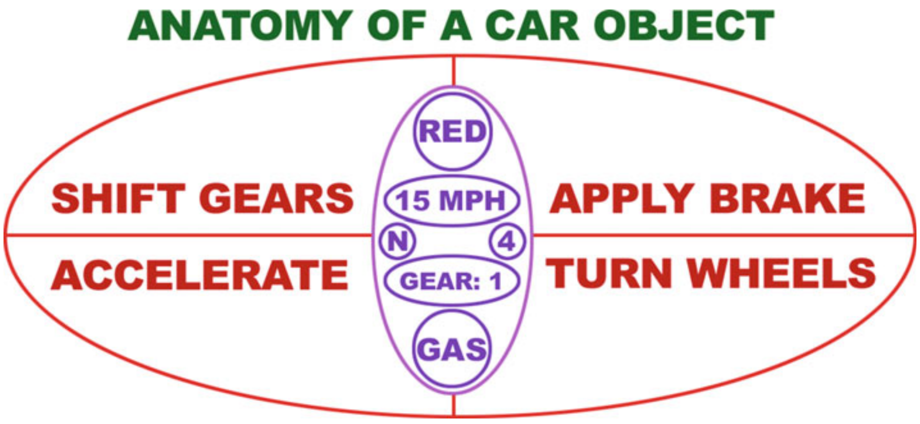


***Figure 4-3.** The anatomy of a Car object, with methods encapsulating variables or constants*

Objects can be as complicated as you want them to be. Java Objects can also nest or contain other Java Objects in the object structure (object hierarchy). An object hierarchy is like a tree structure, with a main trunk, branches, and sub-branches as you move up (or down) the tree, very similar to JavaFX or 3D software scene graph hierarchies.

A perfect example of a hierarchy that you use every day is the multilevel directory (file folder) structure on your computer's hard disk drive (see Figure 1-1). Directories or folders on a hard drive can contain other directories or folders, which can in turn contain yet other directories and folders. This allows complex organizational hierarchies to be created—and an object is similar in its hierarchical organizational capabilities.

You'll notice that, in real life, objects can be made up of other objects. For example, a car engine object is made up of hundreds of discrete objects contained in subcomponents (like the carburetor) that all function together to make the engine object work as a whole.

This same construction of more complicated objects out of simpler objects should be mirrored using OOP languages: complex hierarchies of objects can contain other objects so the structure is well organized and logically defined. Many of these objects are created using preexisting, or previously developed, Java or JavaScript code. This is one of the objectives of object-oriented, modular programming practices.

As a good exercise, you should practice identifying different complex objects in the room around you and then break their definition, or description, down into variable states and constant characteristics, as well as behaviors, or things that these objects can do. Practice creating hypothetical object and sub-object hierarchies. This is a great exercise, because this is how you eventually need to start thinking in order to become more successful in your professional OOP endeavors using Java, JavaScript, or the JavaFX engine in the Java 8 programming language framework.

Encoding Objects: Turning an Object Design into Java Code

To illustrate how to define an object in Java, let's construct a basic class for the `Car` example. To create a `Car` class, you use the Java keyword `class`, followed by the custom name for the new class you are coding, and then curly brackets, which contain a Java class definition. The first things that you usually put in the class (in the curly `{}` brackets) are the data fields (variables). The variables hold the states, or characteristics, of this `Car` object. In this case, six data fields define the car's current gear, current speed, current direction, fuel type, color, and drive-train (two-wheel drive or four-wheel drive), as specified earlier in Figure 4-3.

With all six variables in place, the `Car` class (object blueprint) definition initially looks something just like this:

```
class Car {
    int speed = 15;
    int gear = 1;
    int drivetrain = 4;
    String direction = "N";
    String color = "Red";
    String fuel = "Gas";
}
```

Remember that because you're specifying the starting values using the equals sign for all the variables, these object properties (variables) all contain these default data values. These initial data values are set in the system memory as the `Car` object's default values at construction.

The next part of the Java class definition file contains the object methods. Java methods should define how your `Car` object functions—that is, how it operates on the variables you defined at the beginning of the class. Remember, these hold the `Car` object's current state of operation. Calling these methods invokes the variable state change; methods can also return data values to the entity that calls or invokes the method. Return values may include data values that have been successfully changed, or the result of an equation.

For instance, there should be a method to allow users to shift gears by setting the Car object's gear variable or attribute to a different value. This method should be declared void, because it performs a function but does not return any data values. In the Car class and Car object definition example, you have four methods, as defined in Figure 4-3.

The .shiftGears() method sets the Car object's gear attribute to the newGear value passed in to the .shiftGears() method. You should allow an integer to be passed in to this method to allow for user errors, just as in the real world a user might accidentally shift from first into fourth gear:

```
void shiftGears (int newGear) {
     gear = newGear;
}
```

The .accelerateSpeed() method takes your object's speed state variable and adds an acceleration factor to it. It then sets the result of this addition operation back into the original speed variable, so that the object's speed state now contains the new (accelerated) speed value:

```
void accelerateSpeed (int acceleration) {
     speed = speed + acceleration;
}
```

The .applyBrake() method takes the object's speed state variable and subtracts brakingFactor from it. It then sets the result of this subtraction back into the original speed variable, so the object's speed state now contains the updated (decelerated) braking value:

```
void applyBrake (int brakingFactor) {
     speed = speed - brakingFactor;
}
```

The .turnWheels() method is straightforward, much like the .shiftGears() method, except that it uses a String value of N, S, E, or W to control the direction in which the car turns. When .turnWheels("W") is used, the Car object turns to the left; and when .turnWheels("E") is used, the car turns to the right—given, of course, that the Car object is currently heading north, which, according to its default direction setting, it is:

```
void turnWheels (String newDirection) {
     direction = newDirection;
}
```

The methods that make a Car object function go in the class after the variable declarations, as follows:

```
class Car {
    int speed = 15;
    int gear = 1;
    int drivetrain = 4;
    String direction = "N";
    String color = "Red";
    String fuel = "Gas";

    void shiftGears (int newGear) {
        gear = newGear;
    }

    void accelerateSpeed (int acceleration) {
        speed = speed + acceleration;
    }

    void applyBrake (int brakingFactor) {
        speed = speed - brakingFactor;
    }

    void turnWheels (String newDirection) {
        direction = newDirection;
    }
}
```

Next, let's take a look at how to add the `.Car()` constructor method into this class.

Constructing Objects: Coding Your Constructor Method

This `Car` class lets you define a `Car` object even if you don't specifically include a `.Car()` constructor method, discussed in this section. This is why a collection of variable settings becomes the `Car` object's defaults. It is best to code your own constructor method, however, so that you take total control over your object creation and don't have to preinitialize the variables to one value or another. The first thing to do is make your variable declarations undefined, removing the equals sign and initial data values, as shown in this modified `Car` class:

```
class Car {
    String name;
    int speed;
    int gear;
    int drivetrain;
    String direction;
    String color;
    String fuel;

    public Car (String carName) {
        name = carName;
        speed = 15;
        gear = 1;
        drivetrain = 4;
        direction = "N";
        color = "Red";
        fuel = "Gas";
    }
}
```

The `.Car()` constructor method sets the default data values as a part of the construction and configuration of a `Car` object. As you can see, you add a `String` variable to hold the `Car` object's name, with a default `name` parameter set to the text data value `carName`.

Java constructor methods differ from regular Java methods in a number of distinct ways. First, they do not use any data return type, such as `void` or `int`, because they are used to create Java `Objects` rather than to perform functions. They do not return nothing (`void` keyword) or a number (`int` or `float` keyword), but rather return an object of type `java.lang.object`. Note that every class that needs to create an object features a constructor with the same name as the class itself, so a constructor is one method type whose name can and should always start with a capital letter. If you do not code a constructor, your Java compiler will create one for you.

Another difference between constructor methods and standard Java methods is that constructors need to use a `public`, `private`, or `protected` access-control modifier and cannot use any non-access-control modifiers. Therefore, be sure not to declare your constructor as static, final, abstract, or synchronized.

Creating Objects: Object Instantiation Using the new Keyword

The syntax for constructing an instance of a Java `Object` is similar to declaring a variable. It also uses the Java `new` keyword and the constructor method, using this format:

```
Car myCarObject = new Car();
```

To access the `Car` object's properties (variables) or characteristics (constants), you can use *dot notation*, which is used to chain, or reference, Java constructs to each other. For strict Java programming, you would follow the OOP principle of *encapsulation* and use getter `.getProperty()` and setter `.setProperty()` methods that must be called to access the object property in a more controlled fashion. I am covering the similarities to JavaScript here: dot notation can be used in both Java and JavaScript to access object properties directly without any encapsulation enforced.

Once a Java `Car` object has been declared, named, and instantiated, you can then reference its properties. This is done, for example, using the following Java `Object` format:

```
objectName.propertyName;
```

So to access the `Car` object name, you use the following Java code construct:

```
myCarObject.name
```

To invoke your `Car` object methods using this `myCarObject Car` object also requires the use of dot notation. For example, you can use the following Java construct:

```
objectName.methodName(parameter list variable);
```

So, to shift a `Car` object into third gear, if this `Car` object instance is named `myCarObject`, you use the following Java programming statement:

```
myCarObject.shiftGears(3);
```

This calls or invokes the `.shiftGears()` method of the `myCarObject Car` object and passes the `gear` parameter, which contains the integer value of 3. This is placed into the `newGear` variable, which is used by the `.shiftGears()` method's internal code to change the `gear` attribute of the `myCarObject Car` object instance to third gear. If you think about it, how Java works is very logical, and pretty darned cool as well.

Extending an Object Structure: The OOP Concept of Inheritance

There is also support in Java for developing different types of enhanced classes, and therefore enhanced (more complex or detailed) objects. This is done using a technique called *inheritance*. Inheritance lets you create more specialized classes that contain uniquely defined objects using your original (foundational) object. For instance, the `Car` class can be subclassed using the original `Car` superclass. Once you subclass any class, it becomes a superclass. Ultimately, there can be only one superclass, at the very top of the class chain, but there can be an unlimited number of subclasses. All of these subclasses inherit the methods and data fields from their superclass. The ultimate example of this in Java is the `java.lang.Object` superclass (I sometimes call this the master class), which is used to create all other classes in Java. Every class in Java, because it has a constructor, is also an object! Mind-boggling, to say the least—but once you wrap your head around it, OOP is both logical and powerful.

As an example of inheritance using the `Car` class, you can subclass an `SUV` class, using the `Car` class as the superclass. This is done by using the Java `extends` keyword, which extends a `Car` class definition in order to create an `SUV` class definition. The `SUV` class then defines *only those additional attributes* (constants), *states* (variables), and *behaviors* (methods) that apply to the `SUV` type of `Car` object. The `SUV` object additionally extends all the attributes (constants), states (variables), and behaviors (methods) that apply to all types of `Car` objects as defined by the `Car` superclass.

This is the functionality that the Java `extends` keyword provides for this subclassing (or inheritance) operation, and this is one of the more important and useful features of *code modularization* in OOP languages. You can see the modularization visually in Figure 4-4, which adds `Car` features for each of the subclasses (at the top, in orange).

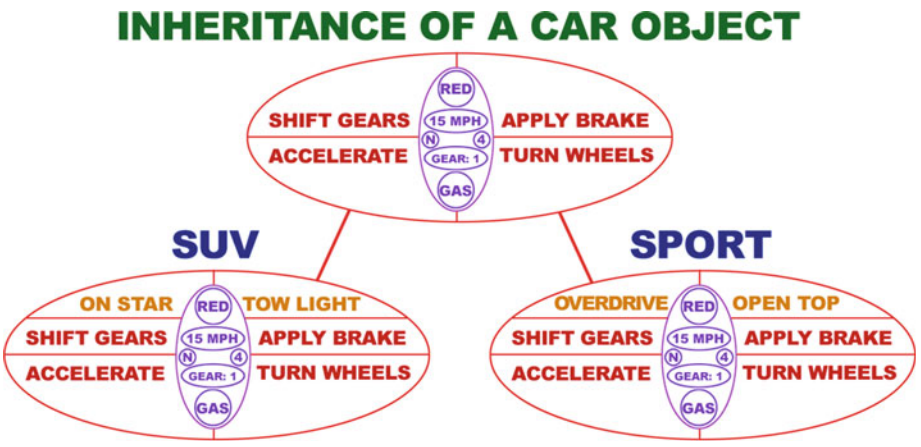


Figure 4-4. *OOP inheritance of the Car object allows you to create an SUV object or Sport object*

As an example, the SUV Car object subclass can have additional `.onStarCall()` and `.turnTowLightOn()` methods defined in addition to inheriting the usual operational methods that allow a Car object to shift gears, accelerate, apply the brakes, and turn the wheels.

Similarly, you can generate a second subclass called the Sport class, which creates Sport Car objects. These may, for example, include an `.activateOverdrive()` method to provide faster gearing, and maybe an `.openTop()` method to put down the convertible roof. To create the subclass using the superclass, you extend the subclass from the superclass by using a Java `extends` keyword in your class declaration. The Java subclass construct thus looks like the following Java SUV class construct, which uses the Java `super` keyword to generate a new `applyBrake()` method programming structure that makes the brakes twice as effective:

```
class SUV extends Car {
    void applyBrake (int brakingFactor) {
        super.applyBrake(brakingFactor);
        speed = speed - brakingFactor;
    }
}
```

This extends the SUV object to have access to (essentially, to contain) all the data fields and methods that the Car object (class) features. You can focus on just the new, or different, data fields and methods, which relate to differentiating the SUV object from the regular or master Car object superclass definition.

To refer to one of your superclass's methods from within the subclass you are coding, you can use the Java `super` keyword. For example, in the new SUV class, you may want to use the Car superclass's `.applyBrake()` method and then apply some additional functionality to the brake that is specific to the SUV. You call the Car object's `.applyBrake()` method by using `super.applyBrake(brakingFactor);` in the Java code for the SUV Car object.

The Java code shown previously adds functionality to the `Car` object's `.applyBrake()` method, in the `SUV` object's `.applyBrake()` method, by using the `super` keyword to access the `Car` object's `.applyBrake()` method; it then adds logic to make `brakingFactor` apply twice. This serves to give the SUV object twice the braking power of a standard car, which an SUV needs in order to stop its far greater mass.

Next, let's take a closer look at JavaScript's OOP approach. JavaScript is even more closely aligned with JSON than the Java programming languages are.

JavaScript OOP Concepts: Hard and Soft Objects

JavaScript can create objects in a couple of different ways. There are hard (constructed) objects and or soft (literal notation) objects, which is what you define using JSON. This section looks at the difference between them and how you access JavaScript objects using each of these two object-encoding approaches. This provides a parallel—at least, using the hard-object approach—to what you looked at in the previous Java `Object` sections. (I capitalize `Object` in Java usage because it is a proper class name, from the `java.lang.Object` master class.) You will find, possibly due to the popularity of Java, that JavaScript allows you to do things in a very similar fashion to Java, using *constructor functions*. Let's look at the hard-object encoding approach first and then look at the soft-object approach, which is more compatible with JSON structures. Finally, you see the JavaScript Object Notation (JSON) description format to finish this chapter.

JavaScript Hard Objects: Using a Constructor Function

Just as you saw with Java `Objects` and constructor methods, you can also create a constructor function in JavaScript to create an object. This section doesn't look at this in much detail, because you already saw this approach using the Java 9 OOP language, and because the JSON model is tailor-made for using the JavaScript soft-object approach (discussed next). A `Car` object constructor function in JavaScript looks like the following code, if you follow the object diagram shown in Figure 4-3 and the definition created in the previous section:

```
function carObject() {
     this.name = 'carName';
     this.speed = 15;
     this.gear = 1;
     this.drivetrain = 4;
     this.direction = 'North';
     this.color = 'Red';
     this.fuel = 'Gas';
     this.ApplyBrake = function() { Brake Application Code in Here };
     this.shiftGears = function() { Gear Shifting Code in Here };
     this.TurnWheels = function() { Wheel Turning Code in Here };
     this.accelerate = function() { Acceleration Code in Here };
};
```

JavaScript literal notation defines an object as a variable, using the var keyword, instead of as a constructor function. Let's take a look at that slightly different approach next.

JavaScript Soft Objects: Using Literal Notation to Define a Variable

The second way to define an object in JavaScript is as a *variable* using *literal notation*. A Car object variable declaration in JavaScript using literal notation looks like the following data construct, if you again follow the Car object diagram in Figure 4-3:

```
var carObject = {
     name : 'myCarsName',
     speed : 15,
     gear : 1,
     drivetrain : 4,
     direction : 'North',
     color : 'Red',
     fuel : 'Gas',
     ApplyBrake : function() { Brake Application Code in Here },
     shiftGears : function() { Gear Shifting Code in Here },
     TurnWheels : function() { Wheel Turning Code in Here },
     accelerate : function() { Acceleration Code in Here }
};
```

Let's look at the differences between these two object-definition approaches.

Differences Between a Constructor Function and Literal Notation

As you can see, the primary difference in the declaration of the constructor function is that it uses the this keyword. The this keyword is used in both Java and JavaScript and allows an object to reference itself. The literal notation does not use the this keyword. Secondarily, a constructor function uses the equals operator (=) to assign values to the object's properties and functions, whereas the literal notation object definition uses the colon assignment operator (:).

A constructor function can use (optional) *semicolons* (;) at the end of each assignment statement for both properties and functions. On the other hand, an object defined as a variable via literal notation is supposed to use *commas* (,) after each assignment statement if there is more than one, which there is in the majority of JSON applications.

As you can see, other than using different keywords (JavaScript uses function, and Java uses method), the OOP languages JavaScript and Java define object construction logic in a similar fashion. These also access object properties and functions in a similar format, using dot notation. Let's take a look at this next.

Accessing JavaScript Objects: Using Dot Notation

To access an object property, you use the object name, then a dot (period), and then the property name, like this: `ObjectName.PropertyName`. If you are using a constructor function (or method in Java), you first instantiate the object using the OOP `new` keyword, like this:

```
var myCarObject = new carObject;     (First Construct Your Object Instance)
myCarObject.name                     (this will return a String: myCarName)
myCarObject.accelerate()   (executes code inside the accelerate() function)
```

If you create an object using literal notation, you do not have to construct the object first, so you can access the JavaScript object without having to first construct an instance of it. The following code example accesses a property and a function:

```
carObject.name         (this will return the String myCarName)
carObject.accelerate() (executes the code inside the accelerate() function)
```

Now you're ready to see the differences when you define objects using JSON.

Defining Soft Objects: Using JavaScript Object Notation (JSON)

As you can see from the `json.org` object definition diagram reproduced in Figure 4-5, the JSON object definition uses the soft-object literal notation approach but does not support the function definitions. The JSON object data definition is contained in curly braces (`{}`) and uses a colon (`:`) to separate the key (object attribute name) from the value (object attribute data value) and a comma (`,`) to separate key-value pairs (object attributes and values) from each other.

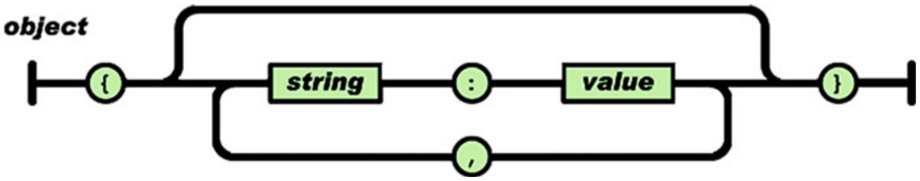


Figure 4-5. *The JSON object-definition diagram (from the `json.org` web site)*

This is what the `Car` object's attributes (data definition) look like using JSON:

```
{ "name":       "carName",
  "speed":      15,
  "gear":       1,
  "drivetrain": 4,
  "direction":  "North",
  "color":      "Red",
  "fuel":       "Gas"    }
```

There are ways to get JavaScript functions into a JSON object by using `String` data and parsing that into functions as part of your JSON parsing and evaluation process. If you really need this functionality (no pun intended), this limitation can be worked around. JSON is primarily designed to construct data object definitions that contain object parameters and their corresponding data values, so functions are generally externalized with JSON application development design approaches.

Summary

This chapter looked at two of the most popular object OOP languages in the world today: Oracle Java and ECMAScript-262 JavaScript. You saw how OOP languages allow you to logically stratify, define, and modularize your apps, and looked at how objects are constructed and referenced using Java, JavaScript, and JSON. In the next chapter, you get into arrays of data and how these are handled in JSON.

CHAPTER 5

JSON Arrays: Serialized Data Store Structures

This chapter examines a data structure that holds more complex data representations: an *array*. An array is similar to a database structure, but far more simplified in nature. Arrays are found in most every programming language, including OOP languages as well as non-OOP languages. You even hear the word *array* used in natural languages, such as in the phrase "We have a wide array of products," referring to a store's stock or collection of items to be sold.

In this chapter, you look at what an array of data is and how it is used and accessed using JavaScript, because this language is tied the most closely to JSON. After that, you see how arrays of data are represented using JSON, and how the popular Java open source language supports JSON arrays using custom classes and libraries.

An Overview of Arrays: Data Structures

The term *array* is most often used to mean the `array data type`. This is a type of data collection that is provided by almost all high-level programming languages. An array consists of an ordered collection of data values held in variables that can be selected by your code constructs. This is done by using one or more indices that are computed at runtime. In some cases, the term *vector* is used in computing to refer to an array, although *tuple* is the more correct mathematical equivalent and is another synonym. Arrays are often used to implement data tables, especially lookup tables in databases. For this reason, the word *table* is also sometimes used as a synonym for *array*.

An array is the oldest and most important computer programming data structure: holds the data to be operated on and is used by almost every software program ever coded. Arrays are also used to implement other important data structures, such as strings and lists. This is because arrays effectively use the inherently linear memory addressing logic and schema used by modern-day computers.

In most computers, as well as most external storage devices, the memory is a one-dimensional array of byte words, whose indices are the memory addresses themselves. Processors, also known as CPUs, are also optimized for array operation so that data can be quickly and efficiently retrieved, speeding up application performance and therefore also enhancing the user experience.

W. Jackson, *JSON Quick Syntax Reference*, DOI 10.1007/978-1-4842-1863-1_5

Array data types are implemented using some type of array structure. These may vary across languages and are often implemented by using hash tables, linked lists, search trees, and similar data structures that are all akin to simplified databases (data stores). In computer programming, an array data structure, usually referred to simply as an array, is a data structure consisting of a collection of *elements* (data values held in variables).

Each of these elements is identified by at least one array's *index*. An index is also sometimes referred to as the *key* because it allows you to reference (retrieve) the data. The simplest type of data structure is a *linear array*, also called a *one-dimensional array*.

An array is stored in such a way that the data access (position in the array) for each element can be computed from the array index or key using a basic mathematical formula to access linear data items (records). As an example, the array of eight 32-bit integer variables with indices 0 through 7 is stored as eight words at memory addresses 2000, 2004, 2008, 2012, 2016, 2020, 2024, and 2028. So, the element with index `i` has the address `index[i] = 2000 + 4 × i`.

There are also two-dimensional arrays, sometimes referred to as *matrices* because the mathematical concept of a matrix can be represented as a two-dimensional grid; there are three-dimensional arrays as well. The data elements of array data structures are usually required to have the same data size and the same data structure, and also use the same data-type representation, so that an element index can be accurately computed during runtime. Among other things, this data-uniformity requirement allows any single iterative statement to process an arbitrary number of data elements in any data array. Sets of valid index tuples and the addresses of the elements, and hence the element-addressing formula, usually, but not always, are 100% fixed while a data array is being accessed.

Declaring a JavaScript Array: Variable Declaration

Let's follow the format used in Chapter 4 and first look at how to define a data array using the JavaScript programming language. Then you see how to access this data using JavaScript dot notation (this works in Java as well) and how arrays can be represented using JSON. Similar to soft-object literal notation, you start the array data declaration with the `var` (variable) keyword and the name that you wish to use for your array, followed by an equals operator. Instead of the curly braces `{...}` used for defining objects, arrays use square brackets `[...]` instead. To define an array containing a used car dealership's inventory, including make, model, color, and list price, you can create an array data structure where each data record is an object, with colon-delimited key:value pairs separated using commas, and with the object definitions themselves separated by commas within the array, which is defined using square brackets. Remember that attribute names and string values are contained in double quotes. The JavaScript code looks like the following array data definition:

```
var carInventory = [
    { "carName":"Toyota", "model":"Prius", "color":"Green", "price":24995 },
    { "carName":"Nissan", "model":"300ZX", "color":"Brown", "price":21795 },
    { "carName":"Dodge",  "model":"Viper", "color":"White", "price":26495 },
    { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "price":27895 }
]
```

Now that you have defined an array of data objects, let's look at how you access elements of arrays using the array name, element indices, object properties, and dot notation.

Accessing a JavaScript Array: Using the Index

Array data elements are accessed by using their indices, which are contained in square brackets just like the array itself. As you may imagine, the array name comes first, using an integer index specification, to specify which data record you wish to access. Then the dot-notation syntax is used to chain together that data record with the property name you wish to reference. The following example puts together a construct that accesses the first (`carName`) object attribute (`"Toyota"`):

```
carInventory[0].carName       // Will return the String value: "Toyota"
```

You can also *concatenate* data together using a plus operator and insert blank space using quotation mark delimiters with a space in between. The following example takes a `carName` object attribute and appends it to the `model` object attribute, creating the full car manufacturer and model name as a result (`"Toyota Prius"`):

```
carInventory[0].carName + " " + carInventory[0].model     // "Toyota Prius"
```

There's an alternate way to access object attributes in an array data record, which does not use dot notation. This involves using the square brackets used with arrays to hold a quoted attribute name value. So instead of using `arrayName[n].attributeName`, you can use `[n]["attributeName"]`, as shown in the following JavaScript code example:

```
carInventory[0]["carName"]   // Will also return the String value: "Toyota"
```

Again, you can concatenate data together using a plus operator and insert blank space using quotation mark delimiters with a space between them. This example takes a `carName` object attribute and appends it to the `model` object attribute, resulting in the manufacturer and model name (`"Toyota Prius"`):

```
carInventory[0]["carName"] +" "+ carInventory[0]["model"] // "Toyota Prius"
```

You can create constructs that access the array object data in whatever order suits your purpose. The following data-access example outputs a more descriptive formatting of the object data, perhaps for sales and marketing purposes:

```
carInventory[3].color   + " "   +
carInventory[3].carName + " "   +
carInventory[3].model   + ": $" +
carInventory[3].price             // Returns: "Blue Chevy Camaro: $27895"
```

This can also be represented using the alternate array object attribute data-access format, which uses the following JavaScript code syntax:

```
carInventory[3]["color"]   + " "   +
carInventory[3]["carName"] + " "   +
carInventory[3]["model"]   + ": $" +
carInventory[3]["price"]            // Returns: "Blue Chevy Camaro: $27895"
```

Next, let's look at the syntax for defining an array using JSON. This is similar to the JavaScript approach but uses a colon instead of the equals operator and does not use the var (variable) keyword. After that, you look at how Java 8 provides JSON array and object libraries for handling the object and array structures you saw in the previous two chapters.

Defining a JSON Array: Using the Colon Operator

The basic format for JSON array data structures can be found on the json.org web site and uses square brackets to contain the data values, which are comma delimited (separated by commas). You saw this in the previous section, where you separated data objects using commas, with each object structure contained in curly braces. The basic JSON array format diagram, from json.org, is reproduced in Figure 5-1.

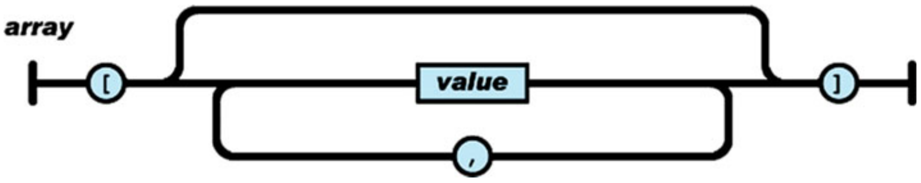


***Figure 5-1.** A JSON array is contained in square brackets and uses commas to separate values*

A basic JSON array of number values thus uses the following data format:

```
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 ]
```

The name for a JSON array is contained in double quotation marks, using the colon operator between the JSON array name and the JSON array structure itself. Here is an example of how the array data structure from the previous section is created as a JSON array data definition structure:

```
"carInventory" : [
    { "carName":"Toyota", "model":"Prius", "color":"Green", "price":24995 },
    { "carName":"Nissan", "model":"300ZX", "color":"Brown", "price":21795 },
    { "carName":"Dodge",  "model":"Viper", "color":"White", "price":26495 },
    { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "price":27895 }
]
```

Now that you've seen the two major JSON data structure types, objects and arrays, the next section presents an overview of the Java 8 programming language capabilities regarding JSON, and after that, you can proceed to the chapters covering numbers, characters and other data values.

Java JSON Support: JSON Utility Classes

Java has an API called JSR 353 created solely for JSON data processing. This API provides portable API components that allow you to parse, generate, transform, and query JSON data definitions using both Object Model and Streaming API libraries and classes. The JSON Object Model API creates a random-access, tree-like structure that represents the JSON data in system memory. This tree-like data structure can subsequently be navigated or queried.

The object programming model is the most flexible approach, enabling processing that requires a random access to the complete contents of the object data-tree structure. This object model usually is not as efficient as a streaming model, requiring more system memory to implement, because it must install the entire data representation into system memory locations.

The JSON Streaming API provides a way to parse and generate JSON using a streaming approach without having to have all the data in memory at one time. This streaming model gives you JSON parsing or generation control. The JSON Streaming API provides an event-based parser, letting you request the next event rather than handle the event using an event callback infrastructure.

This approach gives you more procedural control over JSON data processing. JSON streaming model application code can process or discard the parser event and ask for the next event, which is called *pulling* the event.

The streaming model is appropriate for local processing, where random access of other parts of JSON data is not required. Additionally, the JSON Streaming API provides a way to generate well-formed JSON to a stream by writing one event at a time.

JSON Object Model: Java Object and Array Builder

The JSON Object Model API is similar to the Document Object Model (DOM) API for XML or HTML5. The JSON Object Model is a high-level API that provides immutable object models for JSON object structures, which you learned about in Chapter 5; and JSON array structures, which you saw earlier in this chapter. These JSON structures are represented as Java object models using the Java types `JSONObject` and `JSONArray`.

The `JSONObject` class provides a `Map` object view to access an unordered collection of zero or more key:value pairs from a JSON data-object definition model. The `JSONArray` class provides a `List` object view to access an ordered sequence of zero or more values from a JSON data-array definition model.

The `JSONObject`, `JSONArray`, `JSONString`, and `JSONNumber` classes are subclasses, and therefore subtypes, of the `JSONValue` superclass. Constants are defined in this JSON API for `null`, `true`, and `false` JSON values, which are covered in Chapter 6. These are held in the `JSONNull` and `JSONBoolean` subclasses.

The JSON Object Model API uses *builder patterns* to create JSON object models from scratch. Application code can use the `JSONObjectBuilder` interface to create the JSON data-definition models that represent JSON objects. The resulting model is of the `Object` type `JSONObject`.

Application code can use the Java interface `JSONArrayBuilder` to create models that represent JSON `Array Objects`. The resulting model from this type of builder is of `Object` type `JSONArray`.

These object models can also be created using an input source (such as `InputStream` or `Reader`) using the Java interface `JSONReader`. Similarly, these object models can be written to an output source, such as `OutputStream` or `Writer`, using the Java class `JSONWriter`. These classes and interfaces are shown in Table 5-1, which lists the primary Java classes and Java interfaces contained in the JSON Object Model API for the Java programming language.

Table 5-1. *Primary Classes and Interfaces in the Java JSON Object Model API*

Class or Interface	Description
`JSON`	Contains static methods to create JSON readers, writers, builders, and their factory objects
`JSONGenerator`	Writes JSON data to a stream one value at a time
`JSONReader`	Reads JSON data from a stream, and creates an object model in memory
`JSONObjectBuilder`	Creates an object model in memory by adding values from application code
`JSONArrayBuilder`	Creates an array model in memory by adding values from application code
`JSONWriter`	Writes an object model from memory to a stream
`JSONObject`	Represents the JSON `Object` data type
`JSONArray`	Represents the JSON `Array` data type
`JSONValue`	Represents data types for values in JSON data
`JSONNumber`	Represents data types for numbers in JSON data
`JSONString`	Represents data types for strings in JSON data
`JSONBoolean`	Represents Boolean data types for `TRUE` and `FALSE` in JSON data
`JSONNull`	Represents the `Empty` data type for `NULL` in JSON data

The next section presents a brief overview of the JSON Streaming Model API for Oracle Java.

