CMIS and Apache Chemistry in Action

by Florian Müller
Jay Brown
Jeff Potts

Chapter 1

brief contents

1 Introducing CMIS

This chapter covers

- Presenting the CMIS standard
- Setting up your development environment
- Taking your first CMIS steps using Groovy and the CMIS Workbench
- Understanding possible limitations before using CMIS for your project

This chapter introduces the Content Management Interoperability Services (CMIS) standard. After running through a high-level overview of the standard and learning why it's important, you'll work on a simple hands-on example. By the end of the chapter, you'll have a reference server implementation running on your local machine and you'll know how to use Groovy to work with objects stored in a CMIS server by using a handy tool from Apache Chemistry called *CMIS Workbench*.

1.1 What is CMIS?

We're willing to bet that at some point in your career you've written more than a few applications that used a relational database for data persistence. And we'll further wager that if any of those were written after, say, 1992, you probably weren't too concerned with which relational database your application was using. Sure, you

might have a preference, and the company using your application might have a standard database, but unless you were doing something out of the ordinary, it didn't matter much.

This database agnosticism on the part of developers is only possible because of the standardization of SQL. Before that happened, applications were written for a specific relational back end. Switching databases meant porting the code, which, at best, was a costly exercise and, at worst, might be completely impractical. Before standardization, developers had to write applications for a specific database, as shown in figure 1.1.

This notion of writing applications that only work with a particular database seems odd to modern-day developers who are used to tools like ODBC and JDBC that can abstract away the details of a particular database implementation. But that's the way it was. And that's the way it still is for many developers working in the world of content management.

Until recently, developers writing applications that needed to use Enterprise Content Management (ECM) systems for data persistence faced the same challenge as those pre-SQL-standardization folks: Each ECM system had its own API. A software vendor with expertise in accounts payable systems, for example, and a team of .NET developers were locked into a Microsoft-based repository. If a customer came along who loved the vendor's solution but didn't want to run Microsoft, they had a tough choice to make.

That's where CMIS comes in.

CMIS is a vendor-neutral, language-independent specification for working with ECM systems (sometimes called *rich content repositories* or more loosely, *unstructured repositories*). If you're new to the term *repository* (or *repo*, for short), think of it as a place where data—mostly files, in this case—lives, like a file cabinet.

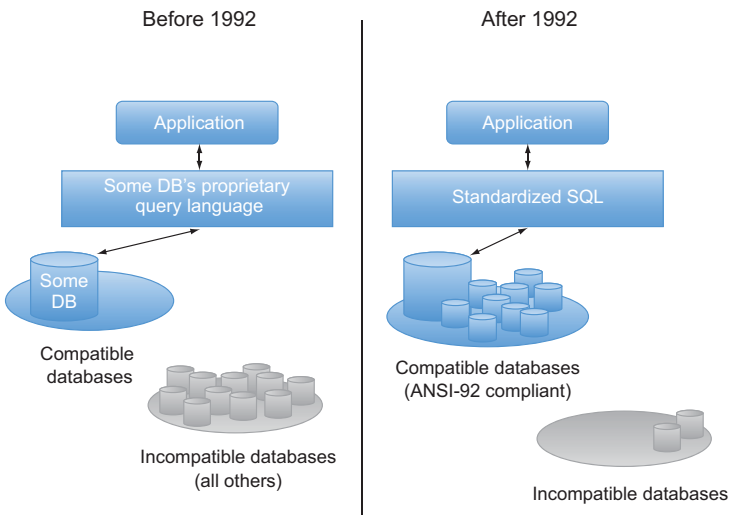

Figure 1.1 Before SQL standardization, developers wrote applications against specific databases.