The JSON Streaming Model: Parser and Generator

The JSON Streaming API is a low-level API that is designed to process very large amounts of JSON data efficiently. Other JSON frameworks (such as JSON binding) can also be implemented using this API. The JSON Streaming API is similar to the StAX XML Streaming API. The JSON Streaming API consists of the Java interface `JSONParser` as well as the Java interface `JSONGenerator`.

As you may have guessed, the `JSONParser` interface contains methods to parse JSON data using this JSON streaming model. The `JSONGenerator` interface contains methods for writing JSON data stream into an output source.

The `JSONParser` interface provides forward-moving, sequential, read-only access to JSON data by using the pull-parsing programming model. In this model, your application code controls the thread and calls methods in the `JSONParser` interface to move the parser forward sequentially or, alternately, to obtain JSON data from the current state of this parser.

The `JSONGenerator` interface, on the other hand, provides Java methods to write JSON data out to a datastream. This interface should be used to write your key:value pairs into JSON objects and to write sequential data values into JSON arrays. The primary Java classes and interfaces contained in the JSON Streaming API are shown in Table 5-2; note that the `JSON` class spans both the object and streaming models.

Table 5-2. *Primary Classes and Interfaces in the Java JSON Streaming Model API*

Class or Interface	Description
`JSON`	Contains static methods to create JSON readers, writers, builders, and their factory objects
`JSONParser`	Represents an event-based parser that can read JSON data from a stream
`JSONGenerator`	Writes JSON data to the stream one value at a time

Now that you have learned about JSON objects and arrays, you can examine the different types of data values that can be contained in these two types of JSON structures.

Summary

This chapter looked at how to define and access data arrays in JavaScript, JSON, and Java using the JSON APIs. You saw how data arrays allow you to store large collections of sequential data for applications, and how data arrays can be constructed and referenced using JavaScript, JSON, and Java. In Chapter 6, you learn about the data types and values that are allowed in JSON.

CHAPTER 6

JSON Data Values: Types of Data that JSON Supports

You have looked at the two types of JSON data structures: one used to define objects, covered in Chapter 4; and the other used to define data arrays, covered in Chapter 5. This chapter discusses the different types of data values that can be supported in your JSON data-definition structures. Many types of data values are supported in computer programming languages, and JSON must work across most if not all of these to define data structures. Thus it must implicitly support all the core types of data values, such as numbers, characters, and booleans, as well as more advanced data structures, such as objects and arrays.

This chapter looks at the data types that are supported in JSON and also how they are supported in JavaScript and Java, because these languages are used the most frequently with JSON. After that, you examine each of these data types.

JSON Value: Supported JSON Data Types

The diagram in Figure 6-1, from the `json.org` web site, shows the data value types that JSON supports. `String` values hold textual values, `number` values hold numeric values, objects and arrays work as discussed in Chapters 4 and 5, and Boolean values can be `true` or `false`. An empty data value is represented using a `null` value. This chapter covers strings, numbers, Booleans, and `null` at a basic level. Chapter 7 covers string values in detail, and Chapter 8 looks at number values.

String Value: The Sequence or Array of Characters

A *string* is traditionally a sequence of characters and is commonly used in computer programming for text data to allow programmers to represent a word, sentence, paragraph, or document. A string variable may be represented either as a literal constant, which has a fixed data value, or as some kind of variable, which can hold different text data values at different times during the execution of the computer software application. A variable's

W. Jackson, *JSON Quick Syntax Reference*, DOI 10.1007/978-1-4842-1863-1_6

elements can *mutated*; that is, the length of the string (number of characters) can be changed, as can the characters used to form the string. A string can also be fixed after it is created, in which case it is used as a constant. A string is generally understood to be a data type, as it is in JSON (as shown at the top of Figure 6-1). It is usually implemented as an array of characters that are represented as bytes (or words, depending on the character set used) and that stores a sequence of elements using a character-encoding format such as UTF-16.

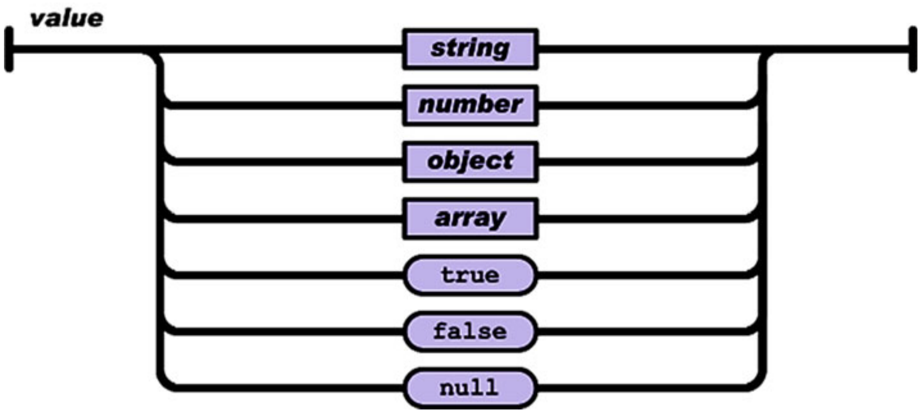


Figure 6-1. *JSON supports string, number, object, array, Boolean, and `null` data values*

When a string appears literally in the source code, it's known as a *string literal* or an *anonymous string*. In formal programming languages used for mathematical logic, or theoretical computer science, a string is defined as a finite sequence of symbols that are chosen from a character set commonly called an *alphabet*. Chapter 7 covers JSON strings, in particular, in greater detail, including Java and JavaScript string usage.

Number Value: Representing the World

A *number* is a mathematical object used to calculate, measure, count, estimate, or label what you see around you in the real world. Examples of a numeric value include what are known as the *natural* numbers, such as 1, 2, 3, and 4. The symbolic notation that represents a number is called a *numeral*, such as a Cardinal numeral (1, 2, 3, 4, 5) or a Roman numeral (I, II, III, IV, V). In addition to their use in counting, numerals are also used for labeling, like the numbers on your smartphone dialer; measuring, as in architectural blueprints; ordering, such as highway mile markers; estimating, as in repair estimates from an auto repair shop; and tracking, such as the Dewey Decimal System used a public libraries.

In everyday usage, the term *number* could refer to a plethora of different things, including a quantity, a symbol, a word, an estimate, or even a complex mathematical abstraction or calculation. In mathematics, the concept of a number has been extended over the centuries to include zero (no number, or nothing); negative numbers; rational numbers, such as fractions; real numbers, such as pi and the square root of any number; and complex numbers, which further extend the concept of real numbers by including, for instance, the square root of a negative number. Calculations using numbers are most commonly done using arithmetical operations, the most familiar being addition, subtraction, multiplication, division, and exponentiation.

The study or use of numbers is generally known as *arithmetic*. This same term can also refer to *number theory*, which is the study of the properties of these natural numbers. Chapter 8 covers JSON numbers, including Java and JavaScript number usage.

Boolean Value: True or False, On or Off, Yes or No

Booleans are named after George Boole, who first defined an algebraic system of logic in the mid-19th century. Boolean data values are intended to represent the truth values of logic with Boolean algebra and are often used to implement decision making.

In computer science, the Boolean data type has two opposing values, such as `On` and `Off`, `Yes` and `No`, or `Stop` and `Go`. These are usually quantified as `TRUE` and `FALSE`, at least in the Java and JavaScript languages. In programming languages that feature this Boolean data type, such as Java and JavaScript (and therefore JSON), comparison operators such as greater-than (`>`), NOT (`!`), and less-than (`<`) are usually defined as returning a Boolean value from a Boolean comparison operation. Conditional and Iterative programming constructs may also be defined to test Boolean-valued expressions and control the internal program logic using these Boolean values as switches to guide what you want your program logic to decide.

Boolean data types are primarily associated with the implementation of *conditional statements*, which allow different actions, and change the logic-control flow, depending on whether the programmer-encoded Boolean condition evaluates to `TRUE` or `FALSE`. Booleans are a specialized case of more general logical data types. It is important to note that logic does not always have to be Boolean in nature; however, Boolean logic is well-suited to programming languages, which is why it is implemented in all the most popular languages.

Most programming languages, including those that do not include Boolean data types, support Boolean algebraic operations, such as conjunction, using AND (`&`); disjunction, using OR (`|`); equivalence, using `=` and `==`; exclusion, using XOR (`!=`); and negation, using NOT (`!`).

Sometimes a Boolean variable is regarded and implemented as a numerical variable using a single binary digit (a bit that is 0 for false or 1 for true), which is logical, because a bit can store only two values. It is important to note that most implementations of Boolean values in computer languages are represented as a full word, rather than using a single bit; this is due to the way computers transfer blocks of information.

In some AI OOP languages, such as Alice and Smalltalk, the true and false values are created using separate classes, `True` and `False`. Thus there is no single Boolean data type.

To create a Boolean value in JSON, you use a lowercase `true` or `false` keyword without quotations, which would signify that it was a string value to the JSON parser rather than a Boolean value. Let's add a `"sold"` attribute to the `Car` object data definition created in Chapter 4, so you can add a Boolean value that tells you whether a car is still available for sale (`"sold":false`) or has already been sold (`"sold":true`). The new JSON data definition is as follows:

```
{ "name":       "carName",
  "speed":      15,
  "gear":       1,
  "drivetrain": 4,
  "direction":  "North",
  "color":      "Red",
  "fuel":       "Gas",
  "sold":       false    }
```

Next, let's take a look at the `null` value data type for undefined (empty) data values.

Null Values: A Placeholder for Future Data Values

The final type of data value you look at in this chapter is the `null` data value, which is often used to signify that a data field or data variable is undefined or currently empty (unused). The `null` character has a long history in computer science, starting as a zero-valued ASCII character in the original character set for 8-bit computing. This `null` data value is designated in some computer languages using `NULL` or even `NUL`; but in JSON, like `true` and `false`, it is written lowercase and without quotation marks (which again would turn the `null` value into a string value, `"null"`).

The `null` character is also frequently used in computer programming as a data-stream terminator or file terminator, a separator of data collections, or a filler. It's important to note that a `null` value, symbol, or character has no visual representation, much like the space character; but `null` uses a different ASCII (or other character set) code.

There is also a software design pattern using an `Object` with defined neutral behavior that is called the *null object software design pattern*; and in computer programming languages, there is also the concept of a *null pointer*. This can be designated as `NULL`, `Nil`, or `NONE` and is used to designate an uninitialized, undefined, empty, or meaningless data value.

The concept of a null value also extends to string values: there's a null string, in which the string value has a zero length (it's empty, or currently unused). There is also a null-terminated string, which is a string value filled with characters where the string's length is determined by the first `null` character that is encountered. This is equivalent to using `null` as a termination value; in this case, it's used to terminate the string, but a `null` value can also terminate a file or a data stream.

Some statically typed programming languages include a nullable type. This programming feature allows a data type to be set to the special value `NULL`. In SQL, `null` is used as a special marker and keyword for SQL data structures; the `null` value indicates that a particular data field has no current data value assigned.

There is also the concept of a *null device*, which is a special virtual computer device or console that discards all data written to it, kind of like a virtual black hole. Null devices are even used in real-world hardware, such as a null modem, which is a serial communications cable with special wiring to create null-device behavior.

Let's add a null value to the growing Car data object JSON definition to hold a value for the buyer of the car. If the "sold" attribute or parameter for the Car object is set to false, the car has no buyer (yet), and so the "buyer" attribute or parameter data field is set to null:

```
{
   "name":        "carName",
   "speed":       15,
   "gear":        1,
   "drivetrain": 4,
   "direction":  "North",
   "color":       "Red",
   "fuel":        "Gas",
   "sold":        false  // false Boolean value designates there is no buyer
   "buyer":       null
}
```

On the other hand, if the "sold" attribute or parameter is set to true, the car has a buyer, and so the "buyer" attribute or parameter data field is set to a string variable (covered in the next chapter). The Car data object JSON definition in this case changes to look like the following. Because the car has been sold ("sold": true), a buyer's name appears in the "buyer" data field:

```
{
   "name":        "carName",
   "speed":       15,
   "gear":        1,
   "drivetrain": 4,
   "direction":  "North",
   "color":       "Red",
   "fuel":        "Gas",
   "sold":        true
   "buyer":       "buyerName"  // null value replaced with a string value
}
```

Now that you have learned about Boolean and null value data types and how they are used in JSON data-definition structures, let's take a brief look of how they are represented in Java and JavaScript.

Java and JavaScript: Boolean and Null

Boolean values in Java are represented using the java.lang.Boolean class, which contains two static fields for the Boolean object: FALSE and TRUE. The Boolean class is a subclass of the java.lang.Object class. In JavaScript, Boolean values are a data type and use lowercase true and false values, just like JSON, which should not be a surprise to you at this point.

Both Java and JavaScript define `null` using lowercase as a *primitive*. A `null` primitive type is used to reset or empty an object and defines something that does not (yet) exist. JavaScript also has a type called `undefined` that is a keyword used in a way similar to null.

Summary

This chapter examined the different types of data values supported in JSON and how the Boolean and `null` data types are used in JavaScript, JSON, and Java. You looked at the basic details of string and number values, which are covered next, and you added Boolean and `null` values to the object definitions created in Chapter 4. The same format and syntax are used to add these values to a JSON array. Chapter 7 explores the different string data types and values allowed in JSON.

CHAPTER 7

JSON Character Values: Defining Strings in JSON

Chapter 6 looked at the two different types of special data types, Boolean and `null`; Chapter 4 covered JSON object data structures, the primary structures used to define objects; and Chapter 5 discussed the secondary array data structure used to define data arrays. This chapter looks at the string data values that define the text values supported in JSON data-definition structures. You saw how to use strings to some extent in Chapters 4 and 5. In this chapter, you go over the finer details of strings, Unicode character support, and some of the specifics of how strings, characters, and special characters are supported in the JSON format specification.

This chapter looks at the `string` data type supported in JSON and how string values are supported in JavaScript and Java, because these languages are used the most frequently with JSON. You see some examples of this `string` data type and how it can be used to extend the current `Car` object data definition to include some special characters such as quotation marks, forward and back slashes, and custom character codes, such as the one used for the copyright symbol.

JSON String: Unicode Character Support

As you know already, string values in JSON need to be delimited using *double quotation marks*, as do the parameter (or attribute) names used as keys in `key:value` pairs. This is because these data-value names, also known as *variable names*, *parameter names*, *attribute names*, or *characteristic names*, are strings containing the descriptive key to be used in JavaScript, Java, or other programming languages as the handle to reference the data value for that `key:value` pairing for that particular piece of data. In this chapter, that data value consists of a string of characters; in Chapter 8, it consists of different types of numbers, such as integers, real numbers, and exponents.

JSON string values are based on the popular Unicode universal character set. This set of predefined standard characters, symbols, and functions references characters using decimal numbers in the format `&#nnnn;` or, alternately, by using the hexadecimal format `&#xhhhh;` with the `x` preceding the hexadecimal always lowercase. You can see a list of the Unicode character set in table format, organized by function, at `https://en.wikipedia.org/wiki/List_of_Unicode_Characters`.

W. Jackson, *JSON Quick Syntax Reference*, DOI 10.1007/978-1-4842-1863-1_7

Figure 7-1 shows a diagram of JSON `string` data value type character support and guidelines, from the `json.org` web site. It also shows supported *escaped* control characters.

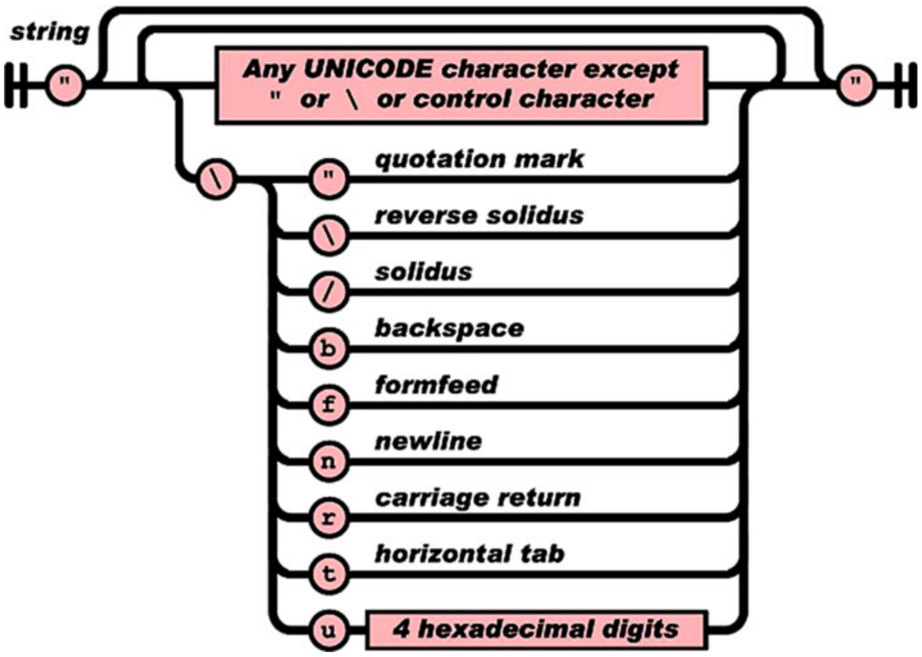


Figure 7-1. *JSON string values support Unicode characters, except escaped control characters*

A JSON `string` value may contain a sequence of zero or more Unicode characters. A single character can also be represented as a string containing only that character; this is how the `char` data type found in many programming languages can be simulated using a JSON `string` value.

JSON `string` values must be contained in *double quote* characters. You can *escape* special characters using the backslash character (\), called a *reverse solidus* in Figure 7-1; this backslash tells the JSON parser that a special character representation is about to be read. As you can see in the figure, a number of important textual functions are also supported.

The quotation marks used to delineate quotations need to be escaped themselves, because the JSON `string` value is contained in quotation marks; this allows the parser to understand that quotation marks internal to the JSON `string` value are not the ending quotes. Thus the JSON parser can "see" the end of the JSON `string` value as being the unescaped quotation marks, so string-value parsing is not ended (aborted) prematurely.

The backslash is also escaped using itself—so, adding a backslash character to a JSON string value requires the use of *two* backslashes. A *forward slash* (`/`), or *solidus*, requires a backslash and then a forward slash.

The backspace character is represented by the letter `b` for *backspace* and is escaped by using the `\b` character sequence. This is more of a printer-related character, as is the form-feed character, which is represented by the letter `f` and is escaped using `\f`.

The newline is a typewriter legacy as well as a printing-related character. It is represented by the letter `n` and advances the printer or text editor to the next line. It is escaped using `\n`. This newline is closely related to the carriage return.

The carriage return is another typewriter legacy and print-related operation represented by the letter `c`. It is escaped using `\c`.

The tab character places even amounts of space between text and is meant to provide precise text alignment. It is represented by the letter `t` and escaped using `\t`.

Finally, there's support for the custom Unicode character set, which allows you to access specialized characters such as the trademark, registered, and copyright symbols. The escaped sequence is `\u` followed by the four hexadecimal digits of any special Unicode character you may wish to use in your character string.

Background of String Values

A *string* is traditionally a sequence of characters and is commonly used in computer programming for text data to allow programmers to represent a word, sentence, paragraph, or document. A string variable may be represented either as a literal constant, which is fixed in its data value, or as a variable, which can hold different text data values at different times during the execution of the computer software application. A variable's elements can be *mutated*; that is, the length of the string (number of characters) can be changed, as can the characters used to form the string value. A string can also be fixed after it is created, in which case it is used as a constant. A string is generally understood to be a data type in computer programming languages, as it is in JSON (as shown at upper left in Figure 7-1). It is usually implemented as an array of characters that are represented as bytes (or words, depending on the character set used) and that stores a sequence of elements using a character-encoding format such as UTF-16.

When a string appears literally in the source code, it's known as a *string literal* or an *anonymous string*. In formal programming languages used for mathematical logic, or theoretical computer science, a string is defined as a finite sequence of symbols that are chosen from a character set commonly called an *alphabet*.

Escaping Control Characters: JSON Examples

Let's take a look at some of the more advanced forms of JSON string values; you have used simple string values in several other chapters already. This section goes through some examples that add new data fields to the `Car` object data definition: you add a nickname using quotes, a directory path (folder filename path), and a copyright symbol, to see how to escape characters in JSON string values so that you can add control characters and special symbols (nonstandard characters).

First, let's look at how you add a nickname to the "buyer" attribute by escaping the quotation marks using the backslash character. Let's say the buyer name is Johnny "RocketMan" Starship. If you just used quotes in the string value, a JSON parser would read it as "Johnny" and then throw an error. You need to escape the quotes using a backslash, which makes the JSON object data definition look like the following:

```
{   "name":        "carName",
    "speed":       15,
    "gear":        1,
    "drivetrain":  4,
    "direction":   "North",
    "color":       "Red",
    "fuel":        "Gas",
    "sold":        true
    "buyer":       "Johnny \" RocketMan \" Starship"   }
```

Next, let's see how you add a "folder" attribute. Folders have backslash characters between the drive specifier, directory name, subdirectory names, and filename, and you implement this by escaping each backslash character using another backslash character. Let's say the folder path to the buyer name is C:\Buyers\Name. You need to escape the backslashes using a second backslash, which makes the new ten-attribute JSON object data definition as follows:

```
{   "name":        "carName",
    "speed":       15,
    "gear":        1,
    "drivetrain":  4,
    "direction":   "North",
    "color":       "Red",
    "fuel":        "Gas",
    "sold":        true
    "buyer":       "Johnny \" RocketMan \" Starship"
    "folder":      "C:\\Buyers\\Name" // Equates to C:\Buyers\Name folder   }
```

Finally, let's look at how to add a "copyright" attribute and symbol. You escape u (Unicode) using a backslash and add the copyright character code 00A9 immediately thereafter. This makes a new 11-attribute JSON object data definition:

```
{   "name":        "carName",
    "speed":       15,
    "gear":        1,
    "drivetrain":  4,
    "direction":   "North",
    "color":       "Red",
    "fuel":        "Gas",
    "sold":        true
    "buyer":       "Johnny \" RocketMan \" Starship"
    "folder":      "C:\\Buyers\\Names"
    "copyright":   "\u00A9 2016, All Rights Reserved." // Equates to ©2016   }
```

The next section discussed how string values are handled in Java and JavaScript, to make sure you have a comprehensive overview of how string data is handled using two of the most popular OOP languages used in conjunction with JSON.

Java and JavaScript: Using String Values

String values in Java are represented using the `java.lang.String` class, which creates a `String Object` using the `Char` (`Character`) data type. The `String` class is a subclass of the `java.lang.Object` class. In JavaScript, `String` values are primitive data types and use either the single-quote character to contain other characters or, more commonly, double quotes, like JSON. Just as in Java, JavaScript can also define strings as objects using the `String()` function. Let's take a look at these two different approaches for defining strings of characters.

Java String Values: Java's String Class and Object

The Java programming language builds a `String Object` by using individual characters, which is a powerful and flexible approach. The `java.lang.String` class can be customized if necessary by subclassing it, thereby making it a superclass. The `java.lang.String` class is a subclass of `java.lang.Object` and is a public final class that implements three Java interfaces: `Serializable`, `CharSequence`, and `Comparable<String>`. Java `Strings` are constants, and therefore their values cannot be changed after they are created.

Strings in Java can be created in several ways. For instance, you can declare a string using the `String` classname, the string constant name, and a data value in quotes, like this:

```
String stringName = "Jon";
```

Or you can declare it using the `char` primitive, an array, and the Java `new` keyword, using two lines of Java code:

```
char stringData[] = {'J', 'o', 'n'};
String stringName = new String(stringData);
```

The `String` class has a plethora of methods for manipulating strings, examining string contents, concatenating strings, measuring strings, and the like. If you want to look into this more closely, see `https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/String.html`.

JavaScript String Values: Primitives and Objects

JavaScript can define strings as primitive data values created from literals, and can also define strings as objects by using the `String()` function in conjunction with the `new` keyword. Let's look at both of these approaches.

JavaScript String Primitives: Using Var to Create a Literal

JavaScript strings contain a sequence of characters in either single quotes or double quotes. You can use quotes in a JavaScript string as long as they don't match the quotes that contain the string. Most JavaScript programmers use double quotes on the outside of the string value and single quotes in the string value, like this:

```
var sampleStringValue = "This is a string with an 'inside quotation' in it";
```

You can find the length of a JavaScript string using its `length` property, like this:

```
var sampleStringLength = sampleStringValue.length;
```

The same escape-character sequences relating to Figure 7-1 work in JavaScript exactly the same way they work in JSON, not surprisingly.

JavaScript String Objects: Using String() with the new Keyword

JavaScript strings can also be created as objects by using the `String()` function and `new` keyword, although this is not recommended because it is more memory- and processor-intensive. The `String()` function parameter area contains your sequence of characters, again in either single quotes or double quotes. As stated earlier, most JavaScript programmers use double quotes on the outside of the string value, but let's mix it up and reverse that convention in this example (which also works):

```
var sampleStringValue = new String('String with "inside quotation" in it');
```

Strings are an important data type, so be sure to master them!

Summary

This chapter looked at how string values are represented in JSON and how the `String` data type is used in JavaScript and Java. You learned about the basics of string values, and you added attributes to the `Car` object JSON data definition that included custom escaped character string values. You saw how to use quotation marks, directory backslash separators, and special character (nonstandard) characters in string data values. The same format and syntax are used to add these values to your JSON array structures. Chapter 8 discusses the different number data types and numeric value representations allowed in JSON.

CHAPTER 8

JSON Numeric Values: Defining Numbers in JSON

Chapter 7 looked at the `string` data type; now let's examine the `number` data value in JSON. It lets you define a wide range of numeric data values that can be supported in your JSON data-definition structures. You learned some things about numbers in Chapters 4 and 5; this chapter goes over some of the finer details, including different types of numbers and specifics of how integers, fractions, decimals, and exponents are supported in the JSON format specification.

This chapter looks at the `number` data type supported in JSON. These numeric data values are supported in the same ways and formats in JavaScript as well as Java. You see examples of JSON number data and how it's used to extend the current `Car` object.

Number Types: Integer, Real, Exponential

As you learned in Chapter 6, a wide range of number representations are used in both everyday life as well as complex mathematics, physics, and similar advanced studies. JSON uses three main types: whole numbers, called *integers*; fractional numbers, called *real numbers* or *decimal numbers*; and exponential numbers, which use the `E` or `e` designation to tell you where the decimal point goes in fractional numbers that are extremely large or small in their magnitude. This chapter shows you how these are represented in JSON. These three types of numbers should allow you to represent any numeric value you need in your JSON application development.

JSON Number: Wide Ranging Numerics

The diagram in Figure 8-1, from `json.org`, shows the JSON `number` data value type numeric support and guidelines along with supported negative (minus sign), decimal (0 and fractional or real numbers using a decimal point), and exponential (`E` or `e` notation with a plus or minus sign) number support. This chapter explains how to traverse this diagram to create the different types of number values in JSON.

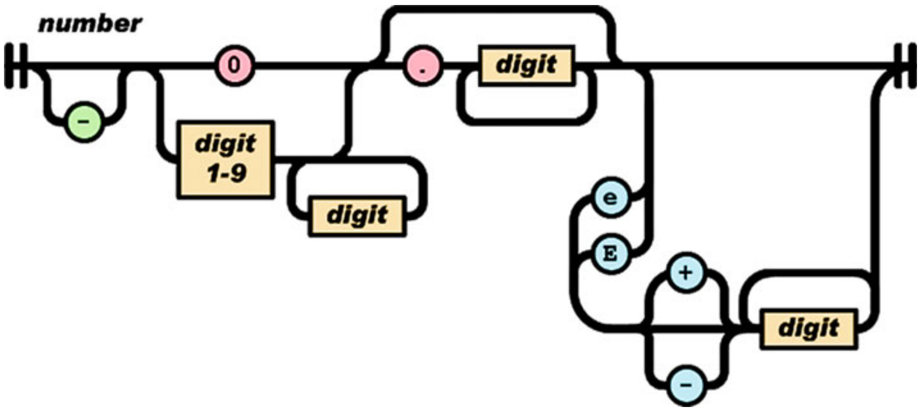


Figure 8-1. *JSON supports integers, negative numbers, real numbers, and exponential numbers*

Let's start with positive integers—the most common numbers. You then progress to negative integer values, positive real (decimal or fractional) numbers, negative real numbers, and finally exponential numbers.

Positive Integers: Positive Whole Number Values

The path through the JSON number diagram is shown in red in Figure 8-2. It simply involves adding whole-number digits together, much as you did for the `Car` object `"price"` data field (or characteristic, or attribute) in Chapter 5.

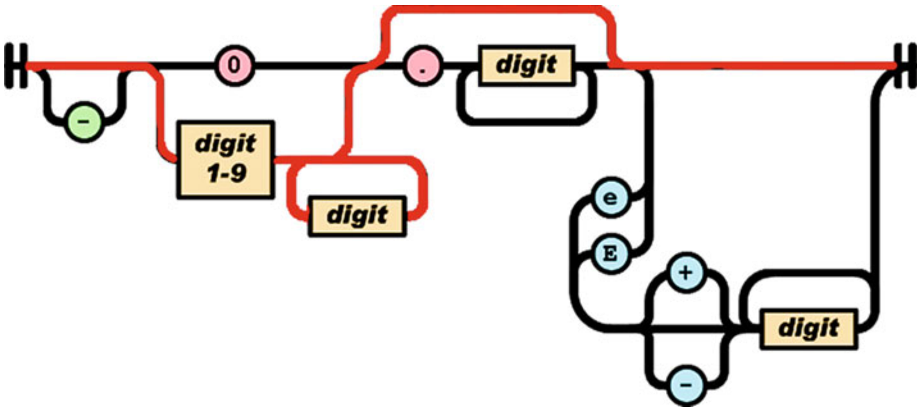


Figure 8-2. *Creating positive integer value using the JSON number value diagram*

Here is the JSON array from Chapter 5 so that you can revisit using positive integer number values:

```
var carInventory = [
    { "carName":"Toyota", "model":"Prius", "color":"Green", "price":24995 },
    { "carName":"Nissan", "model":"300ZX", "color":"Brown", "price":21795 },
    { "carName":"Dodge",  "model":"Viper", "color":"White", "price":26495 },
    { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "price":27895 }
]
```

Notice that there are no commas: formatting is done in the code, not in the JSON data array or data object. As Figure 8-2 shows, you add digits as needed; a plus sign is not needed to designate a positive value. Next, let's look at negative integer (whole) number values, which are created very similarly using a slightly different path through this diagram.

Negative Integers: Negative Whole Number Values

The path through the JSON number diagram to create a negative number is shown in red in Figure 8-3. You add whole-number digits together, but with a minus sign designator appended onto the front of the number data value (which you are familiar with as a programmer, as well as from school).

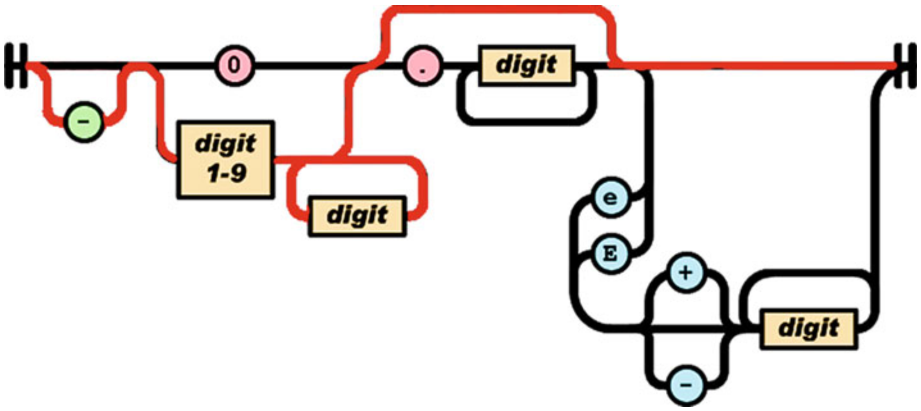


Figure 8-3. *Creating negative integer value using the JSON number value diagram*

If you used a negative data value to designate the cost (how much you would need to pay) for the Car object "price" attribute in the example in Chapter 5, the array of inventory data values would look like the following, replacing “price” with "cost":

```
var carInventory = [
   { "carName":"Toyota", "model":"Prius", "color":"Green", "cost":-24995 },
   { "carName":"Nissan", "model":"300ZX", "color":"Brown", "cost":-21795 },
   { "carName":"Dodge",  "model":"Viper", "color":"White", "cost":-26495 },
   { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "cost":-27895 }
]
```

Now, let’s look at partial or fractional numbers that use decimal components to hold fractions of whole numbers. These are commonly called *real numbers* or *floating-point* numbers in the popular computer programming languages.