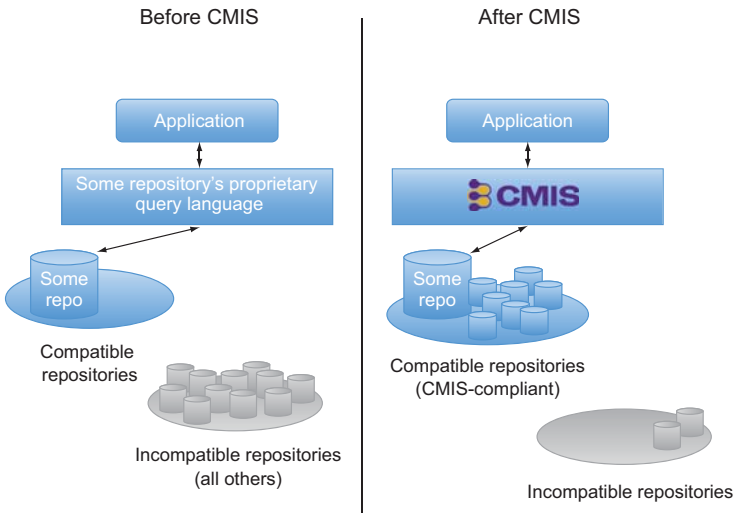

Figure 1.2 CMIS standardizes the way applications work with rich content repositories in much the same way SQL did for relational databases.

With CMIS, developers can create solutions that will work with multiple repositories, as shown in figure 1.2. And customers can have less vendor lock-in and lower switching costs.

The creation of the CMIS specification and its broad adoption is almost as significant and game-changing to the content management industry as SQL standardization and the adoption of that standard was to the relational database world. When enterprises choose repositories that are CMIS-compliant, they reap the following benefits.

Content-centric applications, either custom built or bought off the shelf, are more independent of the underlying repository because they can access repositories in a standard way instead of through proprietary APIs. This reduces development costs and lowers switching costs.

Developers can ramp up quickly because they don't have to learn a new API every time they encounter a new type of repository. Once developers learn CMIS, they know how to perform most of the fundamental operations they'll need for a significant number of industry-leading, CMIS-compliant repositories.

Because CMIS is language-neutral, developers aren't stuck with a particular platform, language, or framework driven by the repository they happen to be using. Instead, developers have the freedom to choose what makes the most sense for their particular set of constraints.

Enterprise applications can be more easily and cheaply integrated with content repositories. Rather than developing expensive, one-off integrations, many enterprise applications have *CMIS connectors* that allow them to store files in any CMIS-compliant repository.

OK, you're convinced. CMIS is kind of a big deal in the Enterprise Content Management world. Let's talk a little bit about how the CMIS specification is defined, look at an example of what you could use CMIS to do, and see a list of places where CMIS exists in the wild.

1.1.1 About the specification

CMIS is a *standard*, and the explanation of the standard is called a *specification*. The CMIS specification describes the data model, services, and bindings (how a specific wire protocol is hooked up to the services) that all CMIS-compliant servers must support. You'll become intimately familiar with the data model, services, and bindings as you work through the rest of this book.

The CMIS specification is maintained using a collaborative, open process managed by the Organization for the Advancement of Structured Information Standards (OASIS). According to its website (www.oasis-open.org), "OASIS is a non-profit consortium that drives the development, convergence, and adoption of open standards for the global information society." Using an organization like OASIS to manage the CMIS specification ensures that anyone who's interested can get involved in the specification, either as an observer or as an active voting member.

The group of people who work on the specification is called the *Technical Committee* or *TC*, for short. What's great is that the CMIS TC isn't made up of only one or two companies or individuals but is composed of more than 100 people from a wide range of backgrounds and industries, including representation from the who's who of content management vendors, large and small.

1.1.2 What does CMIS do?

OK, so CMIS is an open standard for working with content repositories. But what does it do? Well, the standard doesn't do anything. To make it interesting, you need an implementation. More specifically, you need a CMIS-compliant server. When a content repository is CMIS-compliant, that means that it provides a set of standard services for working with the objects in that repository. You'll explore each of those services in the coming chapters, but the set includes things like creating documents and folders, searching for objects using queries, navigating a repository, setting permissions, and creating new versions of documents.

Let's discuss a real-world example. Suppose you work for a company whose content lives in three different repositories: SharePoint, FileNet, and Alfresco. The sales team comes to you and asks for a system that will build PowerPoint presentations on the fly by pulling data from each of these repositories. The PowerPoint presentations need to be based on a template that resides in SharePoint and will include, among other things, images of the last three invoices. The invoice images reside in FileNet. The final PowerPoint file is stored in Alfresco and accessed by the sales team using their tablets. A high-level overview of this application is shown in figure 1.3.