Positive Real Number Values: Positive Fractions

The path through the JSON number diagram to create a positive real number is shown in red in Figure 8-4. You add a 0 if the fractional value is less than 1, and then a decimal point and the digit(s) that specify the fractional component.

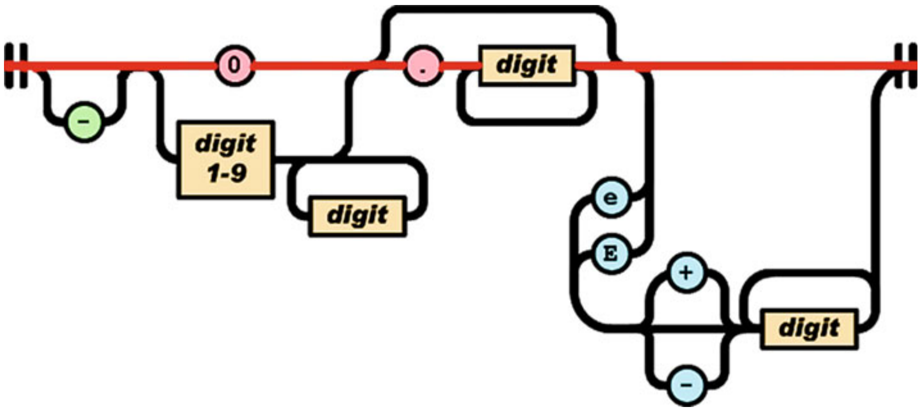


Figure 8-4. *Creating a positive fractional number value that is less than 1*

As an example of using a positive real number that is less than 1, let's create a "gas" attribute that tells how much gas is currently in a car's gas tank. This is a real number that is greater than 0 but less than 1 and that essentially represents a percentage on the gas gauge: 0.33 is a third of a tank, 0.75 is three quarters, 0.25 is a quarter, and so on. A JSON array data for this usage scenario looks like the following:

```
var carInventory = [
   { "carName":"Toyota", "model":"Prius", "color":"Green", "gas":0.33 },
   { "carName":"Nissan", "model":"300ZX", "color":"Brown", "gas":0.75 },
   { "carName":"Dodge",  "model":"Viper", "color":"White", "gas":0.99 },
   { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "gas":0.25 }
]
```

If the fractional value is greater than 1, you take a different path through the diagram, as shown in Figure 8-5. In this case you add one or more non-zero digits before the decimal point. The value can be much greater than 1, such as 12345.6789, if needed for a real number data representation.

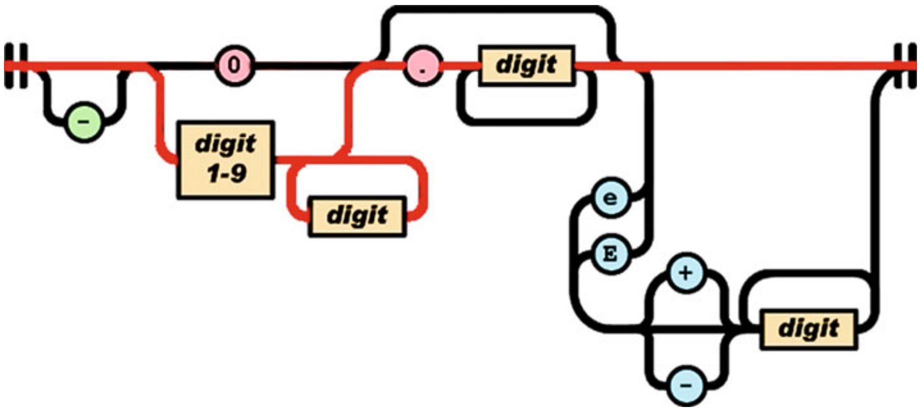


Figure 8-5. *Creating positive fractional number values greater than 1*

For example, let's create a "tax" attribute that tells how much tax will be charged in the municipality where that car is being sold. This is a real number that is greater than 1 but less than 2, representing a percentage of the purchase price (represented as 1.00) that is added: 1.08 is an 8% sales tax, 1.06 is a 6% sales tax, 1.07 is a 7% sales tax, and so on. Here is a JSON array for this scenario:

```
var carInventory = [
   { "carName":"Toyota", "model":"Prius", "color":"Green", "tax":1.08 },
   { "carName":"Nissan", "model":"300ZX", "color":"Brown", "tax":1.06 },
   { "carName":"Dodge",  "model":"Viper", "color":"White", "tax":1.09 },
   { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "tax":1.07 }
]
```

Next, let's look at negative real (fractional or decimal) number values, which are created using a slightly different path through the JSON diagram.

Negative Real Number Values: Negative Fractions

The path through the JSON number diagram to create a negative real number value less than 1 is shown in red in Figure 8-6. You add a minus sign to the front of the fractional (less than 1) number representation created in Figure 8-4.

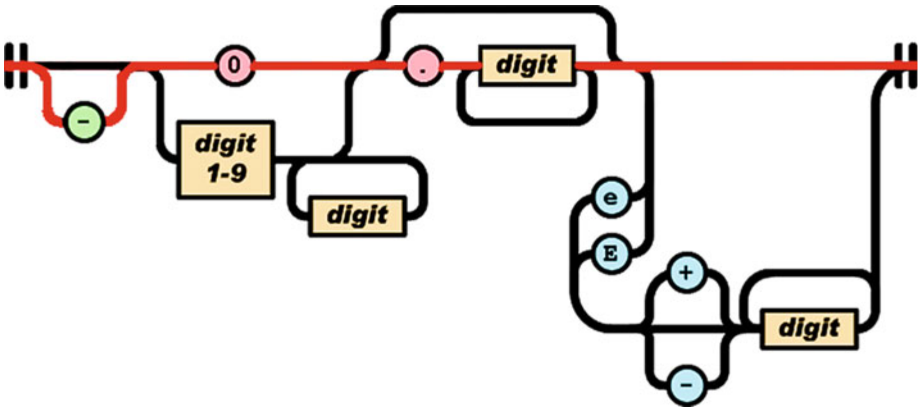


Figure 8-6. *Creating a negative real number value less than 1*

For example, let's create a "fee" attribute that tells what percentage of the car's price you need to give to the person who sells the car for the dealership, as a commission. This is a negative real number that is less than 0. So, -0.12 is a 12% commission, -0.09 is a 9% commission, -0.11 is an 11% commission, -0.08 is an 8% commission, and so forth. The JSON data array structure is as follows:

```
var carInventory = [
   { "carName":"Toyota", "model":"Prius", "color":"Green", "fee":-0.12 },
   { "carName":"Nissan", "model":"300ZX", "color":"Brown", "fee":-0.09 },
   { "carName":"Dodge",  "model":"Viper", "color":"White", "fee":-0.11 },
   { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "fee":-0.08 }
]
```

You can also create negative fractional numbers that are greater than 1. This is done by using the path through the JSON number diagram that is shown in Figure 8-7. This allows you to add a minus sign, loop as many integers before the decimal point as you need to represent your number, and then loop as many integers after the decimal point as you need for the numeric precision of your fractional number representation.

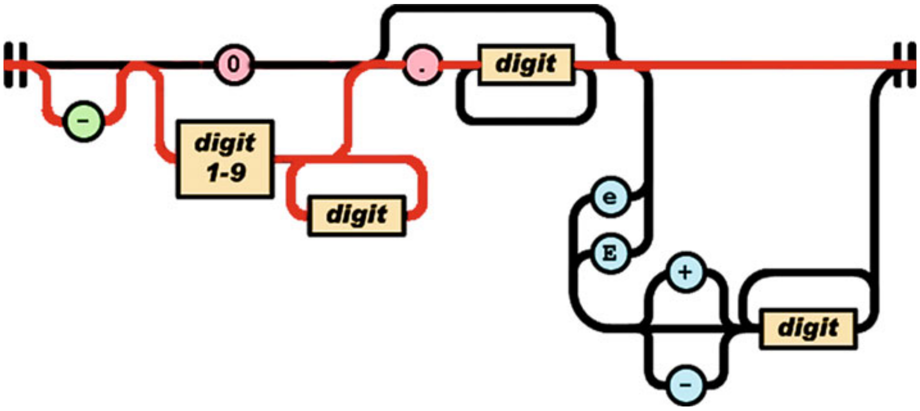


Figure 8-7. *Creating negative real number values greater than 1*

For example, let's create a `"loss"` attribute that is a negative real number greater than 0, representing the sales price plus the percentage of the commission given to the sales team. So, -1.12 is a reduction of 100% of the car's price plus a reduction of a 12% commission, -1.09 is a reduction of 100% of the car's price plus a reduction of a 9% commission, -1.11 is a reduction of 100% of the car's price plus a reduction of an 11% commission, and so forth. The JSON data array structure is as follows:

```
var carInventory = [
   { "carName":"Toyota", "model":"Prius", "color":"Green", "loss":-1.12 },
   { "carName":"Nissan", "model":"300ZX", "color":"Brown", "loss":-1.09 },
   { "carName":"Dodge",  "model":"Viper", "color":"White", "loss":-1.11 },
   { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "loss":-1.08 }
]
```

Next, let's look at scientific notation and how to use exponential notation.

Exponential Number Values: Scientific Notation

Scientific notation, sometimes called exponential notation, is closely related to the fractional (floating-point or decimal) number representation you looked at in the previous sections. In scientific notation, all numbers are written in the form *n* **× 10**e**.** That represents a number *n* times 10 raised to the power of the exponential factor e, where the exponent e is an integer and the coefficient *n* is any real number. The coefficient is called the *significand* or the *mantissa*. If the number needs to be negative, then a minus sign precedes the *n*, as in ordinary decimal notation.

Scientific notation is a way of expressing number values that are too big or too small to be conveniently written in a decimal form—that is, they take up too many spaces or columns. Exponential notation is most commonly used by scientists, mathematicians, and engineers. On scientific calculators, it is known as the SCI display mode and uses the EXP, exp, ex, e, eex, E, EX, and EEX keys. Table 8-1 shows some common decimal-to-exponential conversions.

Table 8-1. *Integers and Decimal Numbers Converted into Scientific or Exponential Notation*

Integer or Decimal Representation	Scientific and Exponential Representation
7	7 x 10^1 or $7e10^1$ or $7E10^1$
-9	-9 x 10^1 or $-9E^1$ or: $-9e^1$
500	$5e^2$ or $5E^2$
5,432.109	$5.432109E^3$ or $5.432109e^3$
-19,876,543.21	$-1.987654321E^7$ or $-1.987654321e^7$
0.0004321	$4.321E^{-4}$ or $4.321e^{-4}$
-0.00000000098765	$-9.8765E^{-10}$ or $-9.8765e^{-10}$

Let's take a closer look at how these exponential notation representations are formed using the JSON number diagram. Then you'll be ready to create JSON objects and JSON arrays for any JSON-compliant applications.

Integer Exponential Number Values: Using Positive Exponents

To create a positive integer value using exponential notation, follow the path shown in Figure 8-8 to create the fractional component, add an e or E, and then add an integer magnitude value.

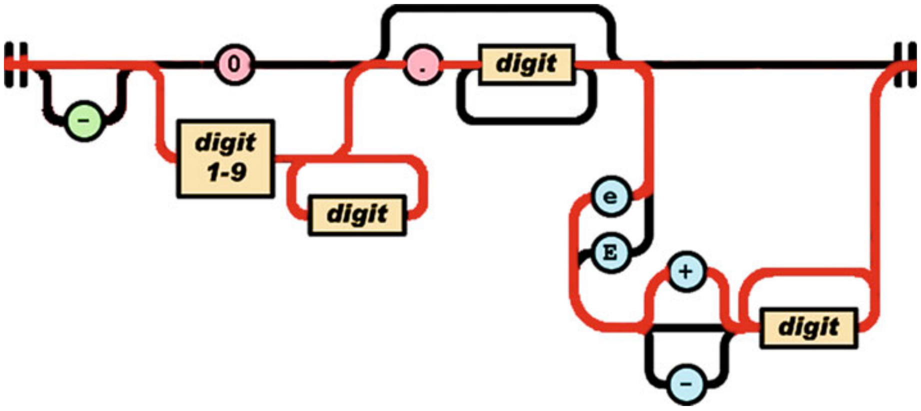


Figure 8-8. *Creating a positive integer exponential number*

If you use an exponential number value to designate the price (how much you need to pay) for the Car object's "price" attribute in the example, then the array of inventory data values looks like the following:

```
var carInventory = [
{ "carName":"Toyota", "model":"Prius", "color":"Green", "price":2.4995e4 },
{ "carName":"Nissan", "model":"300ZX", "color":"Brown", "price":2.1795e4 },
{ "carName":"Dodge",  "model":"Viper", "color":"White", "price":2.6495e4 },
{ "carName":"Chevy",  "model":"Camaro", "color":"Blue", "price":2.7895e4 }]
```

Next, let's go in the other direction and look at how to create small fractional numbers by using negative exponential magnitude values. After that, you'll see how to create negative exponential number values in JSON.

Fractional Exponential Number Values: Negative Exponent

To create a positive fractional value by using exponential notation, follow the path shown in Figure 8-9. The first decimal portion creates the fractional component; then add the e (or E) and a negative integer to establish the magnitude value—that is, how many tens (decimal places, or orders of magnitude) to use to set the magnitude of how small your fractional exponential value should be.

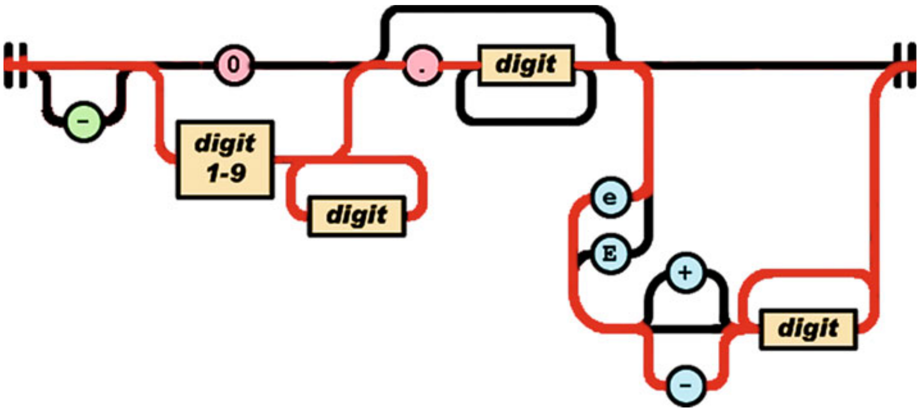


Figure 8-9. *Creating a positive fractional exponential number*

As an example, let's recreate a "gas" attribute that tells how much gas is in a car's gas tank using exponential notation instead of decimal notation. The JSON array data looks like this:

```
var carInventory = [
   { "carName":"Toyota", "model":"Prius", "color":"Green", "gas":3.3e-1 },
   { "carName":"Nissan", "model":"300ZX", "color":"Brown", "gas":7.5e-1 },
   { "carName":"Dodge",  "model":"Viper", "color":"White", "gas":9.9e-1 },
   { "carName":"Chevy",  "model":"Camaro", "color":"Blue", "gas":2.5e-1 }]
```

Now, let's see how to define negative exponential values.

Negative Integer Exponential Number Values: Positive Exponent

To create a negative integer value using exponential notation, follow the path shown in Figure 8-10. You add a minus sign to define the negative overall value for the number, then create the fractional (decimal) component, add the e (or E), and finally add the positive integer magnitude value that defines the positive magnitude (how large the negative number is) for the number value.

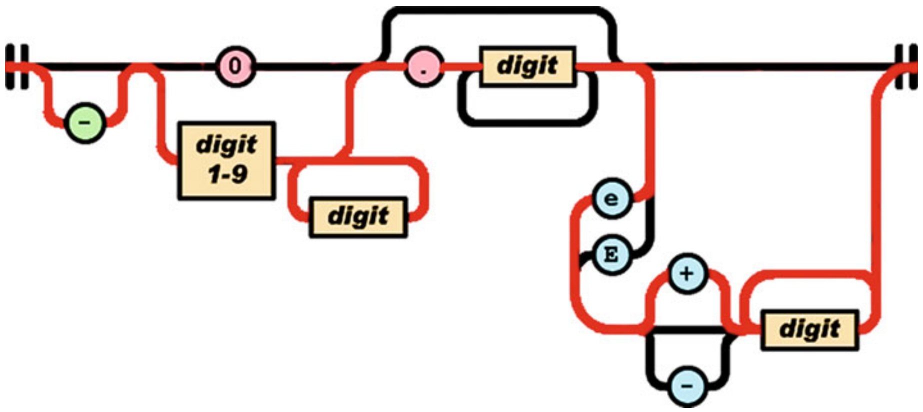


Figure 8-10. *Creating a negative integer exponential number*

There isn't a good example that fits the `Car` object definition paradigm. Instead, let's say you were at the Auto Show in Las Vegas and lost $127,346,000 playing poker. The JSON object to define your losses and where you lost them uses the following JSON object data-definition syntax:

```
var moneyLostAtTradeShow = { "city": "Las Vegas", "losses": -1.27346e9 }
```

Negative Fractional Exponential Numbers: Negative Exponent

To create a negative fractional value by using exponential notation, follow the path shown in Figure 8-11. First add a minus sign to designate the value as being negative, then add the decimal portion that creates the fractional component, add the e (or E), and then add the negative integer to establish the magnitude of smallness—that is, how many tens (decimal places, or orders of magnitude) to use to set the magnitude of how small your fractional exponential value should be.

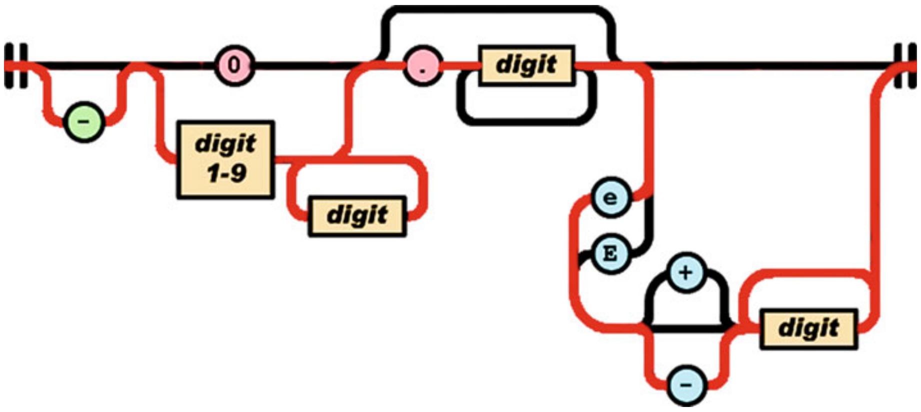


Figure 8-11. *Creating a negative fractional exponential number*

Again, there is no good example using the `Car` object definition paradigm. Suppose that while you are at the Auto Show in Las Vegas, the chances that you will lose $127,346,000 playing poker are -1 million to 1. The JSON object to define the percentage loss and where you lost it is as follows:

```
var ChanceOfLossAtTradeShow = { "city": "Las Vegas", "pctChance": -1.0e-9 }
```

Now you have taken every possible path through the JSON number value diagram. You should be able to define any numeric value that your JSON applications require.

Summary

This chapter looked at how `number` values are represented in JSON, which is also how the `Number` data type is used in JavaScript and Java. You learned the basics of `number` values, and you created attributes for the `Car` object JSON data definition that added different number values to the JSON array data definitions from Chapter 5. You saw how to use negative (minus sign), 0, decimal (fractional), and exponential designators with `number` data values. The same format and syntax are used to add these values to JSON object structures.

I hope you have found the JSON syntax overview in this book enlightening and helpful for your future JSON object and array data-definition endeavors!

APPENDIX A

NetBeans 8.1: Setting Up a JSON Integrated Development IDE

In this first appendix, let's put together your foundation for a highly professional, JSON-friendly, NetBeans 8.1 integrated development environment (IDE). Your development workstation is the most important combination of PC hardware and software, allowing you to reach your goal of JSON-compatible application development. This appendix considers your hardware needs and the software infrastructure to put together a professional, well-rounded JSON workstation that gives you a bunch of arrows in your software development quiver right off the bat. You will then have everything you need when you're reading the book's chapters, no matter what type of JSON application you decide that you want to develop for your end users!

All readers of this book should be developing with identical JSON application software development environments, because everything you learn over the course of this book needs to be able to be experienced equally by everyone. Appendix B outlines all the steps to put together an Eclipse Mars IDE based JSON development workstation, and Appendix C does the same for IntelliJ IDEA. These three IDE-customized appendixes get all the tedious setup tasks out of the way. If you already have your workstation configured, you can proceed to Chapter 2 for an overview of JSON; or, if you are already familiar with JSON, you're ready to read the rest of the book.

In this appendix, you learn where to download and how to install several of the most impressive, professional, open source software packages on the face of this planet. You are about to max out your JSON, HTML5, CSS3, and JavaScript (JS) development workstation, so hold on tight and enjoy this virtual download ride!

Creating a JSON Development Workstation

The first thing you'll do after taking a look at hardware requirements is to download and install the entire Java software development kit (SDK), which Oracle calls Java SE 8 Java Development Kit (JDK). The NetBeans 8.1 IDE uses the Java 8 Standard Edition (SE).

W. Jackson, *JSON Quick Syntax Reference*, DOI 10.1007/978-1-4842-1863-1

The second thing you'll download and install is the NetBeans 8.1 IDE that you get from `www.netbeans.org`. The NetBeans 8.1 IDE allows you to develop JSON-compatible applications using all the popular programming languages, including C, C++, Java, PHP, Python, JavaFX, Ruby, HTML5, CSS3, ECMAScript, and JavaScript.

After your JSON application development environment is set up, you can then download and install new media asset development tools, if you wish. These are used in conjunction with NetBeans 8.1 for things such as image editing (GIMP) and nonlinear digital video editing (Lightworks); special effects (Fusion); digital audio sweetening, or editing (Audacity); i3D modeling, rendering, and animation (Blender); digital illustration (Inkscape); and business productivity (OpenOffice).

This appendix will take your development to an all-new level, showing you how to create the media development and programming workstation that will run your business. All the software development tools you'll download and install come close to matching all the primary feature sets of expensive paid software packages, such as those from Apple (Final Cut Pro), Autodesk (3D Studio Max), Adobe (Photoshop, Illustrator, After Effects), and Avid (ProTools), and all at *zero* cost to your production company!

Open source software is free to download, install, and upgrade and is continually adding features. It's becoming more and more like professional software every day. You will be amazed at how professional open source software packages have become over the last decade or so.

Hardware Foundation

Because in this appendix you put together the foundation for the JSON-capable application development workstation you use throughout this book, I want to take a moment to review NetBeans 8.1's JSON development workstation hardware requirements. This is a factor that will influence your development performance (speed). This is clearly as important as the software itself, because hardware is what is actually running the software package's algorithms.

Minimum requirements for the NetBeans 8.1 IDE include 512MB of memory, 750MB of hard disk space, and an XGA (1024×768) display. Next, let's discuss what you need to make the NetBeans 8 JSON IDE usable. Start with upgrading the 1024×768 XGA display to an HDTV (1920×1080 at 120FPS refresh rate) or UHD (4096×2160 at 120FPS refresh rate) widescreen display. These are now affordable and give you 4 to 16 times the display real estate of an XGA display. HDTVs are now $250 to $500, and UHDTV displays are under $1,000.

I recommend using, at a bare minimum, the Intel i7 quad-core processor or the AMD 64-bit octa-core processor. Install at least 8GB of DDR3-1600 memory. I'm using a 64-bit, octa-core AMD 8350, with 16GB of DDR3-1600. Intel also has a hexa-core i7 processor. This would be the equivalent of having 12 cores, because each i7 core can host two threads. Similarly, an i7 quad-core should look like 8 cores to your 64-bit operating system's thread-scheduling algorithm.

There are also high-speed DDR3-1800 as well as DDR3-2133 clock-speed memory module components available. A high number signifies fast memory-access speeds. To calculate actual megahertz speeds at which the memory is cycling, divide the number by 4 (1333 = 333Mhz, 1600 = 400Mhz, 1800 = 450Mhz, 2133 = 533Mhz). Memory-access speed is a massive workstation performance factor, because your processor is usually bottlenecked by the speed at which processor cores can access the data (in memory) that that processor core needs to process.

With high-speed processing and memory access going on in the workstation while it is operating, it's extremely important to keep everything cool so that you do not experience thermal problems. I recommend using a wide, full-tower enclosure with 120mm or 200mm cooling fans (one or two at least), as well as a captive liquid-induction cooling fan on the CPU. It is important to note that the cooler the system runs, the faster it can run, and the longer it will last, so load the workstation up with lots of silent high-speed fans!

If you really want maximum performance, install a solid state disk (SSD) drive as the primary disk drive from which your applications and operating system software can load. Use legacy HDD hardware for your `D:\` hard drive for slower data storage (long-term). Put your current project files on the SSD.

As far as OS goes, I am using a 64-bit Windows 8 operating system that is fairly memory efficient. The Linux 64-bit OS is extremely memory efficient. I recommend using any 64-bit OS so you can address more than 3.24GB of system memory.

Open Source Software

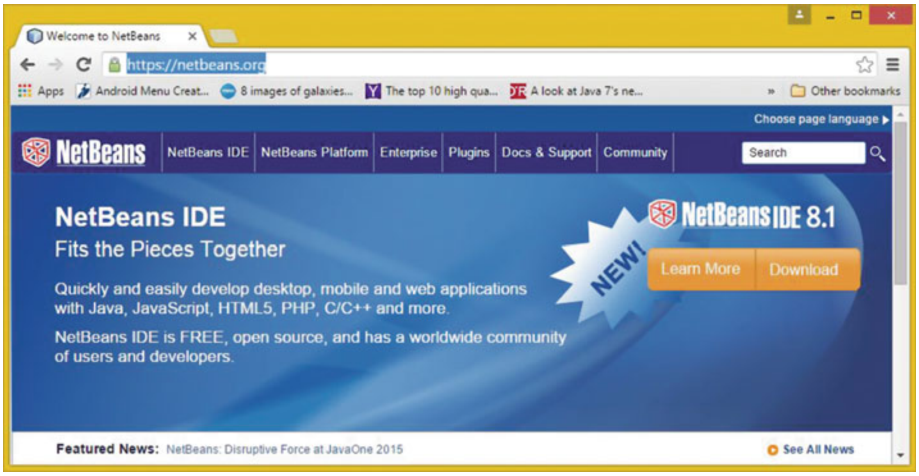
To create a well-rounded, professional JSON application development workstation, you'll install all the primary genres of open source software. First you'll install Java SE 8 and NetBeans 8.1. I will also show you how to download GIMP, Lightworks, Fusion, Blender3D, and Audacity, which are also all open source software packages, in case your JSON applications will use a graphical front end. I also recommend other free software at the end of the appendix so you can put together the major production workstation you have always dreamed of.

Open source software is approaching the level of professionalism of paid development software packages that cost thousands of dollars each. Using open source software packages like Java 8, NetBeans 8, Blender, GIMP, Audacity, Lightworks, Fusion, OpenOffice, and others, you can put together a free application development workstation and rival paid software workstations.

If you just purchased a new JSON development workstation PC and are going to put together the entire development software suite from scratch, this appendix goes through the entire work process.

Java 8: Installing the Foundation for NetBeans 8.1

The first thing to do is visit the NetBeans web site at `www.netbeans.org` to find out what you need to run this IDE. Click the Download button, shown at right in Figure A-1.



NetBeans IDE

Fits the Pieces Together

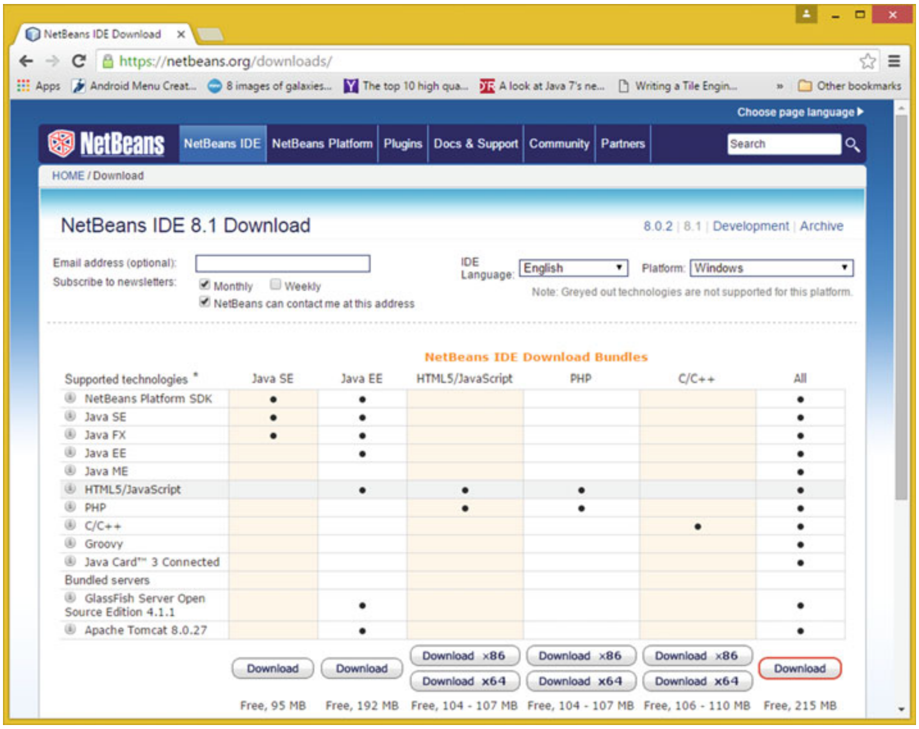
Quickly and easily develop desktop, mobile and web applications with Java, JavaScript, HTML5, PHP, C/C++ and more.

NetBeans IDE is FREE, open source, and has a worldwide community of users and developers.

Featured News: NetBeans: Disruptive Force at JavaOne 2015

Figure A-1. *Go to `netbeans.org`, and click Download*

As you can see, there are nine different download options to consider; six support JavaScript, which JSON is based on. I suggest the All version, which supports all the popular programming languages, all of which JSON works with. If you are wondering why some of these downloads offer 32-bit and 64-bit versions and some do not, as you can see at the bottom of Figure A-2, it is because the ones with both versions have been *precompiled*, whereas the other three require a Java 8 JDK to be installed. This tells you that if you want to use the All version, so that any programming languages you want to use JSON with will be supported, you have to first install Java SE 8.



NetBeans IDE 8.1 Download

8.0.2 | 8.1 | Development | Archive

Email address (optional):
Subscribe to newsletters: ☑ Monthly ☐ Weekly
☑ NetBeans can contact me at this address

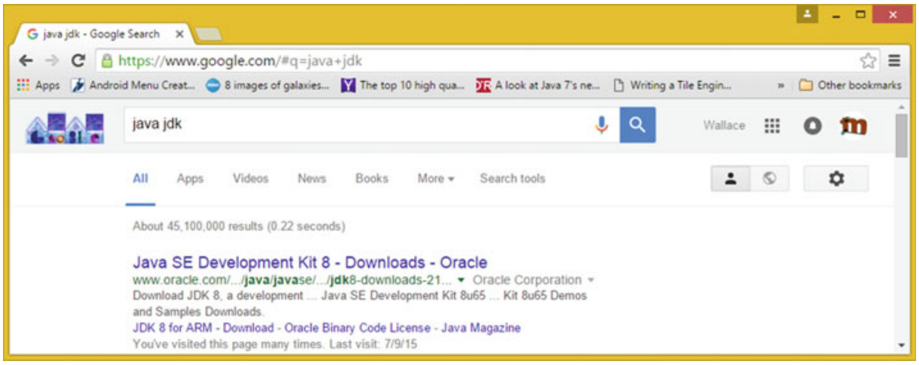
IDE Language: English
Platform: Windows
Note: Greyed out technologies are not supported for this platform.