Before CMIS, your system would have to use at least three different APIs to make this happen. With CMIS, your system can use a single API to talk to each of the three repositories, including the mobile application.

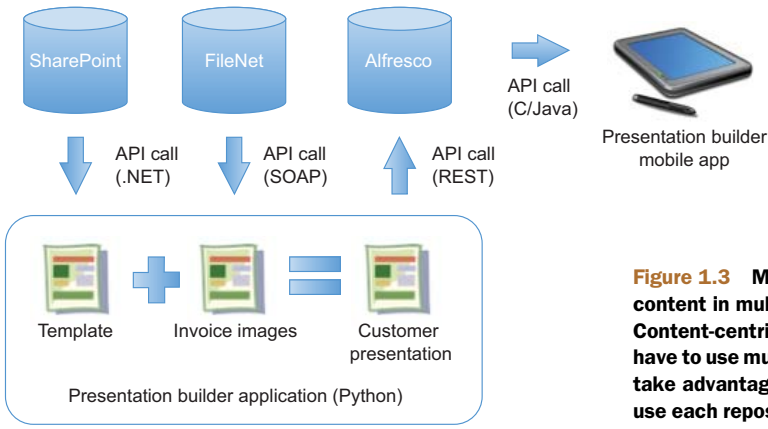

Figure 1.3 Most companies store content in multiple ECM repositories. Content-centric applications either have to use multiple disparate APIs, or take advantage of CMIS's ability to use each repository in a standard way.

Three different ECM systems in the same organization?

You may be wondering how real-world this example is—three ECM systems in the same organization? In fact, it happens quite often. According to AIIM, the Association for Information and Image Management, which is a major ECM industry organization, "72% of larger organizations have three or more ECM, Document Management, or Records Management systems" and "25% have five or more" ("State of the ECM Industry," AIIM, 2011).

How does a company find itself in this situation? It happens for many reasons. Sometimes these systems start out as departmental solutions. In large organizations where there may not be an enterprise-wide ECM strategy, multiple departments may—knowingly or unknowingly—implement different systems because they feel their requirements are unique, they have timelines that don't allow for coordination with other departments, or any number of other reasons.

Similarly, companies often bring in multiple systems because they may fill niche requirements (like digital asset management or records management) and one vendor may be perceived as offering a better fit for those highly specific requirements. But ECM vendors, particularly large ones, often use their niche solution as a foot in the door—it's a common strategy for ECM vendors with "suites" of products to subsequently expand their footprint from their original niche solution to other product offerings.

As each department or niche implementation sees success, the rollouts broaden until what once were small, self-contained solutions may grow to house critical content for entire divisions. Once each ECM system has gotten so big, the business owners are reluctant to consolidate because the risk may not justify the benefit. After all, the business owners are happy—their requirements are being met.

As a result, it's common to walk into a company with many different ECM systems. If this is a problem you deal with, we hope the techniques you learn in this book will save you time, money, and frustration.

1.1.3 Where is CMIS being adopted?

Standards that no one implements aren't useful. So far, CMIS has avoided this fate. Thanks to the early involvement of a number of large ECM vendors in developing the specification, and the specification's language neutrality, CMIS enjoys broad adoption. If you're currently using an ECM repository that's updated to a fairly recent version, it's likely to be CMIS-compliant. Table 1.1 shows a list of common ECM vendors or open source projects and when they started to support CMIS. This list is only a subset of the CMIS-compliant servers available at the time of this writing. The CMIS page on Wikipedia (http://en.wikipedia.org/wiki/Content_Management_Interoperability_Services) contains a more exhaustive list. If you don't see your favorite content server in the list, ask your vendor.

Table 1.1 Selection of ECM vendors, or open source projects, and their support for CMIS

Vendor	Product	Release that first provided CMIS 1.0 support
Alfresco Software	Alfresco	3.3
Alfresco Software	Alfresco Cloud	March 2012
Apache Chemistry	InMemory Repository	0.1
Apache Chemistry	FileShare Repository	0.1
EMC	Documentum	6.7
HP Autonomy Interwoven	Worksite	8.5
IBM	FileNet Content Manager	5.0
IBM	Content Manager	8.4.3
IBM	Content Manager On Demand	9.0
KnowledgeTree	KnowledgeTree	3.7
Magnolia	CMS	4.5
Microsoft	SharePoint Server	2010
Nuxeo	Platform	5.5
OpenText	OpenText ECM	ECM Suite 2010
SAP	SAP NetWeaver Cloud Document Service	July 2012

As the previous table illustrates, a variety of CMIS-compliant servers are available. CMIS gives you a single API that will work across all of these servers.