NetBeans IDE Download Bundles

Supported technologies *	Java SE	Java EE	HTML5/JavaScript	PHP	C/C++	All
NetBeans Platform SDK	●	●				●
Java SE	●	●				●
Java FX	●	●				●
Java EE		●				●
Java ME						●
HTML5/JavaScript		●	●	●		●
PHP			●	●		●
C/C++					●	●
Groovy						●
Java Card™ 3 Connected						●
Bundled servers						
GlassFish Server Open Source Edition 4.1.1		●				●
Apache Tomcat 8.0.27		●				●
	Download	Download	Download x86 Download x64	Download x86 Download x64	Download x86 Download x64	Download
	Free, 95 MB	Free, 192 MB	Free, 104 - 107 MB	Free, 104 - 107 MB	Free, 106 - 110 MB	Free, 215 MB

Figure A-2. *Download one of the HTML5/JavaScript IDE versions*

Open Google Chrome, and Google “Java JDK”, as is shown in Figure A-3. Look for the Java SE Development Kit 8 - Downloads search result, and click it to open Oracle Java 8.

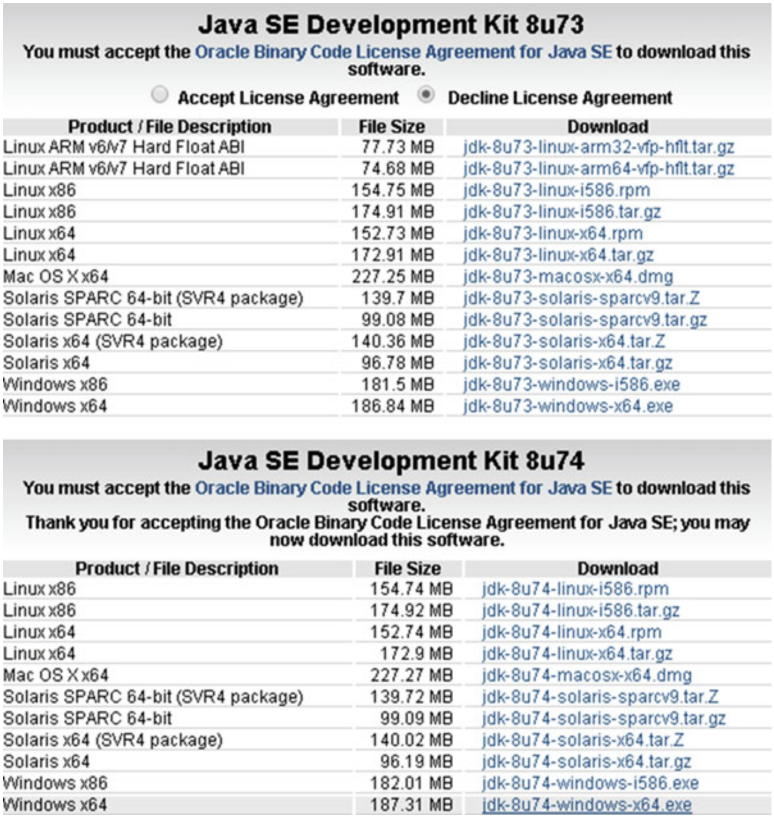


About 45,100,000 results (0.22 seconds)

Java SE Development Kit 8 - Downloads - Oracle
www.oracle.com/.../java/javase/.../jdk8-downloads-21... ▾ Oracle Corporation ▾
Download JDK 8, a development ... Java SE Development Kit 8u65 ... Kit 8u65 Demos and Samples Downloads.
JDK 8 for ARM - Download - Oracle Binary Code License - Java Magazine
You've visited this page many times. Last visit: 7/9/15

Figure A-3. *Google “Java JDK”, and then click the Downloads link*

At the Oracle web site, download and install the latest Java JDK environment, which at the time of this writing is Java SE Development Kit 8u66, as shown in Figure A-4. The URL is in the address bar in Figure A-4 and opens the download page for Java SE Development Kit 8u65 and 8u66.



Java SE Development Kit 8u73

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ● Decline License Agreement

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.73 MB	jdk-8u73-linux-arm32-vfp-hflt.tar.gz
Linux ARM v6/v7 Hard Float ABI	74.68 MB	jdk-8u73-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.75 MB	jdk-8u73-linux-i586.rpm
Linux x86	174.91 MB	jdk-8u73-linux-i586.tar.gz
Linux x64	152.73 MB	jdk-8u73-linux-x64.rpm
Linux x64	172.91 MB	jdk-8u73-linux-x64.tar.gz
Mac OS X x64	227.25 MB	jdk-8u73-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.7 MB	jdk-8u73-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.08 MB	jdk-8u73-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.36 MB	jdk-8u73-solaris-x64.tar.Z
Solaris x64	96.78 MB	jdk-8u73-solaris-x64.tar.gz
Windows x86	181.5 MB	jdk-8u73-windows-i586.exe
Windows x64	186.84 MB	jdk-8u73-windows-x64.exe

Java SE Development Kit 8u74

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	154.74 MB	jdk-8u74-linux-i586.rpm
Linux x86	174.92 MB	jdk-8u74-linux-i586.tar.gz
Linux x64	152.74 MB	jdk-8u74-linux-x64.rpm
Linux x64	172.9 MB	jdk-8u74-linux-x64.tar.gz
Mac OS X x64	227.27 MB	jdk-8u74-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.72 MB	jdk-8u74-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.09 MB	jdk-8u74-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.02 MB	jdk-8u74-solaris-x64.tar.Z
Solaris x64	96.19 MB	jdk-8u74-solaris-x64.tar.gz
Windows x86	182.01 MB	jdk-8u74-windows-i586.exe
Windows x64	187.31 MB	jdk-8u74-windows-x64.exe

***Figure A-4.** The Oracle TechNetwork Java SE JDK Download web site*

Here is the URL, in case you wanted to simply cut and paste it, copy it in, or click it to launch: `www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`.

Pull the scrollbar halfway down the page to display the Java 8 SE Development Kit (or a later version) download links table, as can be seen at the bottom of Figure A-4. You can also click the links above the table to read the explanation of the new CPU and PSU Java release versions; you're going to use Java SE 8.

Once you click Accept License Agreement, the links in the table become bolded and you can click the link you wish to use. If you are on Windows and your OS is 64-bit, use the Windows x64 link; otherwise, use the Windows x86 link. I am using what is described in these links as Windows x64, which is the 64-bit versions of Windows, for my Windows 7 and Windows 8.1 workstations.

Make sure you use this Java SE Development Kit 8u66 downloading link and not a Java Runtime Edition (JRE) link. The JRE is part of the JDK 8u66, so you do not have to worry about getting the Java Runtime separately. In case you're wondering, you use the JRE to launch and run the IntelliJ IDE, and you use the JDK in that software package to provide the Java core class foundation that is used as the foundation for the Android OS Java-based API classes.

Before you run this installation, you should remove older versions of Java. In Windows Control Panel, use Add Or Remove Programs (XP) or Programs And Features (Windows Vista, 7, 8, or 10), as shown selected in Figure A-5.

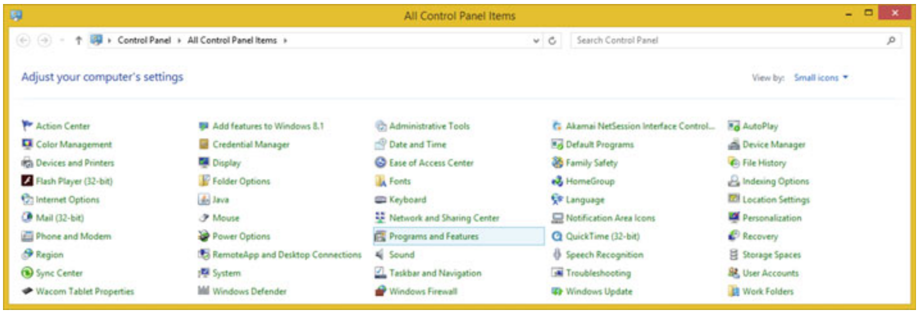
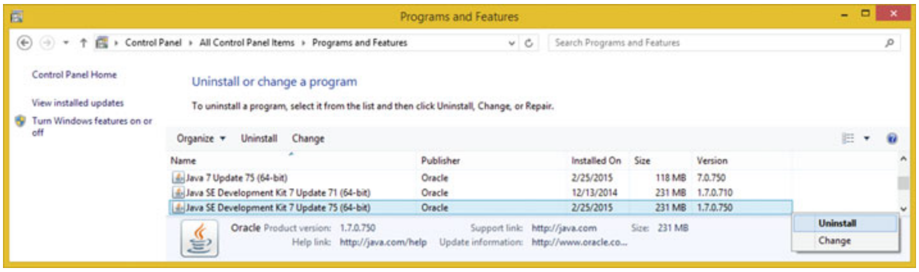


Figure A-5. *Launch your Control Panel, and choose Programs And Features*

This is necessary especially if your workstation is not brand new. You do this so that only the latest Java SE 8u66 and JRE 8u66 Java versions are currently installed on your JSON development workstation.

Select all the older Java versions, right-click each one, and select the Uninstall option, as shown in Figure A-6.

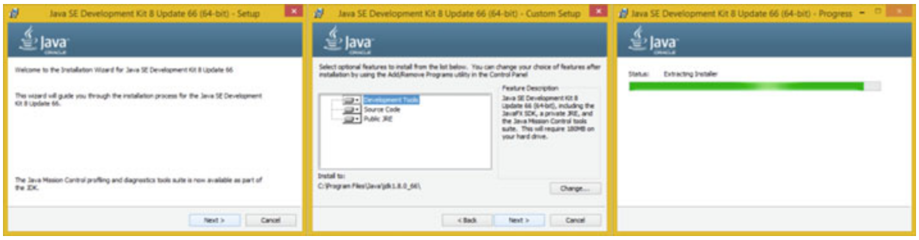

Programs and Features
Control Panel ▸ All Control Panel Items ▸ Programs and Features
Search Programs and Features
Control Panel Home
View installed updates
Turn Windows features on or off
Uninstall or change a program
To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.
Organize ▾
Uninstall
Change
Uninstall
Change

Name	Publisher	Installed On	Size	Version
Java 7 Update 75 (64-bit)	Oracle	2/25/2015	118 MB	7.0.750
Java SE Development Kit 7 Update 71 (64-bit)	Oracle	12/13/2014	231 MB	1.7.0.710
Java SE Development Kit 7 Update 75 (64-bit)	Oracle	2/25/2015	231 MB	1.7.0.750

Oracle Product version: 1.7.0.750 Support link: http://java.com Size: 231 MB
Help link: http://java.com/help Update information: http://www.oracle.co...

Figure A-6. *Find old versions of Java, right-click, and choose Uninstall*

Once you have done this and downloaded the installation executable, locate it, and double-click the `.EXE` file. Doing so launches a Setup dialog, seen at left in Figure A-7. You can also right-click your installer file and then select the Run As Administrator option.


Java SE Development Kit 8 Update 66 (64-bit) - Setup
Java
Next >
Cancel
Java SE Development Kit 8 Update 66 (64-bit) - Custom Setup
Development Tools
Source Code
Public JRE
Feature Description
Change...
< Back
Java SE Development Kit 8 Update 66 (64-bit) - Progress

Welcome to the Installation Wizard for Java SE Development Kit 8 Update 66

This wizard will guide you through the installation process for the Java SE Development Kit 8 Update 66.

The Java Mission Control profiling and diagnostics tools suite is now available as part of the JDK.

Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

Java SE Development Kit 8 Update 66 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.

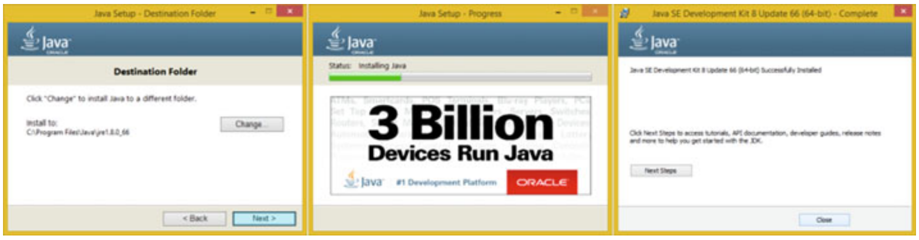
Install to:
C:\Program Files\Java\jdk1.8.0_66\

Status: Extracting Installer

Figure A-7. *Setup, Custom Setup, and Extracting Installer dialogs*

Click Next to access the Custom Setup dialog, shown in the middle of Figure A-7. Accept the default settings, and then click the button to access the Extracting Installer progress dialog shown at right in Figure A-7.

Once you've extracted the installation software, you can select a Java JDK software-installation folder. Use the default `C:\ProgramFiles\Java\jre1.8.0_66` in the Destination Folder dialog, as shown at left in Figure A-8.



Click "Change" to install Java to a different folder.

Install to:
C:\Program Files\Java\jre1.8.0_66

Status: Installing Java

Java SE Development Kit 8 Update 66 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

Figure A-8. *Destination, Progress, and Complete Install dialogs*

Click Next to install a JRE edition in the default specified folder. Interestingly, the installer won't ask you to specify the JDK folder name for some reason, probably because it wants your Java JDK to always be in a set or fixed (locked in the same location) folder.

The JDK folder is named `C:\ProgramFiles\Java\jdk1.8.0_66`. Notice that internally, Java 8 is referred to as Java 1.8.0. Thus Java 6 should be 1.6.0, and Java 7 is 1.7.0. This is useful to know, in case you are looking for Java versions using a search utility, for example, or just to show off your knowledge of legacy Java version numbering.

Once you click Next, you get the Java Setup Progress dialog shown in the middle of Figure A-8. Once Java 8 is finished installing, you finally see the Complete dialog, at right in Figure A-8. Congratulations! You have successfully installed Java 8!

Remember that the reason you did not download a JRE is because it is part of this JDK 8u66 installation. The Java 8 Runtime Edition is the executable (platform) that runs the Java software app once it has been compiled into an application. The latest JRE is also needed to run NetBeans 8, which, as you now know, is 100% completely written using the Java SE 8 development platform.

Once Java 8u66 or later is installed on your workstation, you can then download and install the latest NetBeans 8 software installer from `www.netbeans.org`. You can use the same Programs And Features (or Add Or Remove Programs) utility in your Control Panel to remove any older versions of the NetBeans development environment that may be currently installed on your JSON development workstation.

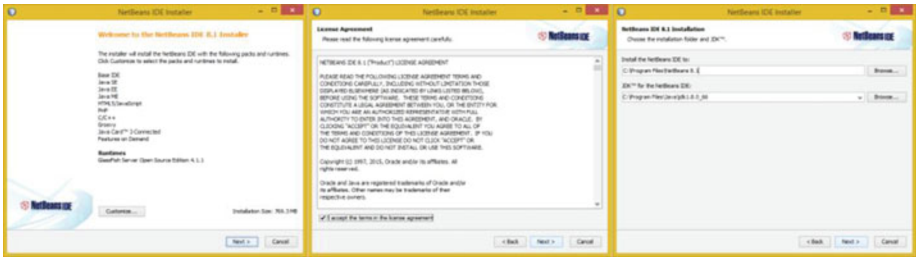
Now you are ready to add the second layer of the NetBeans 8.1 IDE software on top of Java.

NetBeans 8.1: Downloading the NetBeans JSON IDE

The second step in the process is to install the All version of the software that you saw back in Figure A-2, when you downloaded the IDE. If you have not done this yet, download the latest NetBeans version from `www.netbeans.org/downloads/`.

Click the Download button at the bottom of the All column, to the right of the download versions grid. Doing so starts the browser download function, which should put the `netbeans-8.1-windows.exe` file in your `Downloads` folder.

Find this executable file on your workstation and either double-click it or right-click it, and select the Run As Administrator option. This opens a Welcome To The NetBeans IDE 8.1 Installer dialog, shown at left in Figure A-9.


NetBeans IDE Installer

Welcome to the NetBeans IDE 8.1 Installer

The installer will install the NetBeans IDE with the following packs and runtimes.
Click Customize to select the packs and runtimes to install.

Base IDE
Java SE
Java EE
Java ME
HTML5/JavaScript
PHP
C/C++
Groovy
Java Card™ 3 Connected
Features on Demand

Runtimes
GlassFish Server Open Source Edition 4.1.1

Installation Size: 766.3 MB

NetBeans IDE Installer

License Agreement
Please read the following license agreement carefully.

NETBEANS IDE 8.1 ("Product") LICENSE AGREEMENT

PLEASE READ THE FOLLOWING LICENSE AGREEMENT TERMS AND CONDITIONS CAREFULLY, INCLUDING WITHOUT LIMITATION THOSE DISPLAYED ELSEWHERE (AS INDICATED BY LINKS LISTED BELOW), BEFORE USING THE SOFTWARE. THESE TERMS AND CONDITIONS CONSTITUTE A LEGAL AGREEMENT BETWEEN YOU, OR THE ENTITY FOR WHICH YOU ARE AN AUTHORIZED REPRESENTATIVE WITH FULL AUTHORITY TO ENTER INTO THIS AGREEMENT, AND ORACLE. BY CLICKING "ACCEPT" OR THE EQUIVALENT YOU AGREE TO ALL OF THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THIS LICENSE DO NOT CLICK "ACCEPT" OR THE EQUIVALENT AND DO NOT INSTALL OR USE THIS SOFTWARE.

Copyright (c) 1997, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

[✓] I accept the terms in the license agreement

NetBeans IDE Installer

NetBeans IDE 8.1 Installation
Choose the installation folder and JDK™.

Install the NetBeans IDE to:
C:\Program Files\NetBeans 8.1

JDK™ for the NetBeans IDE:
C:\Program Files\Java\jdk1.8.0_66

Figure A-9. *NetBeans install Welcome and License Agreement dialogs*

Click Next, select the "I accept the terms in the license agreement" option, and then click Next again. Accept the default Windows Program Files folder locations for all software installations, and again, click Next.

This opens the Summary dialog, shown in Figure A-10. Select Check For Updates, and then click the Install button to start the NetBeans 8.1 installation.

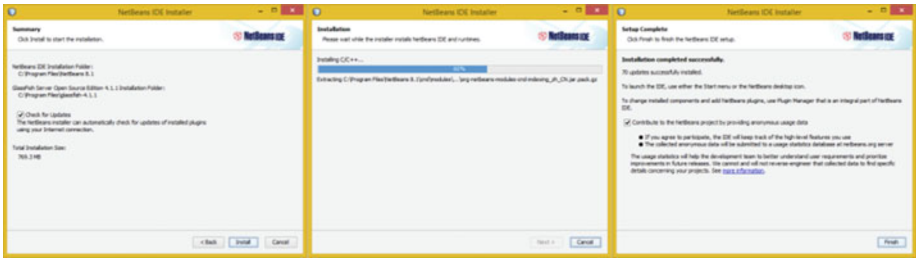

NetBeans IDE Installer
Summary
Click Install to start the installation.
NetBeans IDE Installation Folder:
C:\Program Files\NetBeans 8.1
GlassFish Server Open Source Edition 4.1.1 Installation Folder:
C:\Program Files\glassfish-4.1.1
☑ Check for Updates
The NetBeans installer can automatically check for updates of installed plugins using your Internet connection.
Total Installation Size:
769.3 MB
< Back
Install
Cancel
NetBeans IDE Installer
Installation
Please wait while the installer installs NetBeans IDE and runtimes.
Installing C/C++...
Extracting C:\Program Files\NetBeans 8.1\cnd\modules\...\org-netbeans-modules-cnd-indexing_zh_CN.jar.pack.gz
Next >
Cancel
NetBeans IDE Installer
Setup Complete
Click Finish to finish the NetBeans IDE setup.
Installation completed successfully.
70 updates successfully installed.
To launch the IDE, use either the Start menu or the NetBeans desktop icon.
To change installed components and add NetBeans plugins, use Plugin Manager that is an integral part of NetBeans IDE.
☑ Contribute to the NetBeans project by providing anonymous usage data
● If you agree to participate, the IDE will keep track of the high-level features you use
● The collected anonymous data will be submitted to a usage statistics database at netbeans.org server
The usage statistics will help the development team to better understand user requirements and prioritize improvements in future releases. We cannot and will not reverse-engineer that collected data to find specific details concerning your projects. See more information.
Finish

Figure A-10. *NetBeans Summary, Installation, and Complete dialogs*

Once the setup is complete, click the Finish button, and launch the software to make sure it works; see Figure A-11. You get into how to use NetBeans 8.1 to create a JSON project in chapter 1 covering the NetBeans 8.1 JSON IDE.

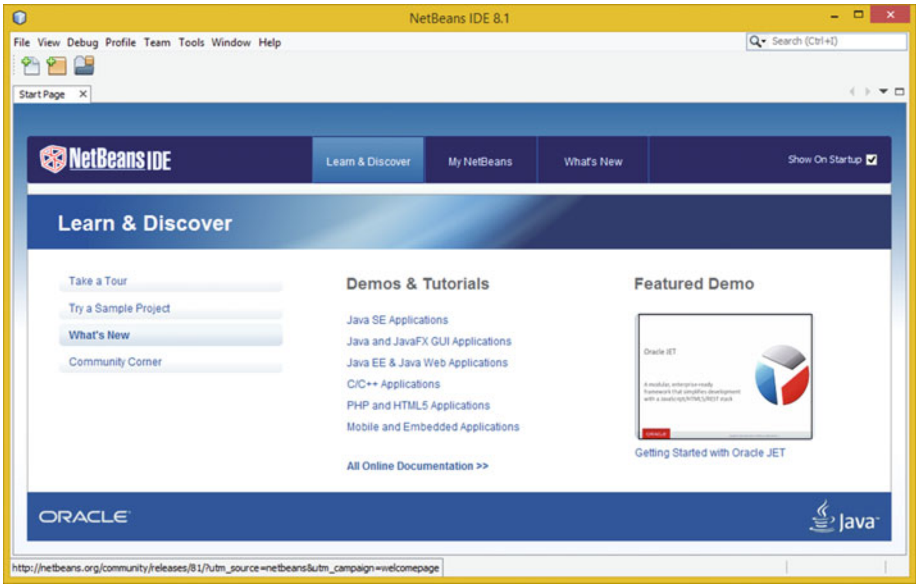

NetBeans IDE 8.1
File View Debug Profile Team Tools Window Help
Search (Ctrl+I)
Start Page
NetBeans IDE
Learn & Discover
My NetBeans
What's New
Show On Startup
Learn & Discover
Take a Tour
Try a Sample Project
What's New
Community Corner
Demos & Tutorials
Java SE Applications
Java and JavaFX GUI Applications
Java EE & Java Web Applications
C/C++ Applications
PHP and HTML5 Applications
Mobile and Embedded Applications
All Online Documentation >>
Featured Demo
Oracle JET
Getting Started with Oracle JET
ORACLE
Java
http://netbeans.org/community/releases/81/?utm_source=netbeans&utm_campaign=welcomepage

Figure A-11. *Launch NetBeans, and explore using Learn & Discover*

If you're going to be creating new media-compatible JSON applications, you need to get seven more open source packages so you can create new media assets referenced by JSON and JavaScript (or Java, PHP, C++, AJAX, and so forth. These span the new media genres including digital image compositing, 3D modeling and 3D animation, digital illustration and digital painting, digital audio editing, visual effects (VFX), and digital video editing.

GIMP 2.8: Digital Image Editing and Compositing

The GIMP project offers a professional imaging software package that allows you to do digital image editing and compositing, much as you would using Adobe Photoshop or Corel PaintShop Professional. Download this software package at `www.gimp.org` and install it; it is quite professional. GIMP is currently at version 2.8.16, but version 3.0 is just around the corner and a preview (2.9.2) of V3 is available! The GIMP home page and download button are shown in Figure A-12.

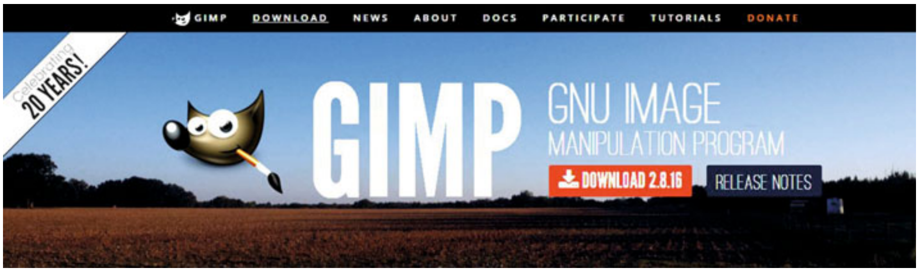


Figure A-12. *Go to `www.gimp.org`, and download GIMP 2.8.16*

If you want to learn digital image compositing, check out *Digital Image Compositing Fundamentals* (Apress, 2015).

Blender: 3D Modeling, Rendering, and Animation

The Blender Foundation project offers a professional i3D software package called Blender that allows you to do modeling of 3D objects as well as rendering and animation. Download this software package at `www.blender.org`, and install it. Blender's home page and blue download button are shown in Figure A-13.

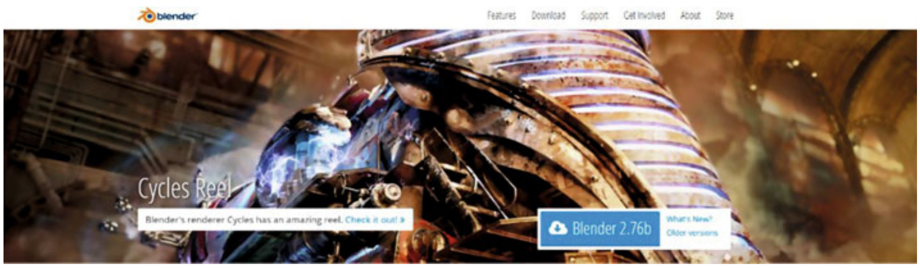


Figure A-13. *Go to `blender.org`, and download the latest version*

This is a professional-level software package with many of the same features as 3D Studio Max, Maya, XSI, and Lightwave.

Inkscape: Digital Illustration and Digital Painting

The Inkscape Project offers a professional digital illustration software package called Inkscape that can also do digital painting. Download this software package at `www.inkscape.org`. Inkscape's home page and download button are shown in Figure A-14.

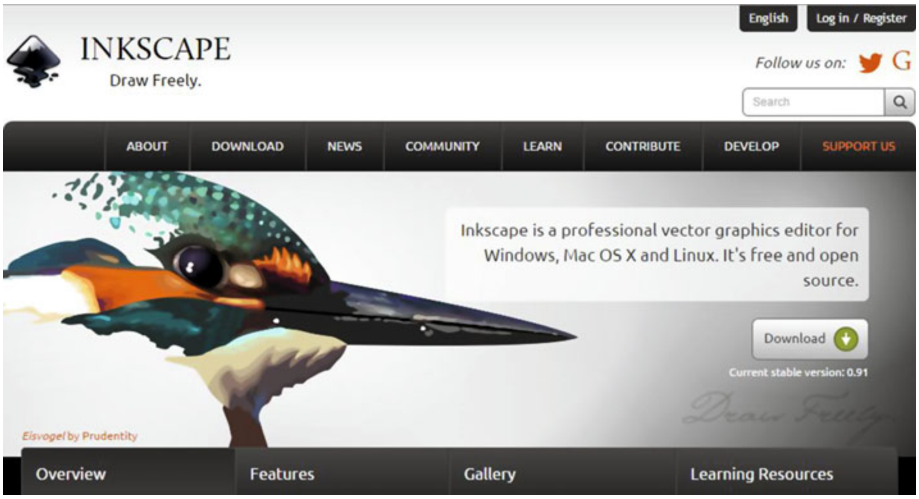
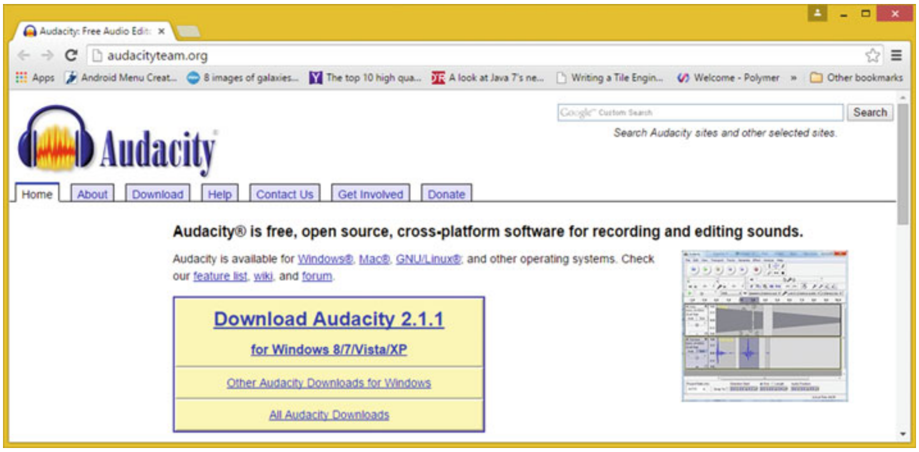


Figure A-14. *Go to `inkscape.org`, and download the latest version*

If you want to learn digital illustration, check out *Digital Illustration Fundamentals* (Apress, 2015), as well as *Digital Painting Techniques* (Apress, 2015).

Audacity: Digital Audio Editing and Special Effects

The Audacity team offers a professional digital audio software package called Audacity that is for digital audio editing, sweetening, and special effects. You can download this software package at `www.audacityteam.org`. The Audacity home page and Download link are shown in Figure A-15.



Audacity® is free, open source, cross-platform software for recording and editing sounds.

Audacity is available for Windows®, Mac®, GNU/Linux®, and other operating systems. Check our feature list, wiki, and forum.

Download Audacity 2.1.1
for Windows 8/7/Vista/XP

Other Audacity Downloads for Windows

All Audacity Downloads

Figure A-15. *Go to* `audacityteam.org`*, and download version 2.1.2*

Audacity offers many of the same digital audio editing features as many professional audio editors, and it is adding 64-bit capabilities and professional features every month. The next version will have a more professional user interface look and feel. If you want to learn more about digital audio editing, synthesis, and special effects, check out *Digital Audio Editing Fundamentals* (Apress, 2015). Digital audio can greatly enhance the user experience for any of the JSON applications you create in the future.

Visual Effects: BlackMagic Design Fusion 8.0 VFX

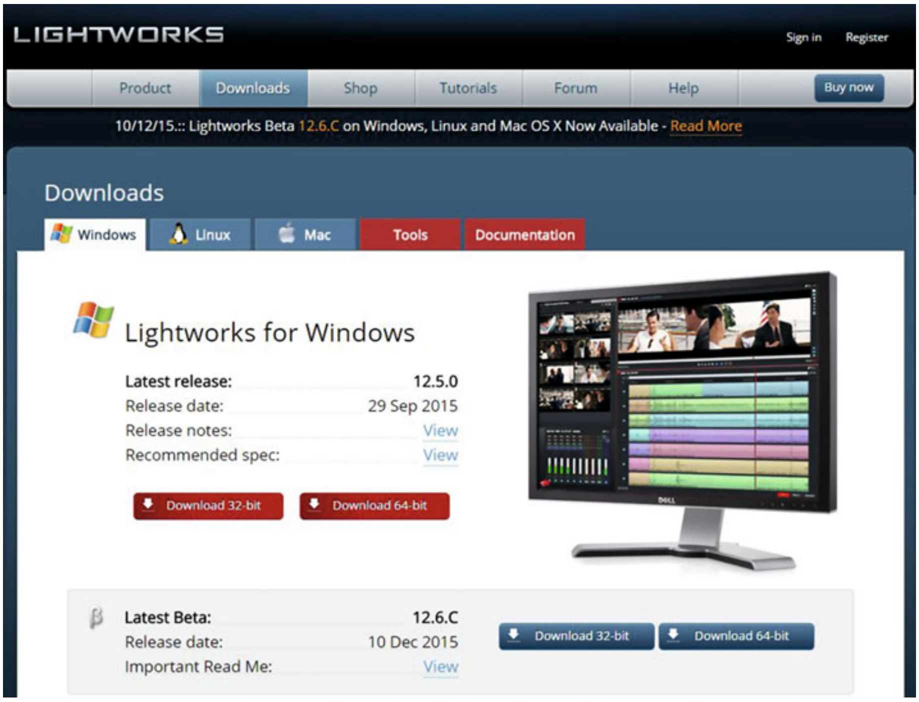
BlackMagic Design's Fusion 8 used to cost thousands of dollars; it offers a professional visual effects (VFX) software package used in film and television. Download this software package at `www.blackmagicdesign.com/products/fusion/`. Fusion 8's home page and blue Download button are shown in Figure A-16. If you want to learn more about VXF pipelines, check out *Visual Effects (VFX) Fundamentals* (Apress, 2016).

Figure A-16. *Go to* `blackmagicdesign.com`*, and download Fusion 8*

Digital Video Editing: EditShare Lightworks 12.6

Next, let's look at a free digital video editing software package called Lightworks 12.6 from EditShare. This software package has been used to create a large number of feature films—I am not setting you up here with just any software package, but rather with software that has been used for professional, commercial development. (That goes for each of these new media software packages I'm having you download and install—I'm not messing around!)

EditShare Lightworks offers professional digital video editing in a software package that also does special effects. Download the software package at `www.lwks.com` after signing up for the download. Lightworks' home page and Downloads tab are shown in Figure A-17, where you can select your OS version. I recommend using a 64-bit OS and software so that you can use 8MB of memory!



10/12/15.:: Lightworks Beta 12.6.C on Windows, Linux and Mac OS X Now Available - Read More

Lightworks for Windows

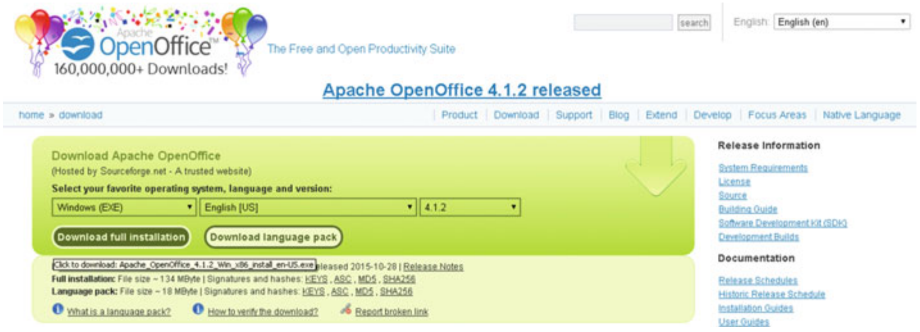
Latest release:	12.5.0
Release date:	29 Sep 2015
Release notes:	View
Recommended spec:	View

Latest Beta:	12.6.C
Release date:	10 Dec 2015
Important Read Me:	View

Figure A-17. *Go to `lwks.com`, and download Lightworks for your OS*

Office Productivity Suite: Apache OpenOffice 4.1.2

Apache OpenOffice, originally Sun Microsystems' StarOffice, was acquired by Oracle and released as open source. This will provide your JSON development business with professional office and business productivity software support. Download this great software package at `www.openoffice.com`. The Apache OpenOffice home page and Download button are shown in Figure A-18, where you can select your OS, language, and software version. I recommend using a 64-bit OS and software so that you can use 8MB of memory.


Apache OpenOffice
160,000,000+ Downloads!
The Free and Open Productivity Suite
search
English: English (en)
Apache OpenOffice 4.1.2 released
home » download
Product | Download | Support | Blog | Extend | Develop | Focus Areas | Native Language
Download full installation
Download language pack
What is a language pack?
How to verify the download?
Report broken link

Download Apache OpenOffice
(Hosted by Sourceforge.net - A trusted website)
Select your favorite operating system, language and version:
Windows (EXE) | English [US] | 4.1.2

Click to download: Apache_OpenOffice_4.1.2_Win_x86_install_en-US.exe leased 2015-10-28 | Release Notes
Full installation: File size ~ 134 MByte | Signatures and hashes: KEYS , ASC , MD5 , SHA256
Language pack: File size ~ 18 MByte | Signatures and hashes: KEYS , ASC , MD5 , SHA256

Release Information
System Requirements
License
Source
Building Guide
Software Development Kit (SDK)
Development Builds

Documentation
Release Schedules
Historic Release Schedule
Installation Guides
User Guides

Figure A-18. *Download Apache OpenOffice 4.1.2 at* `OpenOffice.com`

Summary

In this appendix, you set up a NetBeans 8 JSON workstation by downloading and installing the open source Java 8 JDK and NetBeans 8.1 IDE software that you can use to code JSON applications. I also showed you some valuable professional new media software that is free for commercial use. Appendix B shows you how to set up an Eclipse 4.5.1 Mars JSON workstation in much the same fashion. You also see how to set up an IntelliJ IDEA 15 JSON workstation in Appendix C, if you're an Android Studio developer and prefer to use that IDE instead.

APPENDIX B

Eclipse Mars: Setting Up a JSON Integrated Development IDE

In this appendix, let's put together your foundation for a highly professional, JSON-friendly, Eclipse 4.5.1 (Mars) integrated development environment (IDE). Your development workstation is the most important combination of PC hardware and software, allowing you to reach your goal of JSON-compatible application development. This appendix considers your hardware needs and the software infrastructure to put together a professional, well-rounded JSON workstation that gives you a bunch of arrows in your software development quiver right off the bat. You will then have everything you need when you're reading the book's chapters, no matter what type of JSON application you decide that you want to develop for your end users!

All readers of this book should be developing with identical JSON application software development environments, because everything you learn over the course of this book needs to be able to be experienced equally by everyone. Appendix A outlines all the steps to put together a NetBeans 8.1 IDE based JSON development workstation, and Appendix C does the same for IntelliJ IDEA. These three IDE-customized appendixes get all the tedious setup tasks out of the way. If you already have your workstation configured, you can proceed to Chapter 2 for an overview of JSON; or, if you are already familiar with JSON, you're ready to read the rest of the book.

In this appendix, you learn where to download and how to install several of the most impressive, professional, open source software packages on the face of this planet. You are about to max out your JSON, HTML5, CSS3, and JavaScript (JS) development workstation, so hold on tight and enjoy this virtual download ride!

Creating a JSON Development Workstation

The first thing you'll do after taking a look at hardware requirements is to download and install the entire Java software development kit (SDK), which Oracle calls Java SE 8 Java Development Kit (JDK). Eclipse 4.5.1, which is called the Mars IDE, uses the Java 8 Standard Edition (SE).

The second thing you'll download and install is the Eclipse Mars IDE that you get from www.eclipse.org. The Eclipse Mars IDE allows you to develop JSON compatible applications using all the popular programming languages, including Java EE, Java SE, Java Server Faces (JSF), JavaFX, as well as HTML5, CSS3 and JS.

After your JSON application development environment is set up, you can then download and install new media asset development tools, if you wish. These are used in conjunction with NetBeans 8.1 for things such as image editing (GIMP) and nonlinear digital video editing (Lightworks); special effects (Fusion); digital audio sweetening, or editing (Audacity); i3D modeling, rendering, and animation (Blender); digital illustration (Inkscape); and business productivity (OpenOffice).

This appendix will take your development to an all-new level, showing you how to create the media development and programming workstation that will run your business. All the software development tools you'll download and install come close to matching all the primary feature sets of expensive paid software packages, such as those from Apple (Final Cut Pro), Autodesk (3D Studio Max), Adobe (Photoshop, Illustrator, After Effects), and Avid (ProTools), and all at *zero* cost to your production company!

Open source software is free to download, install, and upgrade and is continually adding features. It's becoming more and more like professional software every day. You will be amazed at how professional open source software packages have become over the last decade or so.

Hardware Foundation

Because in this appendix you put together the foundation for the JSON-capable application development workstation you use throughout this book, I want to take a moment to review Eclipse Mars' JSON development workstation hardware requirements. This is a factor that will influence your development performance (speed). This is clearly as important as the software itself, because hardware is what is actually running the software package's algorithms.

Minimum requirements for the Eclipse Mars IDE include 2GB of memory, 900MB of hard disk space, and a WXGA (1280x768) display. Next let's discuss what you need to make an Eclipse Mars JSON IDE usable. Start by upgrading your 1280×768 WXGA display to an HDTV (1920×1080 at 120FPS refresh rate) or UHD (4096×2160 at 120FPS refresh rate) widescreen display. These are now affordable and give you 3 to 12 times the display real estate of a WXGA display. HDTVs are now $250 to $500, and UHDTV displays are under $1,000.

I recommend using, at a bare minimum, the Intel i7 quad-core processor or the AMD 64-bit octa-core processor. Install at least 8GB of DDR3-1600 memory. I'm using a 64-bit, octa-core AMD 8350, with 16GB of DDR3-1600. Intel also has a hexa-core i7 processor. This would be the equivalent of having 12 cores, because each i7 core can host two threads. Similarly, an i7 quad-core should look like 8 cores to your 64-bit operating system's thread-scheduling algorithm.

There are also high-speed DDR3-1800 as well as DDR3-2133 clock-speed memory module components available. A high number signifies fast memory-access speeds. To calculate actual megahertz speeds at which the memory is cycling, divide the number by 4 (1333 = 333Mhz, 1600 = 400Mhz, 1800 = 450Mhz, 2133 = 533Mhz). Memory-access speed is a massive workstation performance factor, because your processor is usually bottlenecked by the speed at which processor cores can access the data (in memory) that that processor core needs to process.

With high-speed processing and memory access going on in the workstation while it is operating, it's extremely important to keep everything cool so that you do not experience thermal problems. I recommend using a wide, full-tower enclosure with 120mm or 200mm cooling fans (one or two at least), as well as a captive liquid-induction cooling fan on the CPU. It is important to note that the cooler the system runs, the faster it can run, and the longer it will last, so load the workstation up with lots of silent high-speed fans!

If you really want maximum performance, install a solid state disk (SSD) drive as the primary disk drive from which your applications and operating system software can load. Use legacy HDD hardware for your `D:\` hard drive for slower data storage (long-term). Put your current project files on the SSD.

As far as OS goes, I am using a 64-bit Windows 8 operating system that is fairly memory efficient. The Linux 64-bit OS is extremely memory efficient. I recommend using any 64-bit OS so you can address more than 3.24GB of system memory.

Open Source Software

To create a well-rounded, professional JSON application development workstation, you'll install all the primary genres of open source software. First you will install Java SE 8 and Eclipse Mars. I will also show you how to download GIMP, Lightworks, Fusion, Blender3D, and Audacity, which are also all open source software packages, in case your JSON applications will use a graphical front end. I also recommend other free software at the end of the appendix so you can put together the major production workstation you have always dreamed of.

Open source software is approaching the level of professionalism of paid development software packages that cost thousands of dollars each. Using open source software packages like Java 8, NetBeans 8, Blender, GIMP, Audacity, Lightworks, Fusion, OpenOffice, and others, you can put together a free application development workstation and rival paid software workstations.

If you just purchased a new JSON development workstation PC and are going to put together the entire development software suite from scratch, this appendix goes through the entire work process.

Java 8: Installing the Foundation for Eclipse Mars

The first thing to do is visit the Eclipse web site at `www.eclipse.org`. Click the orange Download button on the home page, as shown in Figure B-1.

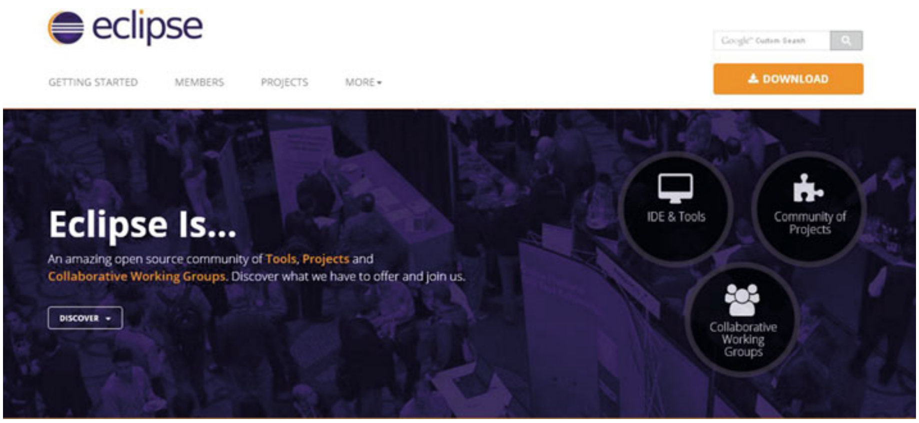

eclipse
GETTING STARTED
MEMBERS
PROJECTS
MORE
DOWNLOAD
Eclipse Is...
An amazing open source community of Tools, Projects and Collaborative Working Groups. Discover what we have to offer and join us.
DISCOVER
IDE & Tools
Community of Projects
Collaborative Working Groups

Figure B-1. *Go to* `eclipse.org`*, and click the orange Download button*

As you will see, there are a plethora of download options to consider, including one that can support JavaScript, which JSON is based on. I suggest this Java EE version, which supports all the popular Java programming languages, with web applications, each of which JSON works with. Each of the downloads offers 32-bit and 64-bit versions, as you can see at the bottom of Figure B-2. This is because these have been precompiled, whereas the other NetBeans IDE installs required Java 8 JDK to be installed. This tells you that although Eclipse Mars was created with Java SE 8, it is distributed in a Windows binary format, not in Java bytecode format, like NetBeans is. You can see further proof of this in Figure A-2, where non-Java versions of NetBeans 8.1 are also compiled out to OS binary format, whereas the Java versions use the Java bytecode format binaries and do not specify a bit level (the Java SE 8 environment does this for you).

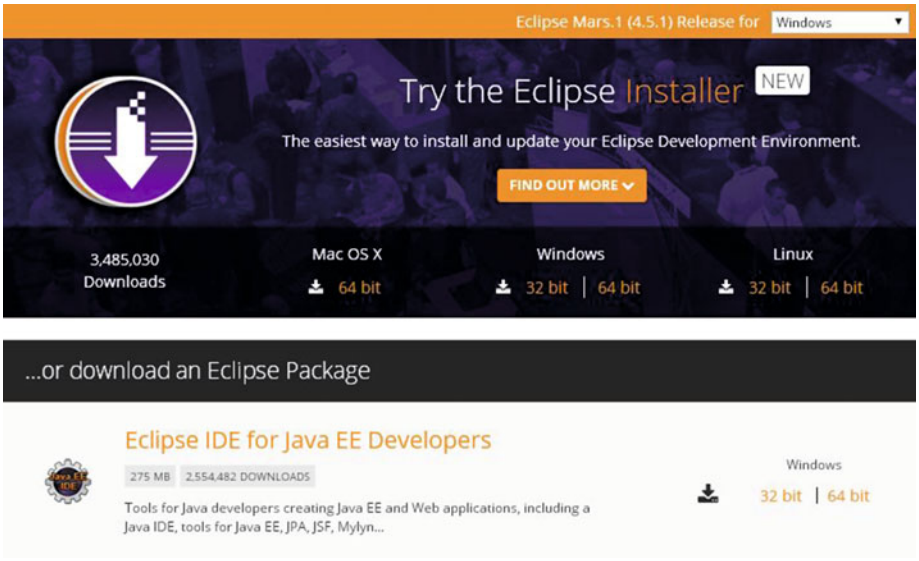
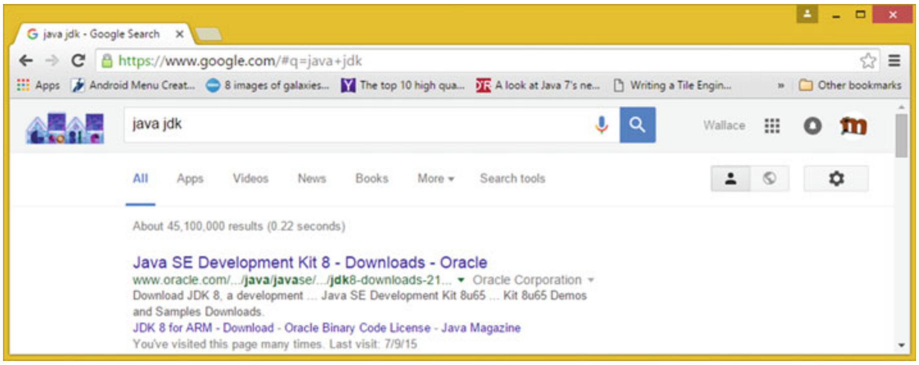

Eclipse Mars.1 (4.5.1) Release for
Windows
Try the Eclipse Installer
NEW
The easiest way to install and update your Eclipse Development Environment.
FIND OUT MORE
3,485,030 Downloads
Mac OS X
64 bit
Windows
32 bit | 64 bit
Linux
32 bit | 64 bit
...or download an Eclipse Package
Eclipse IDE for Java EE Developers
275 MB
2,554,482 DOWNLOADS
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn...
Windows
32 bit | 64 bit

Figure B-2. *Download the Java EE with HTML5/JavaScript version*

Because JSON works with Java 8, just like with JavaScript, get the Java 8 JDK just to be thorough in your development system configuration process. If you want an Enterprise Edition (EE) version of Java 8, download the Java 8 EE JDK.

Open Google Chrome and Google "Java JDK", as shown in Figure B-3. Look for the Java SE Development Kit 8 - Downloads search result, and click it to open the Oracle Java SE site, which is shown in Figure B-4 and located at `www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`.

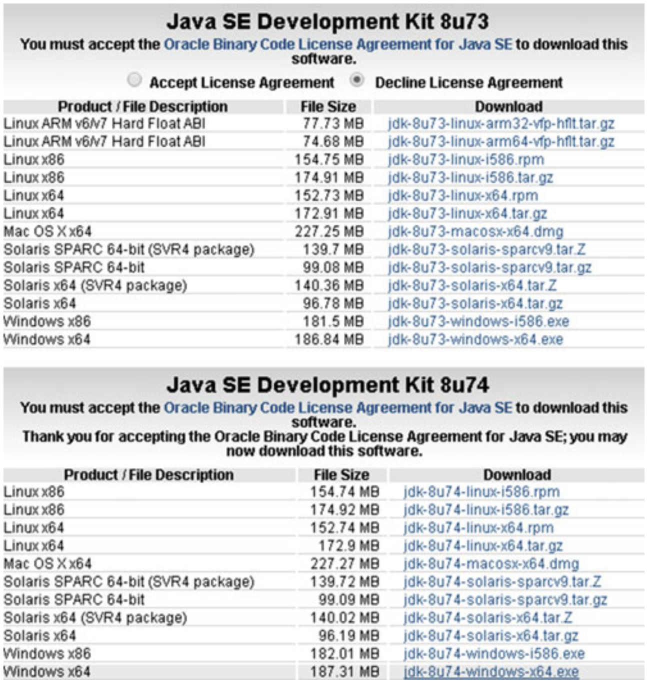


About 45,100,000 results (0.22 seconds)

Java SE Development Kit 8 - Downloads - Oracle
www.oracle.com/.../java/javase/.../jdk8-downloads-21... ▾ Oracle Corporation ▾
Download JDK 8, a development ... Java SE Development Kit 8u65 ... Kit 8u65 Demos and Samples Downloads.
JDK 8 for ARM - Download - Oracle Binary Code License - Java Magazine
You've visited this page many times. Last visit: 7/9/15

Figure B-3. *Google "Java JDK", and then click the Downloads link*

Go to the Oracle web site, and download the latest Java 8 JDK environment, which at the time of this writing is Java SE Development Kit 8u74, as shown in Figure B-4. The URL listed earlier opens a download page for Java SE 8 Development Kit JDK 8u73 as well as JDK 8u74.



Java SE Development Kit 8u73

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement [✓ Decline License Agreement]

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.73 MB	jdk-8u73-linux-arm32-vfp-hflt.tar.gz
Linux ARM v6/v7 Hard Float ABI	74.68 MB	jdk-8u73-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.75 MB	jdk-8u73-linux-i586.rpm
Linux x86	174.91 MB	jdk-8u73-linux-i586.tar.gz
Linux x64	152.73 MB	jdk-8u73-linux-x64.rpm
Linux x64	172.91 MB	jdk-8u73-linux-x64.tar.gz
Mac OS X x64	227.25 MB	jdk-8u73-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.7 MB	jdk-8u73-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.08 MB	jdk-8u73-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.36 MB	jdk-8u73-solaris-x64.tar.Z
Solaris x64	96.78 MB	jdk-8u73-solaris-x64.tar.gz
Windows x86	181.5 MB	jdk-8u73-windows-i586.exe
Windows x64	186.84 MB	jdk-8u73-windows-x64.exe

Java SE Development Kit 8u74

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	154.74 MB	jdk-8u74-linux-i586.rpm
Linux x86	174.92 MB	jdk-8u74-linux-i586.tar.gz
Linux x64	152.74 MB	jdk-8u74-linux-x64.rpm
Linux x64	172.9 MB	jdk-8u74-linux-x64.tar.gz
Mac OS X x64	227.27 MB	jdk-8u74-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.72 MB	jdk-8u74-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.09 MB	jdk-8u74-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.02 MB	jdk-8u74-solaris-x64.tar.Z
Solaris x64	96.19 MB	jdk-8u74-solaris-x64.tar.gz
Windows x86	182.01 MB	jdk-8u74-windows-i586.exe
Windows x64	187.31 MB	jdk-8u74-windows-x64.exe

Figure B-4. *Oracle TechNetwork Java 8 SE JDK downloads web site*

Pull the scrollbar halfway down the page to display the Java 8 SE Development Kit 8u74 (or a later version) download links table, as can be seen at the bottom of Figure B-4. You can also click the links above the table to read the explanation of the new CPU and PSU Java release versions; you're going to use the latest Java 8u74 version.

Once you click Accept License Agreement, the links in the table become bolded and you can click the link you wish to use. If you are on Windows and your OS is 64-bit, use the Windows x64 link; otherwise, use the Windows x86 link. I am using what is described in these links as Windows x64, which is the 64-bit version of Windows, for my Windows 7 and Windows 8.1 workstations.

Make sure you use this Java SE Development Kit 8u74 download link, and not the Java Runtime Edition (JRE) link. This JRE is part of the JDK 8u74, so you do not have to worry about getting the Java Runtime separately.

Before you run this installation, you should remove older versions of Java. In Windows Control Panel, use Add Or Remove Programs (XP) or Programs And Features (Windows Vista, 7, 8, or 10), as shown selected in Figure B-5.

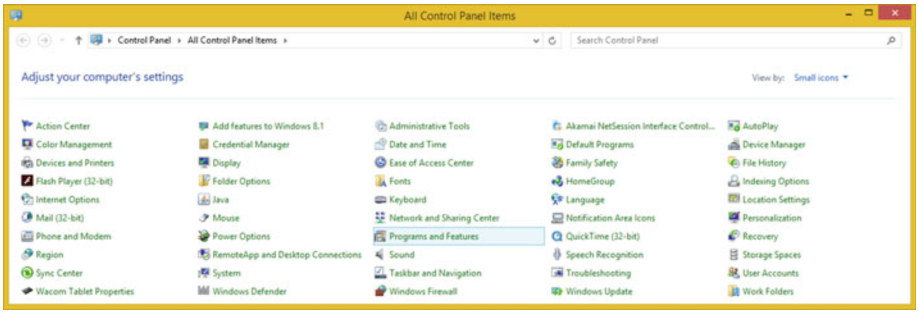
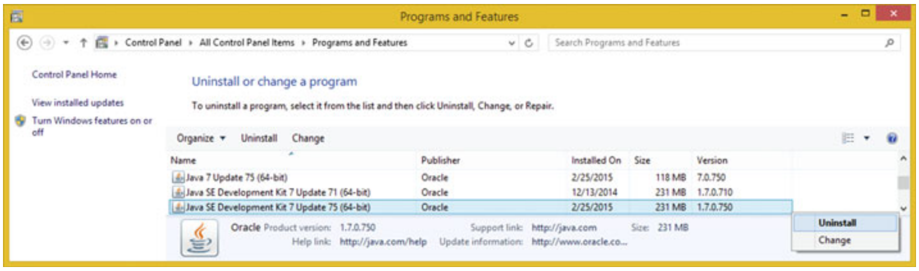


Figure B-5. *Launch your Control Panel, and choose Programs And Features*

This is necessary especially if your workstation is not brand new. You do this so that only the latest Java SE 8u74 and JRE 8u74 versions are currently installed on your JSON development workstation. You will also do this for any older IDEs that you have (NetBeans, Eclipse, or IntelliJ).

Select all the older Java versions, right-click each one, and select the Uninstall option, as shown in Figure B-6. You can also perform this process before installing other software packages, although traditional media production software packages should replace older versions automatically, as part of their install process.

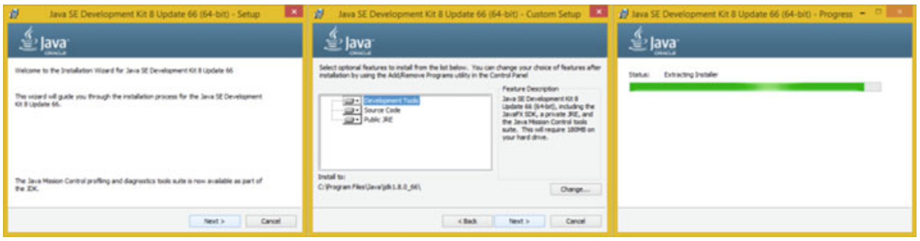

Programs and Features
Control Panel ▸ All Control Panel Items ▸ Programs and Features
Search Programs and Features
Control Panel Home
View installed updates
Turn Windows features on or off
Uninstall or change a program
To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.
Organize ▾
Uninstall
Change
Uninstall
Change

Name	Publisher	Installed On	Size	Version
Java 7 Update 75 (64-bit)	Oracle	2/25/2015	118 MB	7.0.750
Java SE Development Kit 7 Update 71 (64-bit)	Oracle	12/13/2014	231 MB	1.7.0.710
Java SE Development Kit 7 Update 75 (64-bit)	Oracle	2/25/2015	231 MB	1.7.0.750

Oracle Product version: 1.7.0.750 Support link: http://java.com Size: 231 MB
Help link: http://java.com/help Update information: http://www.oracle.co...

Figure B-6. *Find old versions of Java, right-click, and choose Uninstall*

Once you have done this and downloaded the installation executable, locate it, and double-click the `.EXE` file. Doing so launches a Setup dialog, seen at left in Figure B-7. You can also right-click your installer file and then select the Run As Administrator option; this gives you better file access so that you have OS permissions granted, such as read, write, overwrite, append, and delete, which a installer may need to access in order to complete the installation.


Java SE Development Kit 8 Update 66 (64-bit) - Setup

Welcome to the Installation Wizard for Java SE Development Kit 8 Update 66

This wizard will guide you through the installation process for the Java SE Development Kit 8 Update 66.

The Java Mission Control profiling and diagnostics tools suite is now available as part of the JDK.

Java SE Development Kit 8 Update 66 (64-bit) - Custom Setup

Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

Development Tools
Source Code
Public JRE

Feature Description
Java SE Development Kit 8 Update 66 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.

Install to:
C:\Program Files\Java\jdk1.8.0_66\

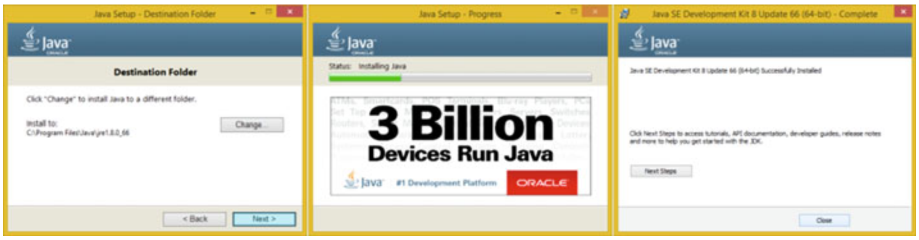
Java SE Development Kit 8 Update 66 (64-bit) - Progress

Status: Extracting Installer

Figure B-7. *Setup, Custom Setup, and Extracting Installer dialogs*

Click Next to access the Custom Setup dialog, shown in the middle of Figure B-7. Accept the default settings, and then click the button to access the Extracting Installer progress dialog shown at right in Figure B-7.

Once you've extracted the installation software, you can select a Java JDK software installation folder. Use the default `C:\ProgramFiles\Java\jre1.8.0_74` in the Destination Folder dialog, as shown at left in Figure B-8.


Java Setup - Destination Folder

Destination Folder

Click "Change" to install Java to a different folder.

Install to:
C:\Program Files\Java\jre1.8.0_66

Java Setup - Progress

Status: Installing Java

Java SE Development Kit 8 Update 66 (64-bit) - Complete

Java SE Development Kit 8 Update 66 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

Figure B-8. *Destination, Progress, and Complete Install dialogs*

Click Next to install a JRE edition in the default specified folder. Interestingly, the installer won't ask you to specify the JDK folder name for some reason, probably because it wants your Java JDK to always be in a set or fixed (locked in the same location) folder.

The JDK folder is named `C:\ProgramFiles\Java\jdk1.8.0_74`. Notice that internally, Java 8 is referred to as Java 1.8.0. Thus Java 6 should be 1.6.0, and Java 7 is 1.7.0. This is useful to know, in case you are looking for Java versions using a search utility, for example, or just to show off your knowledge of legacy Java version numbering.

Once you click Next, you get the Java Setup Progress dialog shown in the middle of Figure B-8. Once Java 8 is finished installing, you finally see the Complete dialog, at right in Figure B-8. Congratulations! You have successfully installed Java 8!

Remember that the reason you did not download a JRE is because it is part of this JDK 8u74 installation. The Java 8 Runtime Edition is the executable (platform) that runs the Java software app once it has been compiled into an application. the latest JRE is also needed to run NetBeans and Eclipse Mars which, as you now know, is 100% completely written using the Java SE 8 development platform, as well as to work with Android Studio 2.

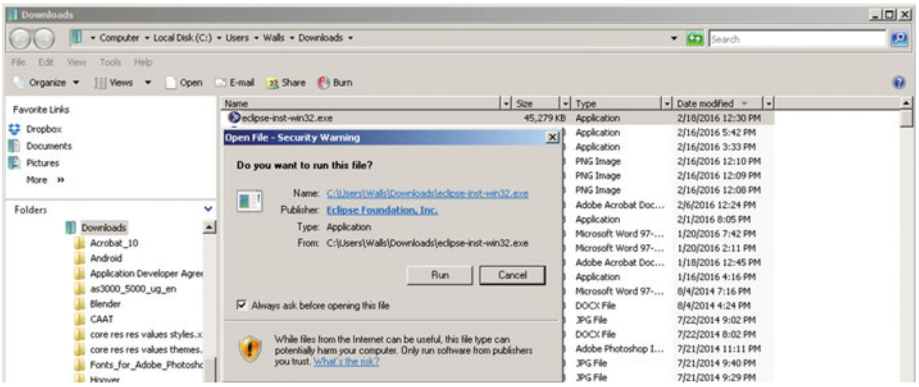
Once Java 8u74 or later is installed on your workstation, you can then download and install the latest Eclipse software installer from `www.eclipse.org`.

You can use the same Programs And Features (or Add Or Remove Programs) utility in your Control Panel to remove any older versions of the NetBeans development environment that may be currently installed on your JSON development workstation.

Now you are ready to add the second layer of the Eclipse 4.5 Mars IDE software.

Eclipse 4.5: Installing the Eclipse Mars JSON IDE

The second step in this process, give that you visited the `eclipse.org` web site and downloaded the installer in the previous section, is to install that JavaEE version of the software, which you saw back in Figure B-2. Find the executable file on your workstation and either double-click it or right-click it, and select the Run As Administrator option. This should open the Security Warning “Do you want to run this file?” dialog, as shown in Figure B-9.



Name	Size	Type	Date modified
eclipse-inst-win32.exe	45,279 KB	Application	2/18/2016 12:30 PM
		Application	2/16/2016 5:42 PM
		Application	2/16/2016 3:33 PM
		PNG Image	2/16/2016 12:10 PM
		PNG Image	2/16/2016 12:09 PM
		PNG Image	2/16/2016 12:08 PM
		Adobe Acrobat Doc...	2/6/2016 12:24 PM
		Application	2/1/2016 8:05 PM
		Microsoft Word 97-...	1/20/2016 7:42 PM
		Microsoft Word 97-...	1/20/2016 2:11 PM
		Adobe Acrobat Doc...	1/18/2016 12:45 PM
		Application	1/16/2016 4:16 PM
		Microsoft Word 97-...	8/4/2014 7:16 PM
		DOCX File	8/4/2014 4:24 PM
		JPG File	7/22/2014 9:02 PM
		DOCX File	7/22/2014 8:02 PM
		Adobe Photoshop I...	7/21/2014 11:11 PM
		JPG File	7/21/2014 9:40 PM
		JPG File	7/21/2014 9:29 PM

Open File - Security Warning

Do you want to run this file?