1.2 Setting up a CMIS test environment

Alright, time to roll up your sleeves and set up a working CMIS development environment that you can take advantage of as you work through the rest of this book.

We'll give you a proper introduction to Apache Chemistry in part 2 of the book. For now, it's important to know that Apache Chemistry is a project at the Apache Software Foundation that groups together a number of CMIS-related subprojects, including client libraries, server frameworks, and development tools. It's the de facto standard reference implementation of the CMIS specification. One of the Apache Chemistry subprojects is called *OpenCMIS*, and it's made up of multiple components. For the rest of this chapter, you'll use two of those components: the OpenCMIS InMemory Repository and the CMIS Workbench.

The OpenCMIS InMemory Repository, as the name suggests, is a CMIS-compliant repository that runs entirely in memory. It's limited in what it can do, but it'll serve our needs quite nicely.

The CMIS Workbench is a Java Swing application that we'll use as a CMIS client to work with objects in the CMIS server. The CMIS Workbench was created using the OpenCMIS API and is typically used by developers who want a view into a CMIS repository that is based purely on the CMIS specification. For example, suppose you're working with Microsoft SharePoint, which has a variety of ways to create, query, update, and delete content that resides within it, and you want to integrate your application with SharePoint using CMIS. You could use the CMIS Workbench to test some queries or inspect the data model. If you want to know if you can do something purely through CMIS, one test is to try to do it through the CMIS Workbench. If the CMIS Workbench can do it, you know you'll be able to do it as part of your integration.

One of the key features of the CMIS Workbench, from both a "developer utility" perspective and a "let's learn about CMIS" perspective, is its interactive Groovy console. The Groovy console is perfect for taking your first steps with CMIS.

When you're finished setting up your environment, it'll look like figure 1.4.

We've made it easy to set up your local CMIS development environment. Everything you need is in the zip file that accompanies this book (see appendix E for links to resources). Let's unzip the components you'll need for the rest of part 1.

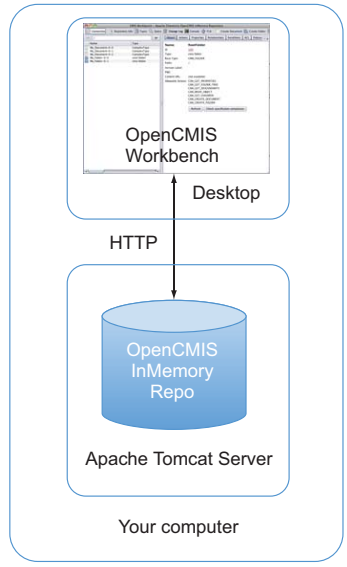

Figure 1.4 Your local CMIS development setup includes two components: the CMIS Workbench and the OpenCMIS InMemory Repository. This is all you'll need for the examples in part 1 of this book.

Downloading and building your own CMIS tools

To save you time and make the setup easier, we've taken distributions from the Apache Chemistry project and packaged them together with some sample configuration and data that will be used throughout the book. When you're ready to learn how to download out-of-the-box versions of these components, or you want to know how to build them from source, or you want to get the latest and greatest release of OpenCMIS, refer to appendix A.

1.2.1 Requirements

For the rest of part 1, all you need is the CMIS Workbench and the OpenCMIS InMemory Repository. These components both need a JDK (version 1.6 or higher will do). Other than that, everything you need is in the zip.

Before continuing, find a place to unzip the archive that accompanies this book. We'll call it $BOOK_HOME. Within $BOOK_HOME, create two directories: server and workbench.

1.2.2 Installing the OpenCMIS InMemory Repository web application

Let's install and start up the OpenCMIS InMemory Repository:

1. Change into the $BOOK_HOME/server directory and unzip inmemory-cmis-server-pack.zip into the directory.
2. Run ./run.sh or run.bat, depending on your platform of choice.

This will start up InMemory Repository on your machine, and it will listen for connections on port 8081. If you're already running something on port 8081, edit run.sh (or run.bat) and change the port number. All of the directions in the book will assume the InMemory repository is running on port 8081.