Name: C:\Users\Walls\Downloads\eclipse-inst-win32.exe
Publisher: Eclipse Foundation, Inc.
Type: Application
From: C:\Users\Walls\Downloads\eclipse-inst-win32.exe

Run | Cancel

[✓] Always ask before opening this file

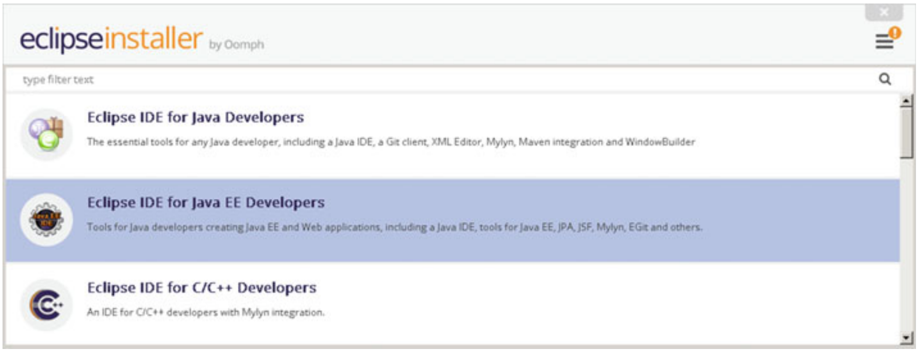
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Figure B-9. *NetBeans install Welcome and License Agreement dialogs*

Click the Run button to launch the installation. You see the Eclipse Installer By Oomph loader screen, shown in Figure B-10.

Figure B-10. *Eclipse Installer By Oomph loader screen*

Once the installer has loaded into memory, the software version selector dialog shown in Figure B-11 appears. Select the version that supports JavaScript (web applications), which is the Eclipse IDE for Java EE Developers.


eclipseinstaller by Oomph

type filter text

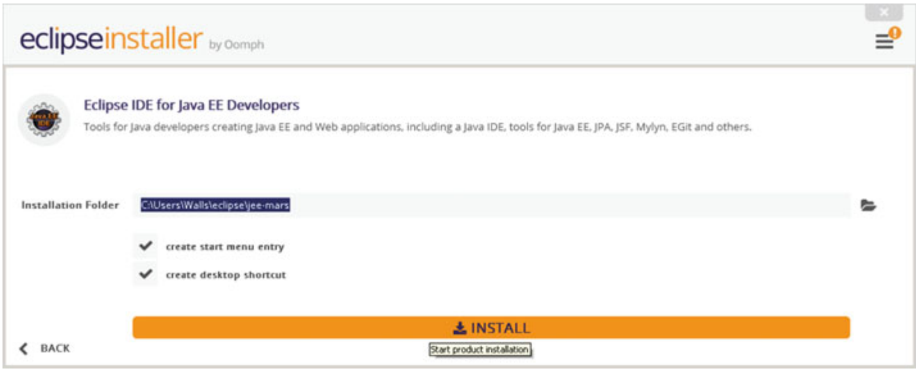
Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven integration and WindowBuilder

Eclipse IDE for Java EE Developers
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.

Figure B-11. *Click the Eclipse IDE For Java EE Developers option*

In the Installation Folder dialog, accept the default folder name offered by Eclipse, and select the Create Start Menu entry and Create Desktop Shortcut option (if needed) for your workstation. These are shown at left in Figure B-12.


eclipseinstaller by Oomph

Eclipse IDE for Java EE Developers
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.

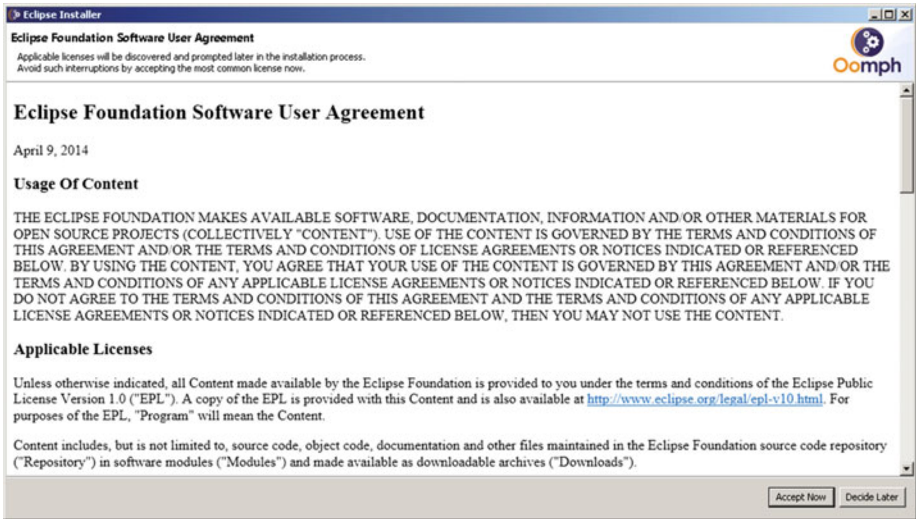
Installation Folder C:\Users\Walls\eclipse\jee-mars

✓ create start menu entry
✓ create desktop shortcut

INSTALL
Start product installation
BACK

Figure B-12. *Select Launch Options, default installation folder*

Click the Install button. Then, in the Eclipse Foundation Software User Agreement dialog, shown in Figure B-13, select the “I accept the terms in the license agreement” option, after you (or your legal department) have reviewed the terms and conditions specifying what you can and cannot do using this software. Acceptance of the licensing terms and conditions is accomplished in this case by clicking the Accept Now button.


Eclipse Installer

Eclipse Foundation Software User Agreement

Applicable licenses will be discovered and prompted later in the installation process.
Avoid such interruptions by accepting the most common license now.

Eclipse Foundation Software User Agreement

April 9, 2014

Usage Of Content

THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT"). USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW, THEN YOU MAY NOT USE THE CONTENT.

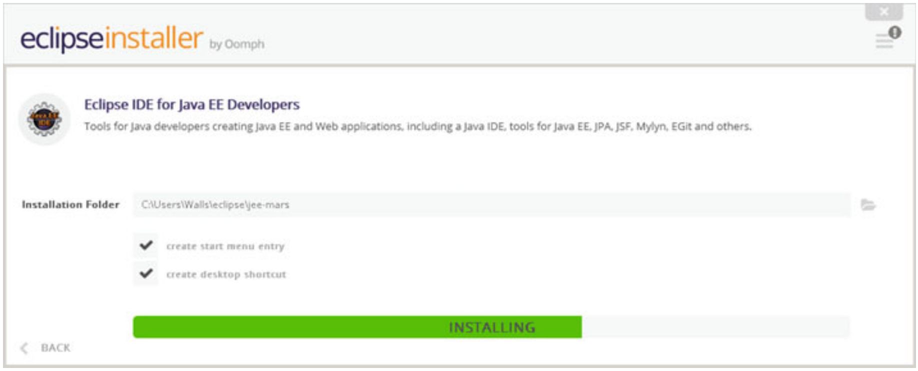
Applicable Licenses

Unless otherwise indicated, all Content made available by the Eclipse Foundation is provided to you under the terms and conditions of the Eclipse Public License Version 1.0 ("EPL"). A copy of the EPL is provided with this Content and is also available at http://www.eclipse.org/legal/epl-v10.html. For purposes of the EPL, "Program" will mean the Content.

Content includes, but is not limited to, source code, object code, documentation and other files maintained in the Eclipse Foundation source code repository ("Repository") in software modules ("Modules") and made available as downloadable archives ("Downloads").

Figure B-13. *Click the Accept Now button to agree to the terms*

Once you agree to the terms of your licensing agreement, you see an Installing dialog with a green progress bar, as shown in Figure B-14.


Eclipse IDE for Java EE Developers

Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.

Installation Folder C:\Users\Walls\eclipse\jee-mars

✓ create start menu entry

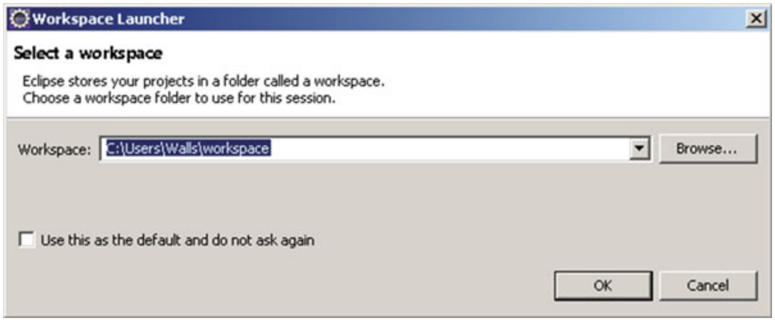
✓ create desktop shortcut

Figure B-14. *The Installing progress bar appears in green*

When the installation is completed, Eclipse Mars should launch automatically, displaying the branded startup screen shown in Figure B-15. This appears as the software loads into memory from your hard disk drive for the first time.

Figure B-15. *Eclipse Mars launch*

Once Eclipse Mars launches, it displays the Workspace Launcher dialog. It prompts you to select a Workspace location for your hard disk drive, as shown in Figure B-16.



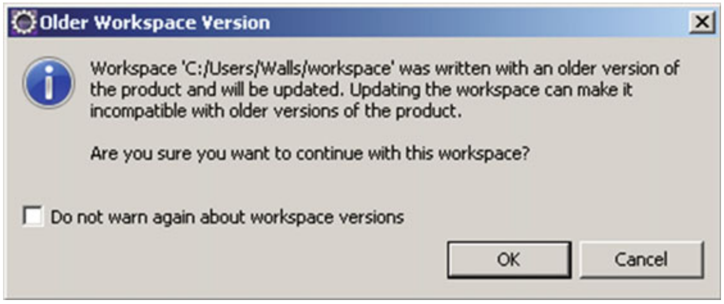
Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\Walls\workspace

Figure B-16. *Select a Workspace folder name for Eclipse projects*

I selected the default `C:\Users\Walls\workspace` directory, because I felt that name was acceptable. As you can see in Figure B-17, if you have a previous version of Eclipse installed, an Older Workspace Version dialog prompts you to update.



Workspace 'C:/Users/Walls/workspace' was written with an older version of the product and will be updated. Updating the workspace can make it incompatible with older versions of the product.

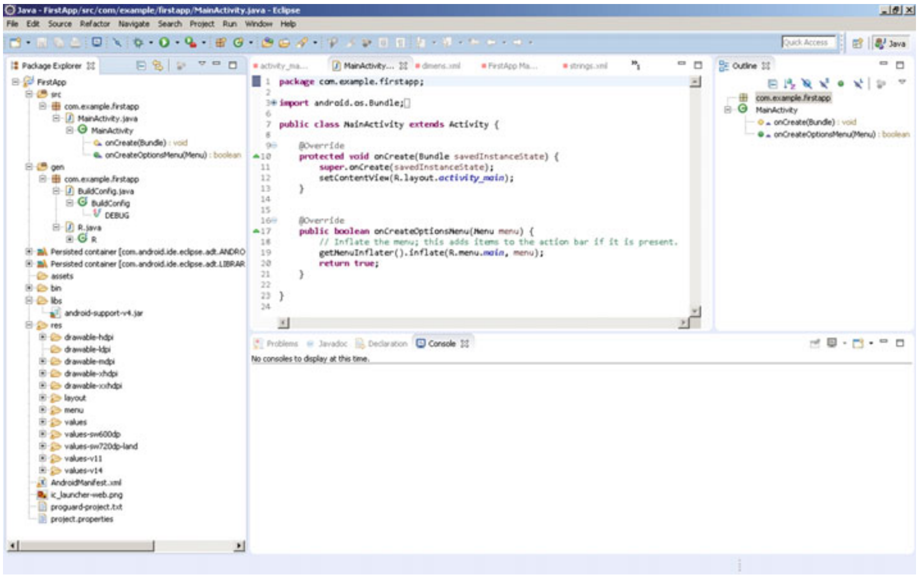
Are you sure you want to continue with this workspace?

Figure B-17. *Update the older version of Workspace (if necessary)*

After all of this is complete, Eclipse Mars launches, using the Eclipse Mars Loader screen shown in Figure B-18.

Figure B-18. *Eclipse Mars loading screen*

Figure B-19 shows Eclipse Mars with the sample FirstApp.



```
package com.example.firstapp;

import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

```
No consoles to display at this time.
```

Figure B-19. *Eclipse IDE on startup, with the `FirstApp` sample app*

If you're going to be creating new media-compatible JSON applications, you need to get open source packages to create new media assets referenced by JSON. They span new media genres including digital imaging, digital illustration, digital audio, digital painting, visual effects, digital video editing, and 3D.

GIMP 2.8: Digital Image Editing and Compositing

The GIMP project offers a professional imaging software package that allows you to do digital image editing and compositing, much as you would using Adobe Photoshop or Corel PaintShop Professional. Download this software package at `www.gimp.org` and install it; it is quite professional. GIMP is currently at version 2.8.16, but version 3.0 is just around the corner and a preview (2.9.2) of V3 is available! The GIMP home page and download button are shown in Figure B-20.

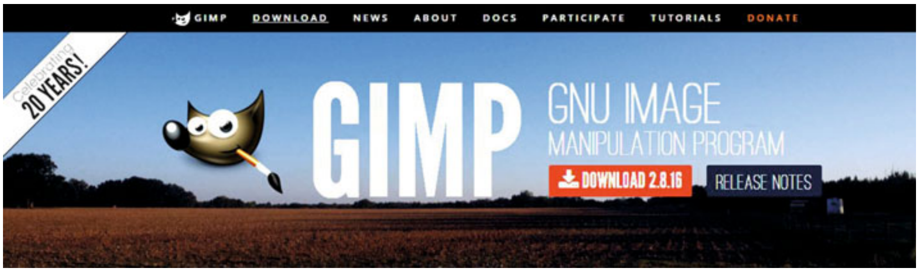


Figure B-20. *Go to `www.gimp.org`, and download GIMP 2.8.16*

If you want to learn digital image compositing, check out *Digital Image Compositing Fundamentals* (Apress, 2015).

Blender: 3D Modeling, Rendering, and Animation

The Blender Foundation project offers a professional i3D software package called Blender that allows you to do modeling of 3D objects as well as rendering and animation. Download this software package at `www.blender.org`, and install it. Blender's home page and blue download button are shown in Figure B-21.

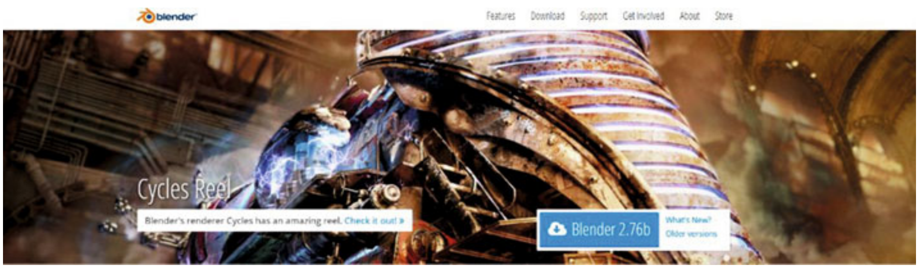
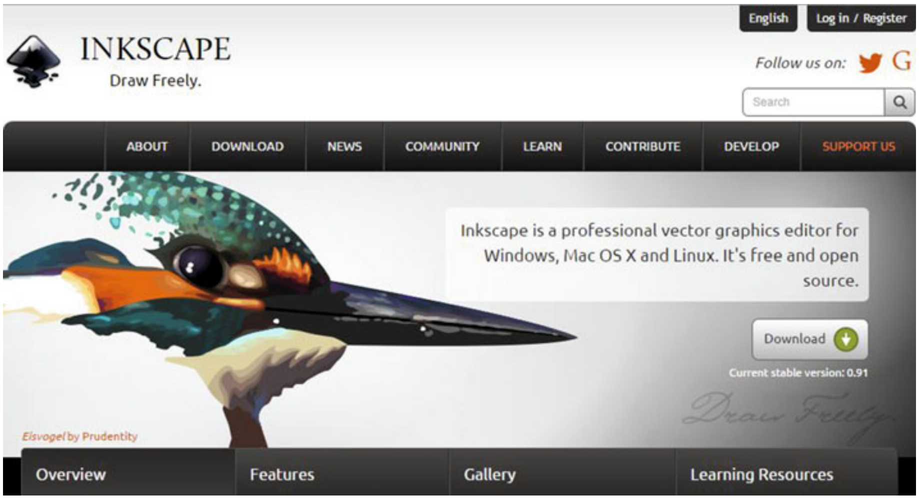


Figure B-21. *Go to `blender.org`, and download the latest version*

This is a professional-level software package with many of the same features as 3D Studio Max, Maya, XSI, and Lightwave.

Inkscape: Digital Illustration and Digital Painting

The Inkscape Project offers a professional digital illustration software package called Inkscape that can also do digital painting. Download this software package at `www.inkscape.org`. Inkscape's home page and download button are shown in Figure B-22.



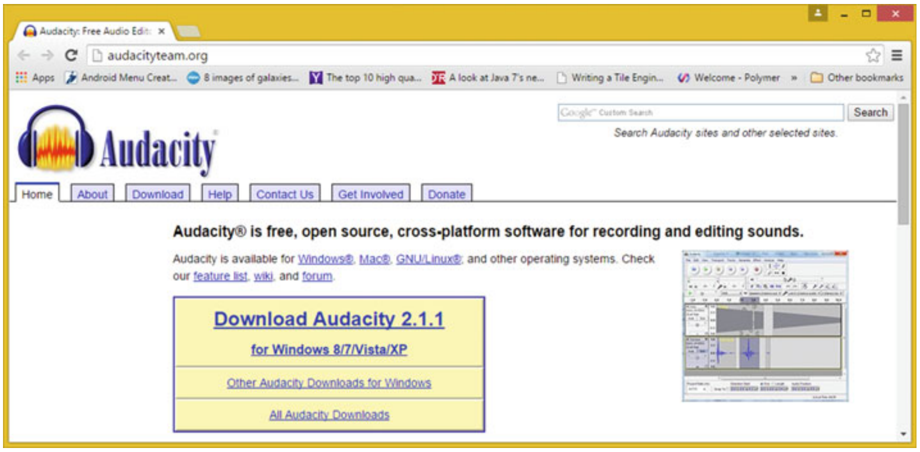
Inkscape is a professional vector graphics editor for Windows, Mac OS X and Linux. It's free and open source.

Figure B-22. *Go to* `inkscape.org`*, and download the latest version*

If you want to learn digital illustration, check out *Digital Illustration Fundamentals* (Apress, 2015), as well as *Digital Painting Techniques* (Apress, 2015).

Audacity: Digital Audio Editing and Special Effects

The Audacity team offers a professional digital audio software package called Audacity that is for digital audio editing, sweetening, and special effects. You can download this software package at `www.audacityteam.org`. The Audacity home page and Download link are shown in Figure B-23.



Audacity® is free, open source, cross-platform software for recording and editing sounds.

Audacity is available for Windows®, Mac®, GNU/Linux® and other operating systems. Check our feature list, wiki, and forum.

Figure B-23. *Go to `audacityteam.org`, and download version 2.1.2*

Audacity offers many of the same digital audio editing features as many professional audio editors, and it is adding 64-bit capabilities and professional features every month. The next version will have a more professional user interface look and feel. If you want to learn more about digital audio editing, synthesis, and special effects, check out *Digital Audio Editing Fundamentals* (Apress, 2015). Digital audio can greatly enhance the user experience for any of the JSON applications you create in the future.

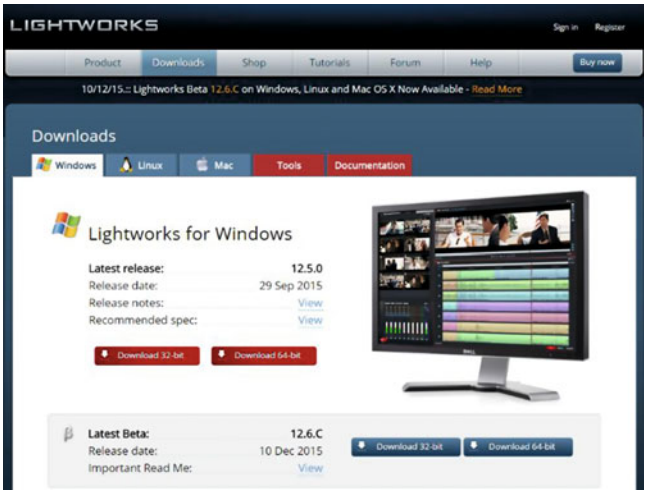
Visual Effects: BlackMagic Design Fusion 8.0 VFX

BlackMagic Design's Fusion 8 used to cost thousands of dollars; it offers a professional visual effects (VFX) software package used in film and television. Download this software package at `www.blackmagicdesign.com/products/fusion/`. Fusion 8's home page and blue Download button are shown in Figure B-24. If you want to learn more about VXF pipelines, check out *Visual Effects (VFX) Fundamentals* (Apress, 2016).

Figure B-24. *Go to `blackmagicdesign.com`, and download Fusion 8*

Digital Video Editing: EditShare Lightworks 12.6

EditShare Lightworks offers professional digital video editing in a software package that also does special effects. Download the software package at `www.lwks.com` after signing up for the download. Lightworks' home page and **Downloads** tab are shown in Figure B-25, where you can select your OS version. I recommend using a 64-bit OS and software so that you can use 8MB of memory!



10/12/15.:: Lightworks Beta 12.6.C on Windows, Linux and Mac OS X Now Available - Read More

Lightworks for Windows

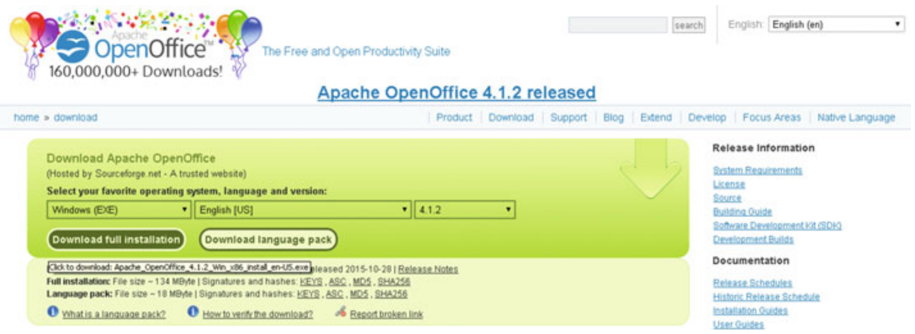
Latest release:	12.5.0
Release date:	29 Sep 2015
Release notes:	View
Recommended spec:	View

Latest Beta:	12.6.C
Release date:	10 Dec 2015
Important Read Me:	View

Figure B-25. *Download Lightworks 12.6 for your OS*

Office Productivity Suite: Apache OpenOffice 4.1.2

Apache OpenOffice, originally Sun Microsystems' StarOffice, was acquired by Oracle and released as open source. This will provide your JSON development business with professional office and business productivity software support. Download this great software package at `www.openoffice.com`. The Apache OpenOffice home page and Download button are shown in Figure B-26, where you can select your OS, language, and software version. I recommend using a 64-bit OS and software so that you can use 8MB of memory.


Apache OpenOffice
160,000,000+ Downloads!
The Free and Open Productivity Suite
search
English: English (en)
home » download
Product | Download | Support | Blog | Extend | Develop | Focus Areas | Native Language
Download full installation
Download language pack

Apache OpenOffice 4.1.2 released

Download Apache OpenOffice
(Hosted by Sourceforge.net - A trusted website)
Select your favorite operating system, language and version:
Windows (EXE) | English [US] | 4.1.2

Click to download: Apache_OpenOffice_4.1.2_Win_x86_install_en-US.exe eleased 2015-10-28 | Release Notes
Full installation: File size ~ 134 MByte | Signatures and hashes: KEYS , ASC , MD5 , SHA256
Language pack: File size ~ 18 MByte | Signatures and hashes: KEYS , ASC , MD5 , SHA256
What is a language pack? How to verify the download? Report broken link

Release Information
System Requirements
License
Source
Building Guide
Software Development Kit (SDK)
Development Builds

Documentation
Release Schedules
Historic Release Schedule
Installation Guides
User Guides

Figure B-26. *Download the Apache OpenOffice 4.1.2 full installation*

Summary

In this appendix, you set up your JSON workstation by downloading and installing the open source Java 8 JDK and Eclipse Mars IDE software you should use to code JSON applications. I also showed you some professional new media and business software that is free for commercial use. I recommend installing all of these packages so you have them on your development workstation if and when you need them for your JSON-capable applications development workflow.

APPENDIX C

IntelliJ IDEA: Setting Up a JSON Integrated Development IDE

In this appendix, let's put together your foundation for a highly professional, JSON-friendly, IntelliJ IDEA integrated development environment (IDE). Your development workstation is the most important combination of PC hardware and software, allowing you to reach your goal of JSON-compatible application development. This appendix considers your hardware needs and the software infrastructure to put together a professional, well-rounded JSON workstation that gives you a bunch of arrows in your software development quiver right off the bat. You will then have everything you need when you're reading the book's chapters, no matter what type of JSON application you decide that you want to develop for your end users!

All readers of this book should be developing with identical JSON application software development environments, because everything you learn over the course of this book needs to be able to be experienced equally by everyone. Appendix A outlines all the steps to put together a NetBeans 8.1 IDE based JSON development workstation, and Appendix B does the same for Eclipse Mars. These three IDE-customized appendixes get all the tedious setup tasks out of the way. If you already have your workstation configured, you can proceed to Chapter 2 for an overview of JSON; or, if you are already familiar with JSON, you're ready to read the rest of the book.

In this appendix, you learn where to download and how to install several of the most impressive, professional, open source software packages on the face of this planet. You are about to max out your JSON, HTML5, CSS3, and JavaScript (JS) development workstation, so hold on tight and enjoy this virtual download ride!

Creating a JSON Development Workstation

The first thing you'll do after taking a look at hardware requirements is to download and install the entire Java software development kit (SDK), which Oracle calls Java SE 8 Java Development Kit (JDK). IntelliJ IDEA, which is called the IntelliJ IDE, uses Java 8 Standard Edition (SE).

W. Jackson, *JSON Quick Syntax Reference*, DOI 10.1007/978-1-4842-1863-1

The second thing you'll download and install is the IntelliJ IDEA, which you can get at the http://www.jetbrain.org web site. IntelliJ IDEA (Integrated Development Environment App or Integrated Development Environment Advanced) allows you to develop JSON compatible applications with all the popular programming languages, including Java EE, Java SE, JavaScript, HTML5, CSS3, JavaFX, Android Studio 2, XML, XSL, PHP, and SQL.

After your JSON application development environment is set up, you can then download and install new media asset development tools, if you wish. These are used in conjunction with NetBeans 8.1 for things such as image editing (GIMP) and nonlinear digital video editing (Lightworks); special effects (Fusion); digital audio sweetening, or editing (Audacity); i3D modeling, rendering, and animation (Blender); digital illustration (Inkscape); and business productivity (OpenOffice).

This appendix should take your JSON development to an all-new level, showing you how to create a media development and programming workstation that will run your JSON business. All the software development tools you'll download and install come close to matching all the primary feature sets of expensive paid software packages, such as those from Apple (Final Cut Pro), Autodesk (3D Studio Max), Adobe (Photoshop, Illustrator, After Effects), and Avid (ProTools), and all at *zero* cost to your production company!

Open source software is free to download, install, and upgrade and is continually adding features. It's becoming more and more like professional software every day. You will be amazed at how professional open source software packages have become over the last decade or so.

Hardware Foundation

Because in this appendix you put together the foundation for the JSON-capable application development workstation you use throughout this book, I want to take a moment to review Eclipse Mars' JSON development workstation hardware requirements. This is a factor that will influence your development performance (speed). This is clearly as important as the software itself, because hardware is what is actually running the software package's algorithms.

Minimum requirements for IntelliJ IDEA include 1 GB of memory, 300MB of hard disk space, and a Java 6 JDK, or higher. Next, let's discuss what you need to make your IntelliJ JSON IDE usable. Start by upgrading your 1280×768 WXGA display to an HDTV (1920×1080 at 120FPS refresh rate) or UHD (4096×2160 at 120FPS refresh rate) widescreen display. These are now affordable and give you 3 to 12 times the display real estate of a WXGA display. HDTVs are now $250 to $500, and UHDTV displays are under $1,000.

I recommend using, at a bare minimum, the **Intel i7** quad-core processor or the **AMD 64-bit** octa-core processor. Install at least 8GB of DDR3-1600 memory. I'm using a 64-bit, octa-core AMD 8350, with 16GB of DDR3-1600. Intel also has a hexa-core i7 processor. This would be the equivalent of having 12 cores, because each i7 core can host two threads. Similarly, an i7 quad-core should look like 8 cores to your 64-bit operating system's thread-scheduling algorithm.

There are also high-speed DDR3-1800 as well as DDR3-2133 clock-speed memory module components available. A high number signifies fast memory-access speeds. To calculate actual megahertz speeds at which the memory is cycling, divide the number by 4 (1333 = 333Mhz, 1600 = 400Mhz, 1800 = 450Mhz, 2133 = 533Mhz). Memory-access speed is a massive workstation performance factor, because your processor is usually bottlenecked by the speed at which processor cores can access the data (in memory) that that processor core needs to process.

With high-speed processing and memory access going on in the workstation while it is operating, it's extremely important to keep everything cool so that you do not experience **thermal problems**. I recommend using a wide, full-tower enclosure with 120mm or 200mm cooling fans (one or two at least), as well as a captive liquid-induction cooling fan on the CPU. It is important to note that the cooler the system runs, the faster it can run, and the longer it will last, so load the workstation up with lots of silent high-speed fans!

If you really want maximum performance, install a solid state disk (SSD) drive as the primary disk drive from which your applications and operating system software can load. Use legacy HDD hardware for your `D:\` hard drive for slower data storage (long-term). Put your current project files on the SSD.

As far as OS goes, I am using a 64-bit Windows 8 operating system that is fairly memory efficient. The Linux 64-bit OS is extremely memory efficient. I recommend using any 64-bit OS so you can address more than 3.24GB of system memory; this is a limitation with a 32-bit operating system that does not exist once you have upgraded to 64-bit OS and have that full 64-bits of memory-addressing headroom.

Open Source Software

To create a well-rounded, professional JSON application development workstation, you'll install all the primary genres of open source software. First you will install Java SE 8 and Eclipse Mars. I will also show you how to download GIMP, Lightworks, Fusion, Blender3D, and Audacity, which are also all open source software packages, in case your JSON applications will use a graphical front end. I also recommend other free software at the end of the appendix so you can put together the major production workstation you have always dreamed of.

Open source software is approaching the level of professionalism of paid development software packages that cost thousands of dollars each. Using open source software packages like Java 8, NetBeans 8, Blender, GIMP, Audacity, Lightworks, Fusion, OpenOffice, and others, you can put together a free application development workstation and rival paid software workstations.

If you just purchased a new JSON development workstation PC and are going to put together the entire development software suite from scratch, this appendix goes through the entire work process.

Java 8: Installing the Foundation for IntelliJ IDEA

The first thing to do is visit the IntelliJ web site at `http://www.jetbrains.com/idea/`. Click the black Download button in the center of the IntelliJ IDEA home page, as shown in Figure C-1.

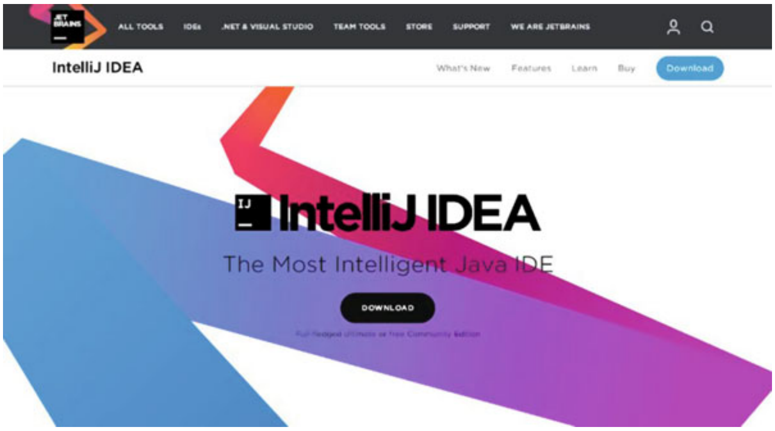
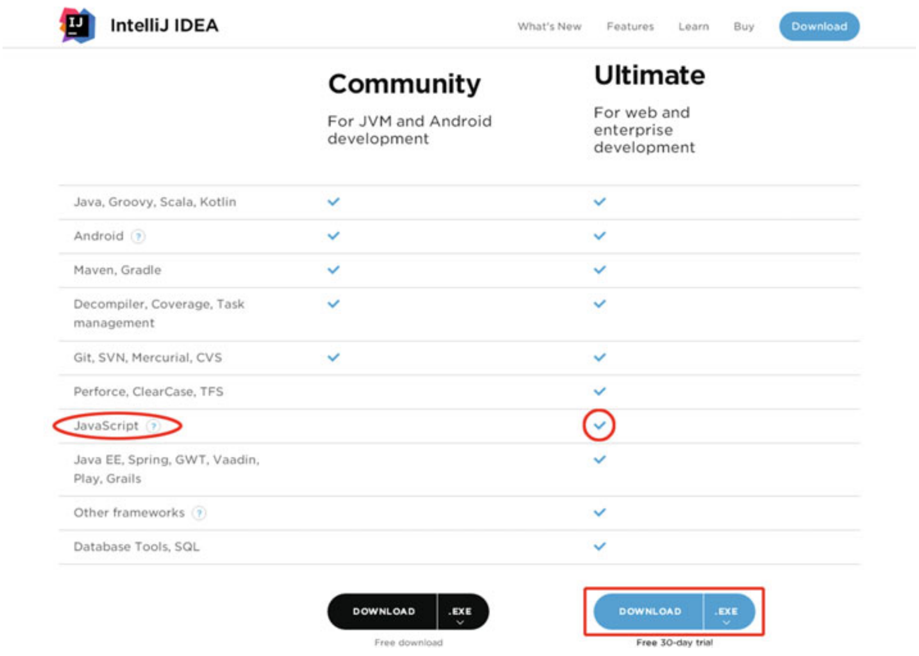


Figure C-1. *Load `jetbrains.com/idea/`, and click Download*

As you can see, there are two different download options, but only one that supports JavaScript, which JSON is based on. I am therefore forced to suggest the Ultimate version! This supports all the popular programming languages that work with JSON. If you don't want to purchase IntelliJ IDEA Ultimate, use the 30-day trial version, or use NetBeans 8.1 (see Appendix A) or Eclipse 4.5.1 Mars (see Appendix B), both of which are free. The two different versions of IntelliJ IDEA are shown in Figure C-2 along with the programing features they support.



	Community For JVM and Android development	Ultimate For web and enterprise development
Java, Groovy, Scala, Kotlin	✓	✓
Android	✓	✓
Maven, Gradle	✓	✓
Decompiler, Coverage, Task management	✓	✓
Git, SVN, Mercurial, CVS	✓	✓
Perforce, ClearCase, TFS		✓
JavaScript		✓
Java EE, Spring, GWT, Vaadin, Play, Grails		✓
Other frameworks		✓
Database Tools, SQL		✓
	DOWNLOAD .EXE Free download	DOWNLOAD .EXE Free 30-day trial

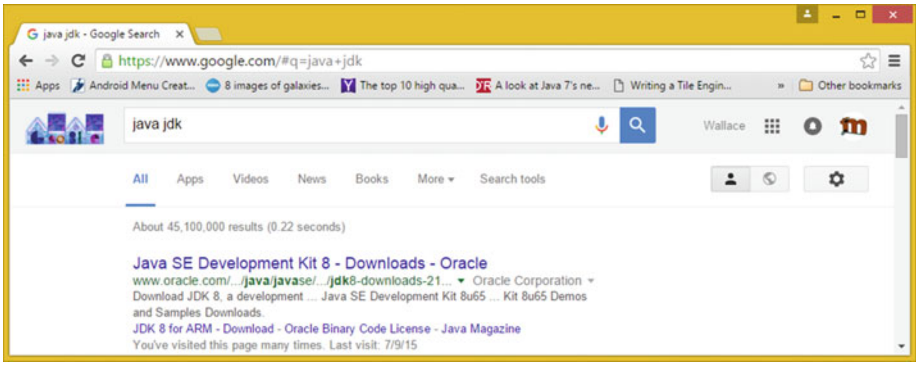
Figure C-2. *Download the IntelliJ Ultimate 30-day trial version*

If you're wondering why the IntelliJ IDEA download isn't offered in 32-bit and 64-bit versions, this is because the IDEs that do this have both versions precompiled, whereas IntelliJ IDEA requires the Java 6 (or later Java 8) JDK to be installed. This tells you that if you want to use the IntelliJ IDEA version, so that any Java or web programming languages you want to use JSON with are supported, you have to first install Java SE 6 or later. In this case, this should be Java 8. This is very similar to what you see using NetBeans 8.1 (see Appendix A); both of these use a Java bytecode `.JAR` file to run off of, by using the JRE.

This approach allows the bit-versions to be handled by a JRE bit-version, so be sure to install the correct JDK version: 32-bit for Windows Vista and XP or 64-bit for Windows 7, 8.1, or 10. Note that there are other bit-versions of these OSs, and I am just generalizing here; XP and Vista were 32-bit, and Windows 7, 8, and 10 are 64-bit, due to when they came out.

Dowload IntelliJ. After that is completed, go get the latest Java 8 SE JDK, so you can run IntelliJ.

Open Google Chrome, and Google "Java JDK", as shown in Figure C-3. Look for the Java SE Development Kit 8 - Downloads search result, and click it to open the Oracle Java web site.

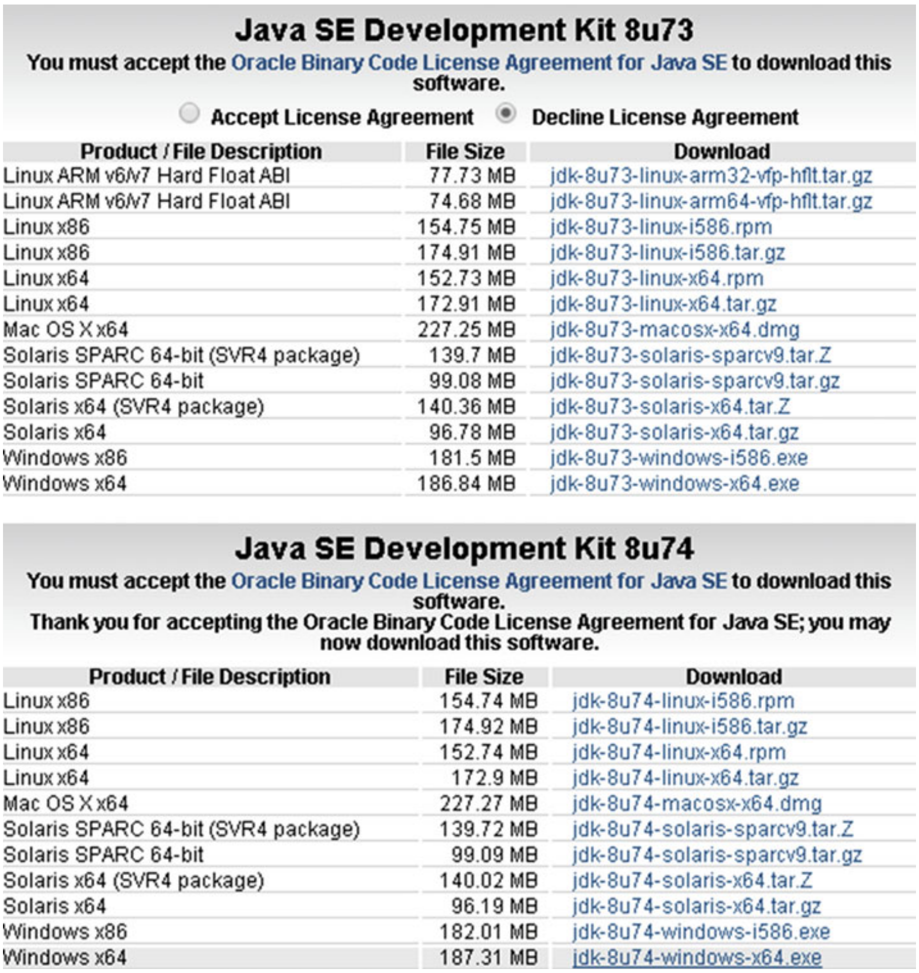


About 45,100,000 results (0.22 seconds)

Java SE Development Kit 8 - Downloads - Oracle
www.oracle.com/.../java/javase/.../jdk8-downloads-21... ▾ Oracle Corporation ▾
Download JDK 8, a development ... Java SE Development Kit 8u65 ... Kit 8u65 Demos and Samples Downloads.
JDK 8 for ARM - Download - Oracle Binary Code License - Java Magazine
You've visited this page many times. Last visit: 7/9/15

Figure C-3. *Google "Java JDK", and then click the Downloads link*

Download and install the latest Java 8 JDK, which at the time I wrote this book was Java SE 8u74, as shown in Figure C-4.



Java SE Development Kit 8u73

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement [✓ Decline License Agreement]

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.73 MB	jdk-8u73-linux-arm32-vfp-hflt.tar.gz
Linux ARM v6/v7 Hard Float ABI	74.68 MB	jdk-8u73-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.75 MB	jdk-8u73-linux-i586.rpm
Linux x86	174.91 MB	jdk-8u73-linux-i586.tar.gz
Linux x64	152.73 MB	jdk-8u73-linux-x64.rpm
Linux x64	172.91 MB	jdk-8u73-linux-x64.tar.gz
Mac OS X x64	227.25 MB	jdk-8u73-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.7 MB	jdk-8u73-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.08 MB	jdk-8u73-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.36 MB	jdk-8u73-solaris-x64.tar.Z
Solaris x64	96.78 MB	jdk-8u73-solaris-x64.tar.gz
Windows x86	181.5 MB	jdk-8u73-windows-i586.exe
Windows x64	186.84 MB	jdk-8u73-windows-x64.exe

Java SE Development Kit 8u74

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	154.74 MB	jdk-8u74-linux-i586.rpm
Linux x86	174.92 MB	jdk-8u74-linux-i586.tar.gz
Linux x64	152.74 MB	jdk-8u74-linux-x64.rpm
Linux x64	172.9 MB	jdk-8u74-linux-x64.tar.gz
Mac OS X x64	227.27 MB	jdk-8u74-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.72 MB	jdk-8u74-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.09 MB	jdk-8u74-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.02 MB	jdk-8u74-solaris-x64.tar.Z
Solaris x64	96.19 MB	jdk-8u74-solaris-x64.tar.gz
Windows x86	182.01 MB	jdk-8u74-windows-i586.exe
Windows x64	187.31 MB	jdk-8u74-windows-x64.exe

Figure C-4. *The Oracle TechNetwork Java SE JDK Download web site*

The URL is in the address bar in Figure C-4 and opens the download page for Java SE Development Kit 8u73 and 8u74. I will put this link here as well, in case you want to simply cut and paste it, copy it in, or click it to launch the site: `www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`.

Pull the scrollbar halfway down the page to display the Java 8 SE Development Kit 8u74 (or a later version) download links table, as can be seen at the bottom of Figure C-4. You can also click the links above the table to read the explanation of the new CPU and PSU Java release versions; you're going to use the latest Java 8u74 version.

Once you click Accept License Agreement, the links in the table become bolded and you can click the link you wish to use. If you are on Windows and your OS is 64-bit, use the Windows x64 link; otherwise, use the Windows x86 link. I am using what is described in these links as Windows x64, which is the 64-bit version of Windows, for my Windows 7 and Windows 8.1 workstations.

Make sure you use this Java SE Development Kit 8u74 download link, and not the Java Runtime Edition (JRE) link. This JRE is part of the JDK 8u74, so you do not have to worry about getting the Java Runtime separately.

In case you are wondering, you indeed use this JRE to launch and run the IntelliJ IDE. You'll use the JDK in that IntelliJ software package to provide the Java SE 8 core class foundation that is used as the foundation for JavaFX and for Android's Java-based API classes.

Before you run this installation, you should remove older versions of Java. In Windows Control Panel, use **Add Or Remove Programs** (XP) or **Programs And Features** (Windows Vista, 7, 8, or 10), as shown selected in Figure C-5. This opens a Windows utility that manages installed software packages, allowing you to remove them from your OS.

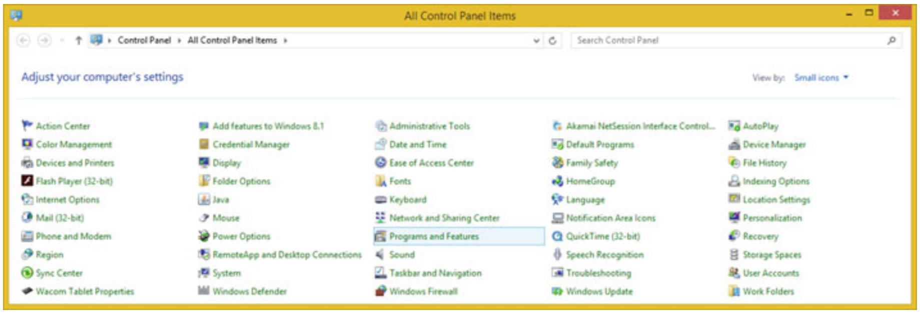
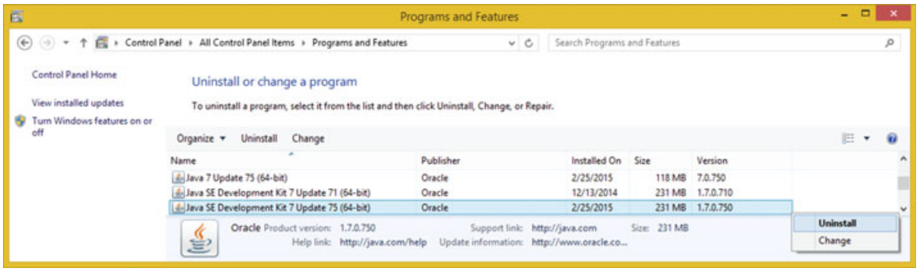


Figure C-5. *Launch your Control Panel, and choose Programs And Features*

This is necessary especially if your workstation is not brand new. You do this so that only the latest Java SE 8u74 and JRE 8u74 versions are currently installed on your JSON development workstation.

Select all the older Java versions, right-click each one, and select the Uninstall option, as shown in Figure C-6.


Programs and Features
Control Panel › All Control Panel Items › Programs and Features
Search Programs and Features
Control Panel Home
View installed updates
Turn Windows features on or off
Organize ▾
Uninstall
Change
Uninstall
Change

Uninstall or change a program

To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Name	Publisher	Installed On	Size	Version
Java 7 Update 75 (64-bit)	Oracle	2/25/2015	118 MB	7.0.750
Java SE Development Kit 7 Update 71 (64-bit)	Oracle	12/13/2014	231 MB	1.7.0.710
Java SE Development Kit 7 Update 75 (64-bit)	Oracle	2/25/2015	231 MB	1.7.0.750

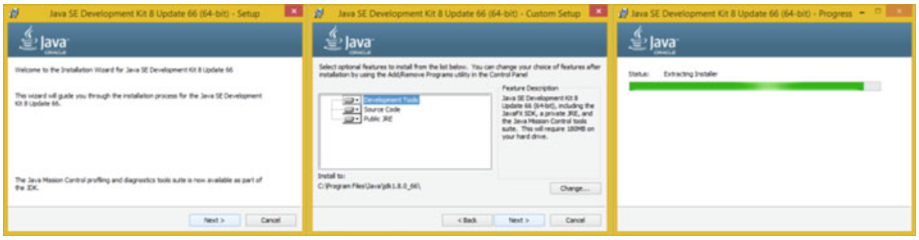
Oracle Product version: 1.7.0.750 Support link: http://java.com Size: 231 MB
Help link: http://java.com/help Update information: http://www.oracle.co...

Figure C-6. *Find old versions of Java, right-click, and choose Uninstall*

As you can see, I am more than 100 versions old, because I am at Java 7, update 71! If you install different versions of the Java SE JDK on your system, they will not replace each other and will instead exist in parallel, or next to each other.

The reason is that you may have older projects and software (such as IDEs), which use these older versions of Java without crashing. For instance, Android 4.4 and earlier use the Java 6 SDK, Android 5.x or 6.x (64-bit Android) use Java 7 SDK, and JavaFX and everything else use Java 8 and, soon, Java 9!

Once you have done this and downloaded the installation executable, locate it, and install Java SE 8u74 JDK on your system by double-clicking the `.EXE` file. Doing so launches a Setup dialog, seen at left in Figure C-7. You can also right-click your installer file and then select the Run As Administrator option, ensuring that you have proper file access.


Java SE Development Kit 8 Update 66 (64-bit) - Setup
Java
Next >
Cancel
Java SE Development Kit 8 Update 66 (64-bit) - Custom Setup
Java
Development Tools
Source Code
Public JRE
Feature Description
Change...
< Back
Next >
Cancel
Java SE Development Kit 8 Update 66 (64-bit) - Progress
Java

Welcome to the Installation Wizard for Java SE Development Kit 8 Update 66

This wizard will guide you through the installation process for the Java SE Development Kit 8 Update 66.

The Java Mission Control profiling and diagnostics tools suite is now available as part of the JDK.

Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

Java SE Development Kit 8 Update 66 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.

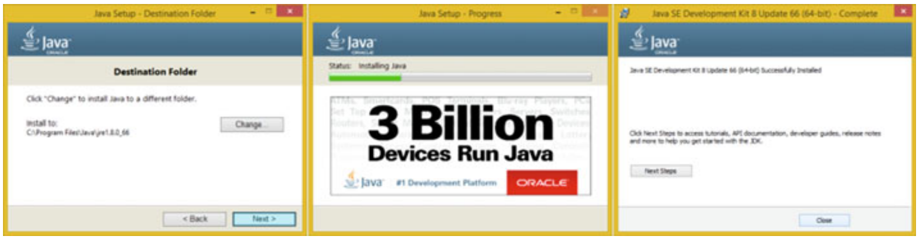
Install to:
C:\Program Files\Java\jdk1.8.0_66\

Status: Extracting Installer

Figure C-7. *Setup, Custom Setup, and Extracting Installer dialogs*

Click Next to access the **Custom Setup** dialog, shown in the middle of Figure C-7. Accept the default settings, and then click the button to access the **Extracting Installer** progress dialog shown at right in Figure C-7.

Once you've extracted the installation software, you can select a Java JDK software installation folder. Use the default `C:\ProgramFiles\Java\jre1.8.0_74` in the Destination Folder dialog, as shown at left in Figure C-8.



Click "Change" to install Java to a different folder.

Install to:
C:\Program Files\Java\jre1.8.0_66

Status: Installing Java

Java SE Development Kit 8 Update 66 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

Figure C-8. *Destination, Progress, and Complete Install dialogs*

Click Next to install a JRE edition in the default specified folder. Interestingly, the installer won't ask you to specify the JDK folder name for some reason, probably because it wants your Java JDK to always be in a set or fixed (locked in the same location) folder.

The JDK folder is named `C:\ProgramFiles\Java\jdk1.8.0_74`. Notice that internally, Java 8 is referred to as Java 1.8.0. Thus Java 6 should be 1.6.0, and Java 7 is 1.7.0. This is useful to know, in case you are looking for Java versions using a search utility, for example, or just to show off your knowledge of legacy Java version numbering.

Once you click **Next**, you get the Java **Setup Progress** dialog shown in the middle of Figure B-8. Once Java 8 is finished installing, you finally see the **Complete** dialog, at right in Figure C-8. Congratulations! You have successfully installed Java 8!

Remember that the reason that you did not download a JRE is because it is part of this JDK 8u74 installation. The Java 8 Runtime Edition is the executable (platform) which willthat runs the Java software app once it has been compiled into an application. and also the latest JRE will be is also needed to run NetBeans 8, IntelliJ and Eclipse, which, as you now know, is 100% completely written using the Java SE 8 development platform.

Once Java 8u74 or later is installed on your workstation, you can then download and install the latest IntelliJ software installer from `www.jetbrains.com/idea/`.

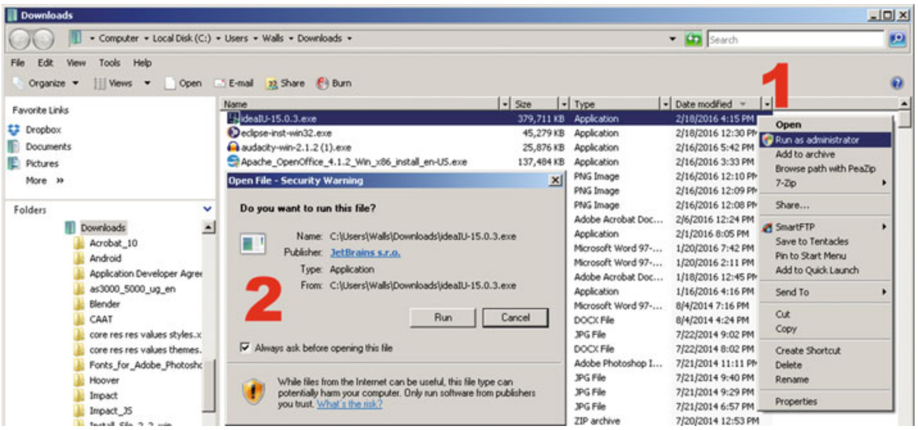
You can use the same Programs And Features (or Add Or Remove Programs) utility in your Control Panel to remove any older versions of the NetBeans development environment that may be currently installed on your JSON development workstation.

Now you are ready to add the second layer of the IntelliJ IDEA IDE software, which runs on top of the Java 8 environment you have now installed.

IntelliJ IDEA: Downloading the IntelliJ IDEA for JSON

The second step in this process is to install IntelliJ, which you downloaded earlier, as shown in Figure C-2. This starts a download function, which should put the `ideaIU-15.0.3.exe` file into your `C:\Users\Your-Name-Goes-Here\Downloads\` folder.

Find this executable file on your workstation, and either double-click it or right-click it, and select the Run As Administrator option. This opens a Security Warning: "Do you want to run this file?" dialog, shown at left in Figure C-9. The two steps shown are numbered in red.



Name	Size	Type	Date modified
ideaIU-15.0.3.exe	379,711 KB	Application	2/18/2016 4:15 PM
eclipse-inst-win32.exe	45,279 KB	Application	2/18/2016 12:30 PM
audacity-win-2.1.2 (1).exe	25,876 KB	Application	2/16/2016 5:42 PM
Apache_OpenOffice_4.1.2_Win_x86_install_en-US.exe	137,484 KB	Application	2/16/2016 3:33 PM

Open File - Security Warning

Do you want to run this file?

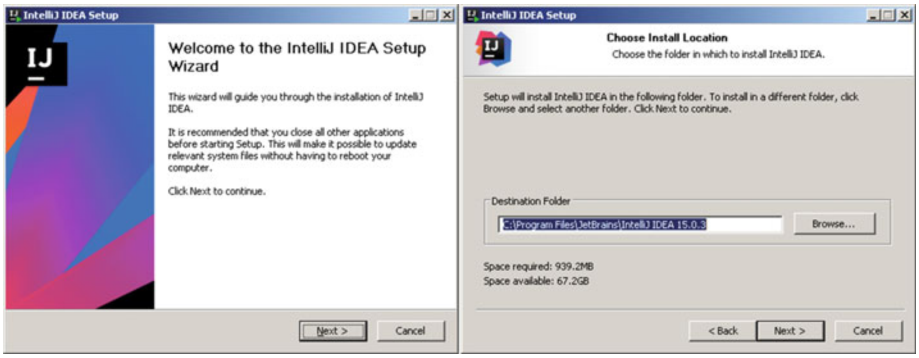
Name: C:\Users\Walls\Downloads\ideaIU-15.0.3.exe
Publisher: JetBrains s.r.o.
Type: Application
From: C:\Users\Walls\Downloads\ideaIU-15.0.3.exe

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Figure C-9. *Right-click IDEA Installer and Run As Administrator*

Click the Run button, and launch the installer. This opens the IntelliJ IDEA Setup Wizard dialog, shown in Figure C-10. Click Next to continue, as the dialog instructs you. In the Choose Install Location dialog, select a default value provided for the Destination Folder data field. Click Next to proceed with your install.



Welcome to the IntelliJ IDEA Setup Wizard

This wizard will guide you through the installation of IntelliJ IDEA.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Choose Install Location
Choose the folder in which to install IntelliJ IDEA.

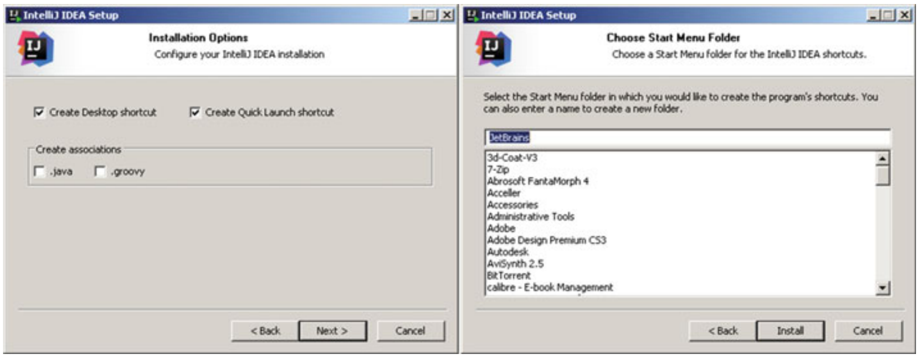
Setup will install IntelliJ IDEA in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\JetBrains\IntelliJ IDEA 15.0.3

Space required: 939.2MB
Space available: 67.2GB

Figure C-10. *Start your install, and accept the default location*

In the Installation Options dialog, select the shortcuts you want to have created for you, and specify an association for Java and Groovy files that you wish to have put into your system registry, as shown in Figure C-11. Click Next.In the Choose Start Menu Folder dialog, select the default JetBrains Start Menu folder name. Then click the Install button to start your installation process.



IntelliJ IDEA Setup

Installation Options
Configure your IntelliJ IDEA installation

[✓] Create Desktop shortcut   [✓] Create Quick Launch shortcut

Create associations
[ ] .java   [ ] .groovy

< Back | Next > | Cancel

IntelliJ IDEA Setup

Choose Start Menu Folder
Choose a Start Menu folder for the IntelliJ IDEA shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

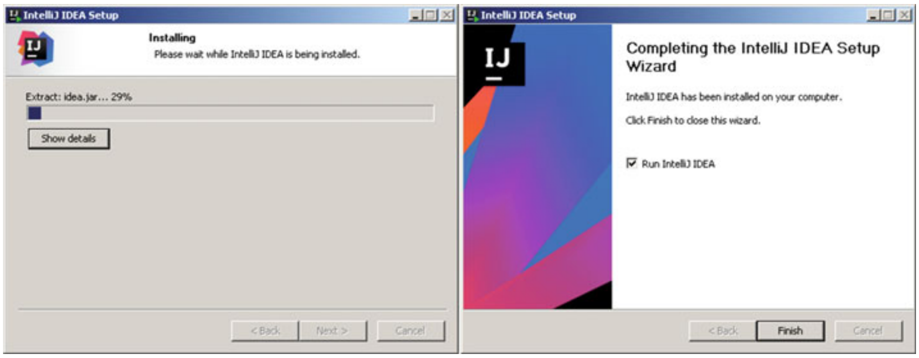
JetBrains

3d-Coat-V3
7-Zip
Abrosoft FantaMorph 4
Acceller
Accessories
Administrative Tools
Adobe
Adobe Design Premium CS3
Autodesk
AviSynth 2.5
BitTorrent
calibre - E-book Management

< Back | Install | Cancel

Figure C-11. *Installation Options and Choose Start Menu Folder dialogs*

The Installing dialog shows you a progress bar along with the files that are being installed and the percentage complete for each one, as shown at left in Figure C-12. Once the IDEA installation is complete, you see the Completing the IntelliJ IDEA Setup Wizard dialog shown at right in Figure C-12. Notice that I have selected the Run IntelliJ IDEA option so I can show you the IDE itself.



IntelliJ IDEA Setup

Installing
Please wait while IntelliJ IDEA is being installed.

Extract: idea.jar... 29%

Show details

< Back | Next > | Cancel

IntelliJ IDEA Setup

Completing the IntelliJ IDEA Setup Wizard

IntelliJ IDEA has been installed on your computer.

Click Finish to close this wizard.

[✓] Run IntelliJ IDEA

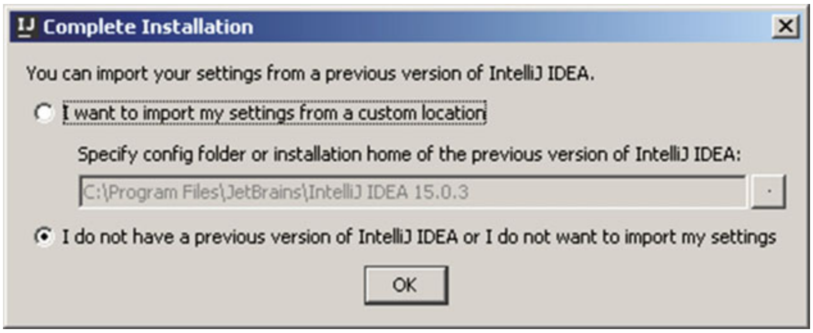
< Back | Finish | Cancel

Figure C-12. *Select Run IntelliJ IDEA after install completes*

When you click the Finish button to exit the install, you see the IntelliJ IDEA 15 loading screen. This is shown every time you start IntelliJ IDEA; see Figure C-13.

Figure C-13. *IntelliJ IDEA 15 software loading startup screen*

If you have a previous version of IntelliJ IDEA, you can import its settings using the Complete Installation dialog, shown in Figure C-14.



Complete Installation

You can import your settings from a previous version of IntelliJ IDEA.

I want to import my settings from a custom location

Specify config folder or installation home of the previous version of IntelliJ IDEA:

C:\Program Files\JetBrains\IntelliJ IDEA 15.0.3

[✓ I do not have a previous version of IntelliJ IDEA or I do not want to import my settings]

Figure C-14. *Complete Installation dialog*

Click the OK button, and you see the IDEA License Activation dialog shown in Figure C-15. If you want to purchase IntelliJ IDEA, click Buy IntelliJ IDEA, and enter an activation code in the data field. Otherwise, click “Evaluate for free for 30 days” at right.

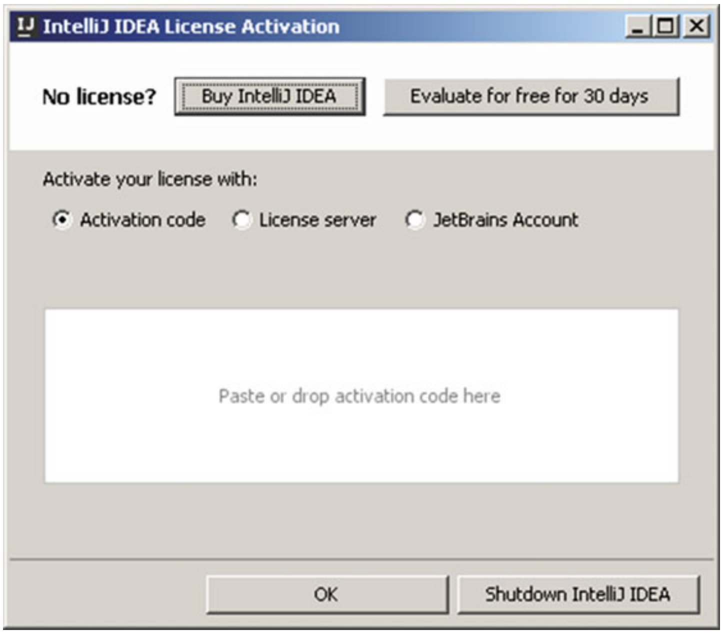
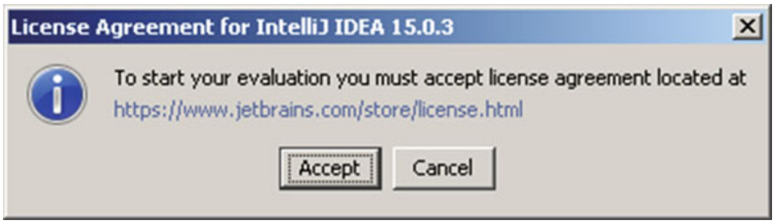

IntelliJ IDEA License Activation
No license?
Buy IntelliJ IDEA
Evaluate for free for 30 days
Activate your license with:
Activation code
License server
JetBrains Account
Paste or drop activation code here
OK
Shutdown IntelliJ IDEA

Figure C-15. *IDEA License Activation*

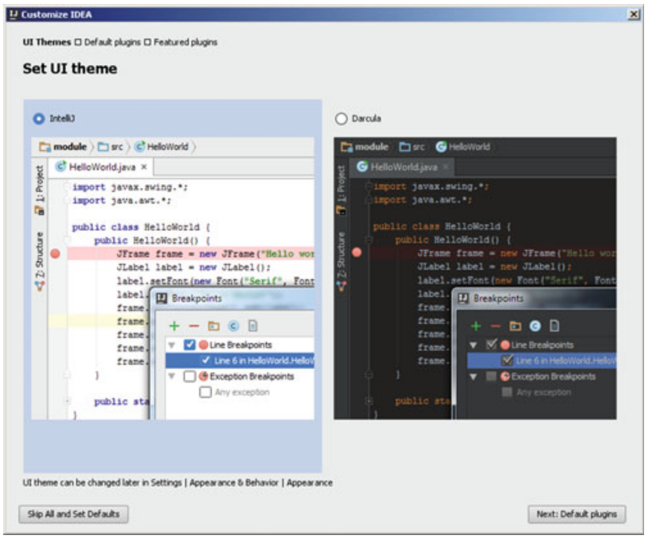
In the next dialog, click Accept to accept the licensing agreement at the JetBrains web site's store, so that you can use the IDEA software for JSON development (see Figure C-16).



To start your evaluation you must accept license agreement located at https://www.jetbrains.com/store/license.html

Figure C-16. *License Agreement dialog*

Accept the default IntelliJ UI theme or choose the dark version, and click Next, as shown in Figure C-17.



Set UI theme

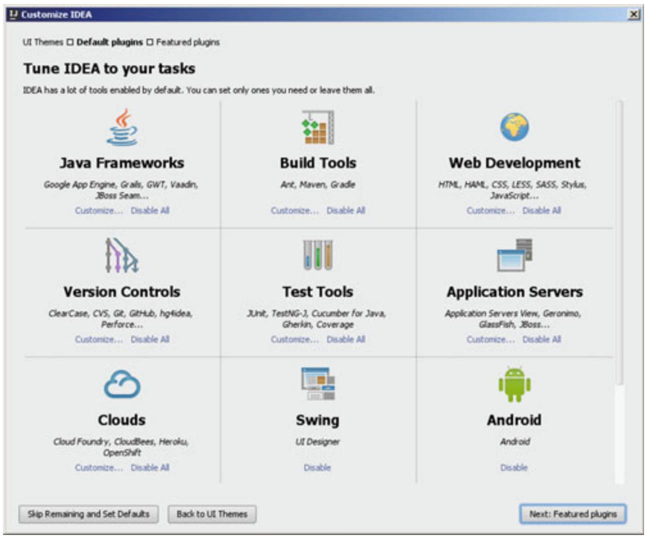
[✓ Intellij] Darcula

```
import javax.swing.*;
import java.awt.*;

public class HelloWorld {
    public HelloWorld() {
        JFrame frame = new JFrame("Hello wor
        JLabel label = new JLabel();
        label.setFont(new Font("Serif", Font
        label.
        frame.
        frame.
        frame.
        frame.
        frame.
    }

    public sta
}
```

UI theme can be changed later in Settings | Appearance & Behavior | Appearance

Figure C-17. *IntelliJ Set UI Theme dialog*

I left all the IntelliJ capabilities enabled and then clicked the Next: Featured Plugins button shown in Figure C-18.



Tune IDEA to your tasks

IDEA has a lot of tools enabled by default. You can set only ones you need or leave them all.

Java Frameworks — *Google App Engine, Grails, GWT, Vaadin, JBoss Seam...*

Build Tools — *Ant, Maven, Gradle*

Web Development — *HTML, HAML, CSS, LESS, SASS, Stylus, JavaScript...*

Version Controls — *ClearCase, CVS, Git, GitHub, hg4idea, Perforce...*

Test Tools — *JUnit, TestNG-J, Cucumber for Java, Gherkin, Coverage*

Application Servers — *Application Servers View, Geronimo, GlassFish, JBoss...*

Clouds — *Cloud Foundry, CloudBees, Heroku, OpenShift*

Swing — *UI Designer*

Android — *Android*

Figure C-18. *Tune IDEA To Your Tasks dialog*

If there are any featured plug-ins you want to add to IntelliJ, select them in the Download Featured Plugins dialog shown in Figure C-19, and click Start Using IntelliJ IDEA.

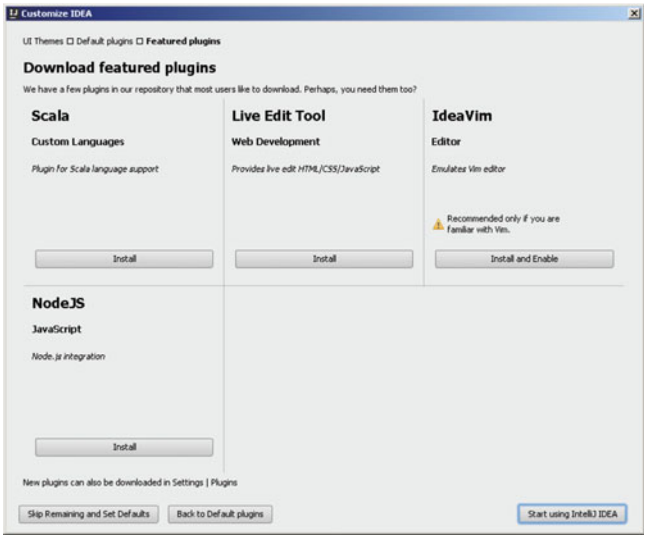

Customize IDEA
UI Themes ◻ Default plugins ◻ Featured plugins
Download featured plugins
We have a few plugins in our repository that most users like to download. Perhaps, you need them too?
Scala
Custom Languages
Plugin for Scala language support
Install
Live Edit Tool
Web Development
Provides live edit HTML/CSS/JavaScript
Install
IdeaVim
Editor
Emulates Vim editor
Recommended only if you are familiar with Vim.
Install and Enable
NodeJS
JavaScript
Node.js integration
Install
New plugins can also be downloaded in Settings | Plugins
Skip Remaining and Set Defaults
Back to Default plugins
Start using IntelliJ IDEA

Figure C-19. *Download Featured Plugins dialog*

The first time IntelliJ launches, you see the screen shown in Figure C-20, with a Create New Project option as well as configuration and help options. Click Create New Project to create a new, empty project so you can make sure IntelliJ IDEA is working.

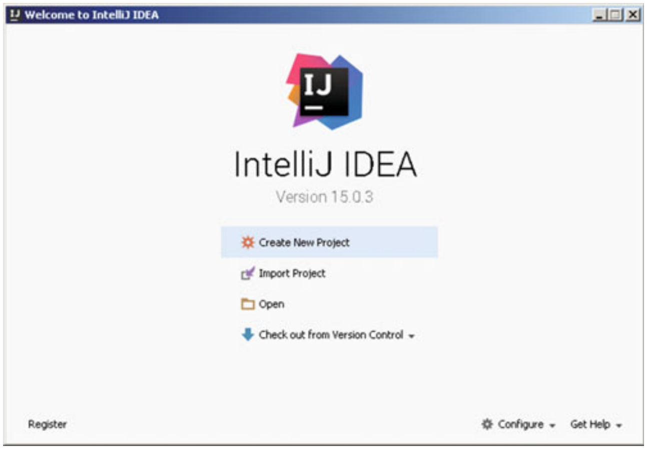


Figure C-20. *IntelliJ IDEA startup screen*

The default New Project dialog has the Java project type selected (left) with options (right) as shown in Figure C-21.

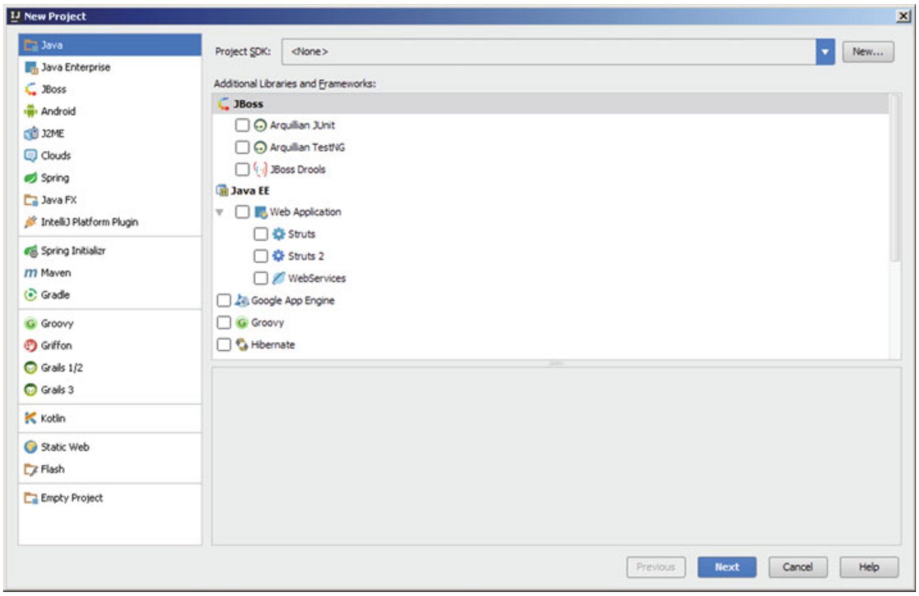
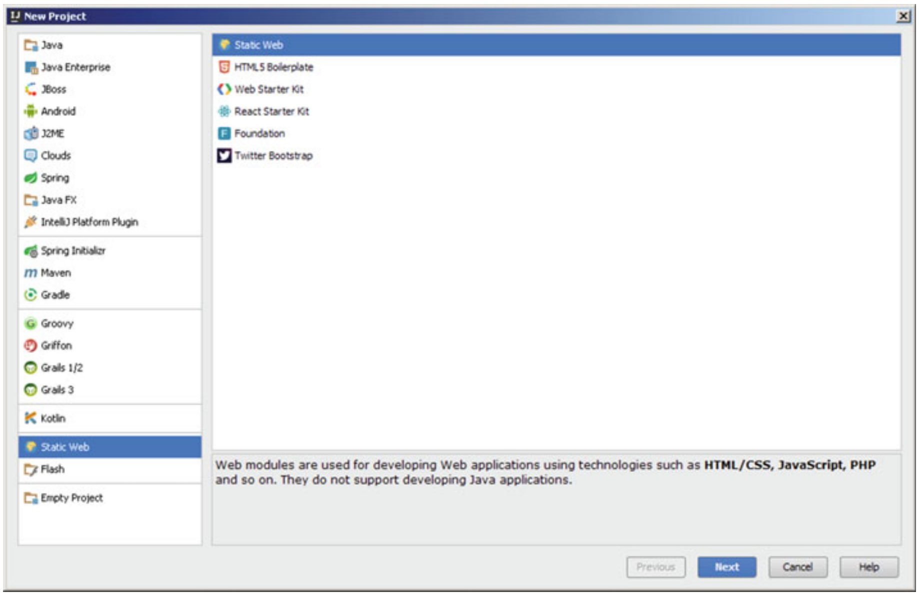


Figure C-21. *New Project dialog with the default Java project*

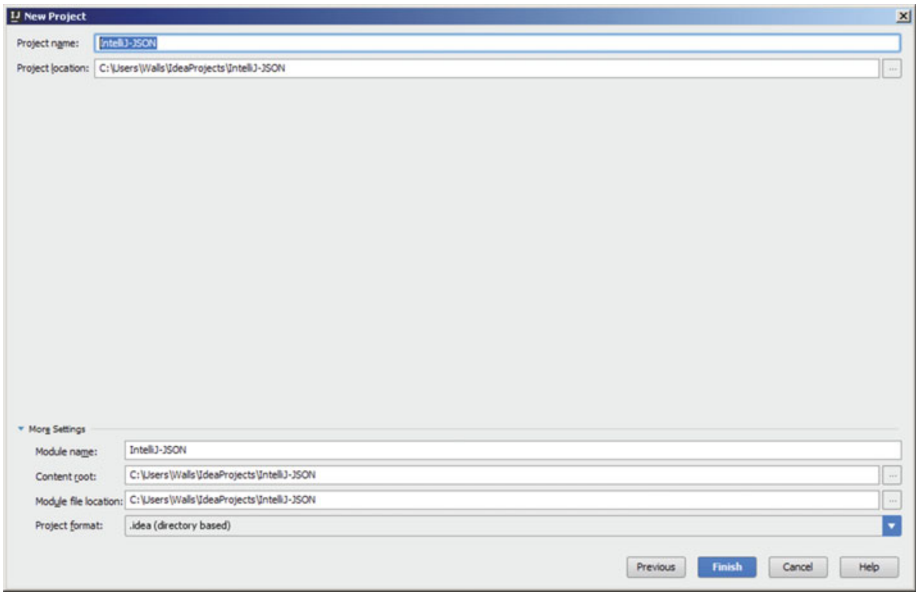
Select the Static Web project type shown in Figure C-22.



Web modules are used for developing Web applications using technologies such as **HTML/CSS, JavaScript, PHP** and so on. They do not support developing Java applications.

Figure C-22. *New Project dialog with Static Web (JSON) project selected*

Name your project **IntelliJ-JSON** using the Project Name dialog shown in Figure C-23, and click Finish.

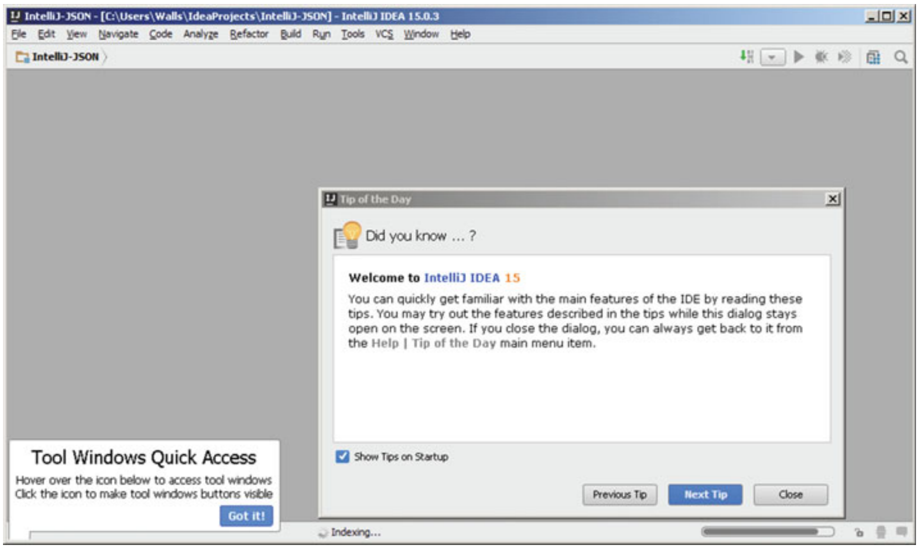


Project name: IntelliJ-JSON
Project location: C:\Users\Walls\IdeaProjects\IntelliJ-JSON

More Settings
Module name: IntelliJ-JSON
Content root: C:\Users\Walls\IdeaProjects\IntelliJ-JSON
Module file location: C:\Users\Walls\IdeaProjects\IntelliJ-JSON
Project format: .idea (directory based)

Figure C-23. *Name the project **IntelliJ-JSON**, and click Finish*

An empty IntelliJ IDEA loads with the Tip of the Day dialog showing and tooltip pop-ups enabled, as shown in Figure C-24. You can use these to explore IntelliJ IDEA features.


IntelliJ-JSON - [C:\Users\Walls\IdeaProjects\IntelliJ-JSON] - IntelliJ IDEA 15.0.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IntelliJ-JSON
Tip of the Day
Did you know ... ?
Show Tips on Startup
Previous Tip
Next Tip
Close
Tool Windows Quick Access
Got it!
Indexing...

Welcome to IntelliJ IDEA 15

You can quickly get familiar with the main features of the IDE by reading these tips. You may try out the features described in the tips while this dialog stays open on the screen. If you close the dialog, you can always get back to it from the Help | Tip of the Day main menu item.

Hover over the icon below to access tool windows
Click the icon to make tool windows buttons visible

Figure C-24. *The empty IDEA loads, showing tips and pop-ups*

You can also see the loading progress bar at lower right in Figure C-24. When this finishes, the IntelliJ IDEA shown in Figure C-25 appears with an empty project structure.

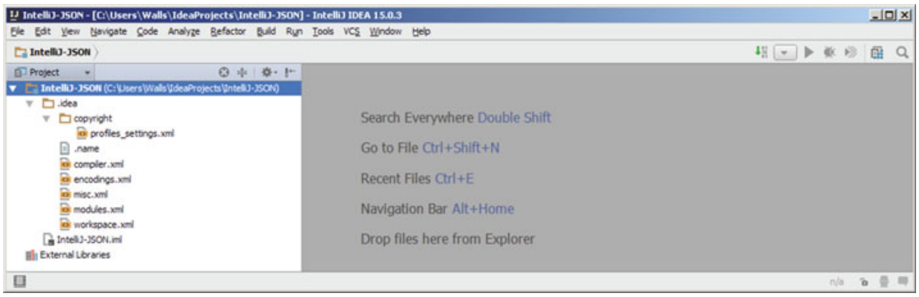

IntelliJ-JSON - [C:\Users\Walls\IdeaProjects\IntelliJ-JSON] - IntelliJ IDEA 15.0.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
IntelliJ-JSON
Project
IntelliJ-JSON (C:\Users\Walls\IdeaProjects\IntelliJ-JSON)
.idea
copyright
profiles_settings.xml
.name
compiler.xml
encodings.xml
misc.xml
modules.xml
workspace.xml
IntelliJ-JSON.iml
External Libraries
Search Everywhere Double Shift
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here from Explorer
n/a

Figure C-25. *IntelliJ has an empty project and is ready to use*

If you're going to be creating new media-compatible JSON applications, you need to get more open source packages so you can create new media assets referenced by JSON, JavaScript, Java, and so forth. These span all new media genres, including digital imaging, i3D, digital illustration, VFX, digital audio, digital painting, and digital video editing.

GIMP 2.8: Digital Image Editing and Compositing

The GIMP project offers a professional imaging software package that allows you to do digital image editing and compositing, much as you would using Adobe Photoshop or Corel PaintShop Professional. Download this software package at `www.gimp.org` and install it; it is quite professional. GIMP is currently at version 2.8.16, but version 3.0 is just around the corner and a preview (2.9.2) of V3 is available! The GIMP home page and download button are shown in Figure C-26.

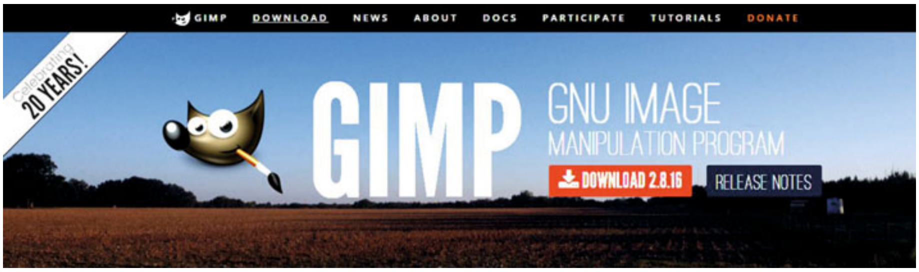


Figure C-26. *Go to* `www.gimp.org`*, and download GIMP 2.8.16*

If you want to learn digital image compositing, check out *Digital Image Compositing Fundamentals* (Apress, 2015).

Blender: 3D Modeling, Rendering, and Animation

The Blender Foundation project offers a professional i3D software package called Blender that allows you to do modeling of 3D objects as well as rendering and animation. Download this software package at `www.blender.org`, and install it. Blender's home page and blue download button are shown in Figure C-27.

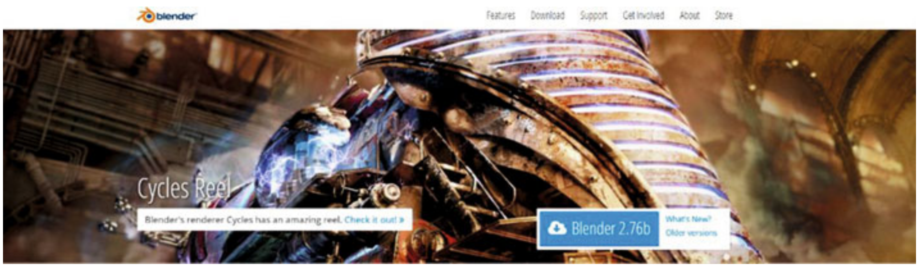
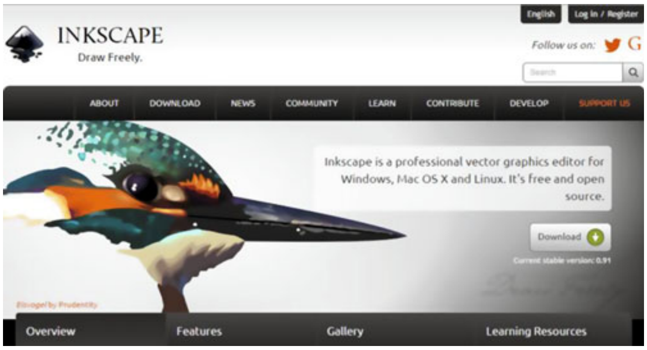


Figure C-27. *Go to* `blender.org`*, and download the latest version*

This is a professional-level software package with many of the same features as 3D Studio Max, Maya, XSI, and Lightwave.

Inkscape: Digital Illustration and Digital Painting

The Inkscape Project offers a professional digital illustration software package called Inkscape that can also do digital painting. Download this software package at `www.inkscape.org`. Inkscape's home page and download button are shown in Figure C-28.



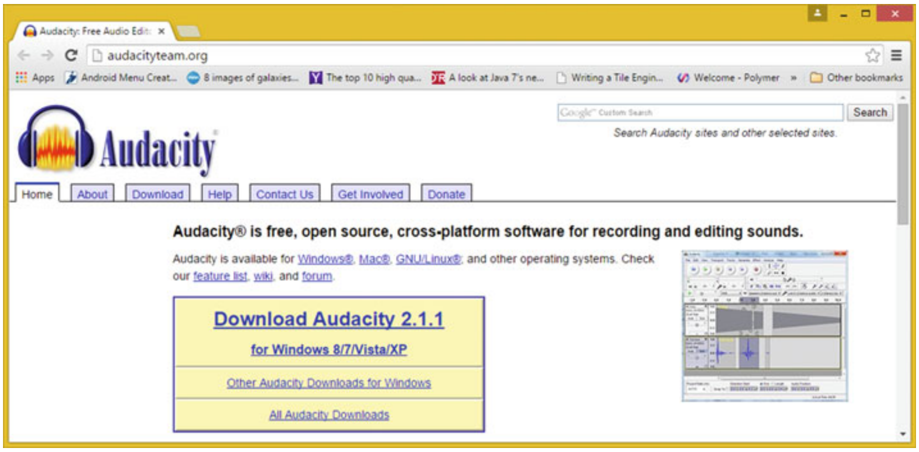
Inkscape is a professional vector graphics editor for Windows, Mac OS X and Linux. It's free and open source.

***Figure C-28.** Go to `inkscape.org`, and download the latest version*

If you want to learn digital illustration, check out *Digital Illustration Fundamentals* (Apress, 2015), as well as *Digital Painting Techniques* (Apress, 2015).

Audacity: Digital Audio Editing and Special Effects

The Audacity team offers a professional digital audio software package called Audacity that is for digital audio editing, sweetening, and special effects. You can download this software package at `www.audacityteam.org`. The Audacity home page and **Download** link are shown in Figure C-29.



Audacity® is free, open source, cross-platform software for recording and editing sounds.

Audacity is available for Windows®, Mac®, GNU/Linux®, and other operating systems. Check our feature list, wiki, and forum.

***Figure C-29.** Go to `audacityteam.org`, and download version 2.1.1*

Audacity offers many of the same digital audio editing features as many professional audio editors, and it is adding 64-bit capabilities and professional features every month. The next version will have a more professional user interface look and feel. If you want to learn more about digital audio editing, synthesis, and special effects, check out *Digital Audio Editing Fundamentals* (Apress, 2015). Digital audio can greatly enhance the user experience for any of the JSON applications you create in the future.

Visual Effects: BlackMagic Design Fusion 8.0 VFX

BlackMagic Design's Fusion 8 used to cost thousands of dollars; it offers a professional visual effects (VFX) software package used in film and television. Download this software package at `www.blackmagicdesign.com/products/fusion/`. Fusion 8's home page and blue Download button are shown in Figure C-30. If you want to learn more about VXF, check out *VFX Fundamentals* (Apress, 2016).

Figure C-30. *Go to* `blackmagicdesign.com`*, and download Fusion 8*

Digital Video Editing: Editshare Lightworks 12.6

EditShare Lightworks offers professional digital video editing in a software package that also does special effects. Download the software package at `www.lwks.com` after signing up for the download. Lightworks' home page and **Downloads** tab are shown in Figure C-31, where you can select your OS version. I recommend using 64-bit OS and software so that you can use 8MB of memory!

Figure C-31. *Go to* `lwks.com`*, and download Lightworks for your OS*

Office Productivity Suite: Apache OpenOffice 4.1.2

Apache OpenOffice, originally Sun Microsystems' StarOffice, was acquired by Oracle and released as open source. This will provide your JSON development business with professional office and business productivity software support. Download this great software package at `www.openoffice.com`. The Apache OpenOffice home page and Download button are shown in Figure C-32, where you can select your OS, language, and software version. I recommend using a 64-bit OS and software so that you can use 8MB of memory.

Figure C-32. *Download the Apache OpenOffice 4.1.2 full installation*

Summary

In this appendix, you set up your JSON workstation by downloading and installing the open source Java 8 JDK and the IntelliJ IDEA IDE software you should use to code JSON applications. I also showed you some professional new media software that is free for commercial use and that you should avail yourselves of, because it is exceptionally valuable.

Index

A, B

Apache OpenOffice, 97, 115–116, 138

C

Character values
- copyright attribute, 68
- definition, 68
- folder attribute, 68
- JavaScript (*see* String values)
- newline, 67
- reverse solidus, 66
- string literal, 67
- tab character, 67
- Unicode character set, 65

Classes, 32
COBOL, 3
Constants, 32

D

Data arrays, 17
Data structures, 16
- array object data, 53
- data collection, 51–52
- data definition structure, 54
- data format, 54
- object attribute data, 54
- one-dimensional array, 52
- two-dimensional array, 52
- variable declaration, 52

Data types
- Boolean value, 61–64
- null value, 62–64
- number value, 60–61
- string value, 59–60

Document Object Model (DOM), 55

E, F, G

Eclipse Mars
- Apache OpenOffice 4.1.2, 115
- Audacity, 113
- BlackMagic Design Fusion 8.0 VFX, 114
- Blender, 112
- core processor, 100
- digital video editing, 115
- Eclipse 4.5
  - accept now button, 109
  - Eclipse Installer, 107
  - Eclipse Mars launch, 110
  - Eclipse Mars Loader screen, 111
  - FirstApp sample app, 111
  - installation folder, 108
  - Installing progress bar, 109
  - Java EE developers, 108
  - NetBeans install, 107
  - update older version, 110
  - Workspace Launcher dialog, 110
- GIMP 2.8, 112
- hardware, 100
- Inkscape Project, 113
- Java 8
  - destination folder dialog, 106
  - downloads web site, 104
  - download eclipse.org, 102
  - download Java EE, 102
  - Java JDK, 103
  - programs and features, 105
  - setup dialog, 106
  - uninstall option, 105
- Mars IDE, 99
- memory access, 100
- Open source software, 101

ECMAScript, 15

H

Hard objects
    constructor function, 47–48
    dot property, 49
    literal notation, 48
Hyper-Schema, 22

I

Inheritance, 45
IntelliJ IDEA
    administrator option, 126
    Apache OpenOffice, 138
    Audacity, 136
    BlackMagic Design Fusion 8.0 VFX, 137
    Blender, 135
    complete installation dialog, 128
    EditShare Lightworks 12.6, 137
    empty IDEA, 134
    GIMP 2.8, 135
    hardware, 118
    Inkscape Project, 136
    installation options dialog, 127
    install complete, 127
    IntelliJ-JSON, 133
    Java 8
        destination folder dialog, 124
        installer dialogs, 124
        Java JDK, 121
        Java SE JDK, 122
        load jetbrains.com/idea/, 120
        programs and features, 123
        trial version, 120
        uninstall, 123
    Java project type, 132
    license activation, 129
    license agreement dialog, 129
    memory access, 119
    open source software, 119
    plugins dialog, 131
    run button, 126
    start IntelliJ IDEA, 128
    startup screen, 131
    Static Web project type, 132
    tasks dialog, 130
    theme dialog, 130
Interfaces, 32

J, K, L, M

Java development kit (JDK), 33
JavaScript Object Notation (JSON)
    advantages, 15
    description, 15
    eval() function, 18
    hard object, 17
    license, 20
    parse() function, 18–19
    soft objects, 17
    stringify() function, 19
    text-based format, 16
JSON Object Model
    DOM, 55
    Java classes and Java interfaces, 56
    JSONArrayBuilder, 56
    JSONGenerator interface, 57
    JSONParser interface, 57
    JSR 55, 353
    parser event, 55
    streaming approach, 55
    streaming model API, 57

N

NetBeans 8.1
    Apache OpenOffice, 97
    Audacity, 94
    BlackMagic Design Fusion 8.0 VFX, 95
    Blender, 93
    bootstrap code, 2
    client-side software, 3
    code refactoring, 2
    CSS3 style editing support, 4
    debugging, 5
    EditShare Lightworks 12.6, 96
    extensible architecture, 3
    GIMP 2.8, 93
    hardware, 84
    Inkscape Project, 94
    Installer dialog, 91
    Java 6/Java 7 code, 2
    Java 8
        Destination Folder dialog, 90
        HTML5/JavaScript IDE, 87
        installation, 86
        Java JDK, 87
        Java SE JDK, 88

Programs And Features, 89
uninstall, 89
JSON project creation
bower.json file, 10
Chrome Extension dialog, 12
Google Chrome Web Store page, 12
Gruntfile.js file, 8, 11
gulpfile.js file, 10
HTML5-JSON project, 7
HTML/JavaScript, 6
NetBeans Connector plug-in, 11–12
output, 9
package.json file, 8–9
Re-Run Project button, 12
site templates, 7
starting-point project, 6
startup screen, 5
Tools dialog, 8
Launch NetBeans, 92
matching code, 2
memory access, 84
Open source software, 85
program code profiler, 5
project-management tool, 3
responsive web design, 4
Summary dialog, 92
Node.JS Package Manager (NPM), 8
Numeric values
exponential notation, 78
array data, 80
data value, 79
integer value, 79
negative fractional value, 81–82
negative integer value, 80–81
positive fractional value, 80
json.org, 71
negative real number values, 76–78
negative whole number values, 73–74
positive real number values, 74–76
positive whole number values, 72–73
types, 71

O

Object-Oriented Programming languages (OOPs), 31
Java
Car object, 40
Car() constructor method, 43–44
class constructor method, 37
classes, 32
concepts and principles, 32
constants, 39
constructor method, 37
directories and folders, 40
encoding objects, 41
event handling functions, 36
features, 32
games/IoT application, 33
hard objects (*see* Hard objects)
import statement, 33
inheritance, 45
instance, 37
InvinciBagel class, 39
InvinciBagel game, 33
JavaFX EventHandler class, 33
JavaScript (HTML5), 32
JDK, 33
method declaration, 36
methods, 40
new keyword, 38, 44
object creation, 37
object instantiation, 38, 44
packages, 32
programming structure, 34
soft objects (*see* Soft objects)
variables, 39
virtual reality, 39
JSON.parse(), 32

P, Q, R

Python, 3

S, T, U

Schema
advantage, 22
components, 21
Hyper-Schema, 22
Schema Core (*see* Schema Core)
Schema Core
applicability, 24
array, 23
Boolean, 23
client-server interfaces, 24
correlation mechanisms, 28
customization, 25
empty schema, 23
HTTP specifications, 24

Schema Core (*cont.*)
  HTTP values, 25
  hypermedia and linking, 24
  initial resolution scope, 26
  instance, 24
  integer, 23
  keywords, 23, 25
  mathematical integers, 25
  MIME type, 24, 28
  null, 23
  number, 23
  object, 23
  programming language, 24
  RFC2119, 23
  RFC4627, 23
  root schema, 23
  security, 25, 28
  string, 23
  subschema, 23
  URI resolution-scope alteration, 26
  validation, 24
  value equality, 23
Soft objects
  attributes, 49
  JSON object, 49
String values
  char primitive, 69
  definition, 59
  java.lang.String class, 69
  new keyword, 70
  primitive data values, 69
  string literal/anonymous string, 60
  Unicode character
    support, 65–66
  var sampleStringValue, 70

■ V, W, X, Y, Z

Visual Effects (VFX) software package, 95,
    114, 137