After the server starts up, you should be able to point your browser to http://localhost:8081/inmemory and see something that looks like figure 1.5.

Now you have a working CMIS server running on your machine. The CMIS server has some test data in it, but in order to work with it, you need a CMIS client. In part 1, you'll use a CMIS client that's already been built. It's a Java Swing desktop application called CMIS Workbench. Setting it up is the subject of the next section.

Figure 1.5 **Apache Chemistry OpenCMIS InMemory Repository welcome page**

1.2.3 Installing the CMIS Workbench

The CMIS Workbench is distributed as a standalone Java Swing application. Everything you need to run it is in the package included with the book. To install it, follow these steps:

1. Open a new window and switch to the $BOOK_HOME/workbench directory.
2. Unzip cmis-workbench.zip into the directory.
3. Run the appropriate batch file for your operating system. For example, on Windows, run `workbench.bat`. On Mac and Unix/Linux systems, run `workbench.sh`.

The Workbench will start up, and you should see an empty login dialog box, like the one in shown in figure 1.6.

Congratulations! You now have everything you need to explore a working CMIS implementation.

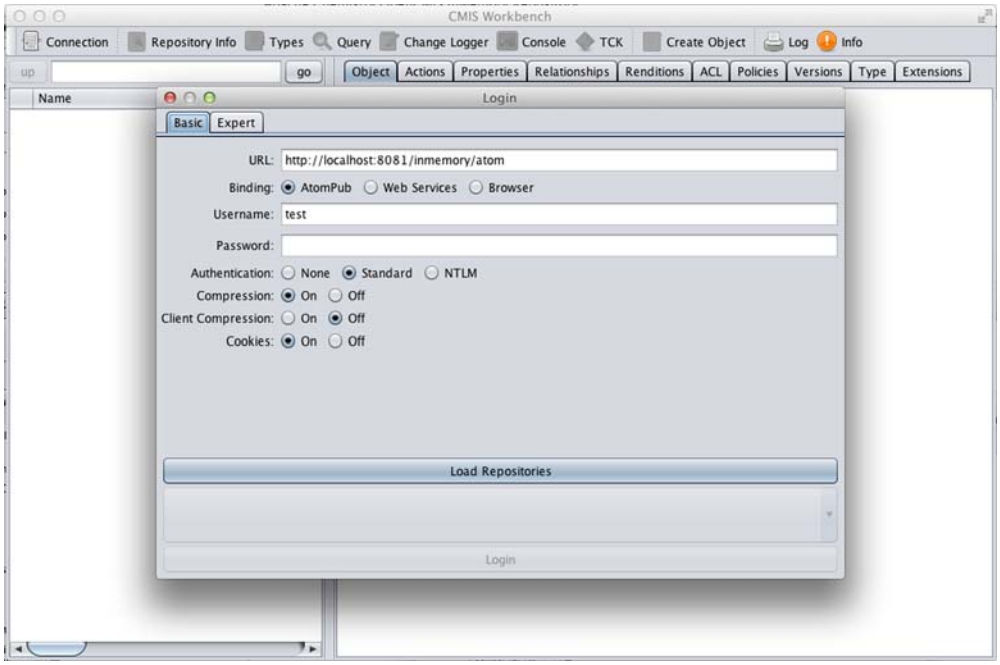

Figure 1.6 An empty CMIS Workbench login dialog box

1.3 Writing your first CMIS code using Groovy

Your OpenCMIS InMemory Repository is running, and so is the first CMIS client you'll be working with, the CMIS Workbench. It's time to get the two to work together.

1.3.1 Connecting to the repository

To talk to the OpenCMIS InMemory Repository, you need to choose a *binding* and you need to know the server's *service URL*, which depends on the binding you choose, as you can see in figure 1.7.

The binding is the method the CMIS client will use to talk to the server. You can also think of it as the protocol it'll use to communicate. In CMIS version 1.0, the two choices for binding are Atom Publishing Protocol (AtomPub) and Web Services. CMIS version 1.1 adds a third binding called the Browser binding. We'll go through the binding details in chapter 11. For now, we'll use the AtomPub binding.

The service URL is the entry point into the server. The CMIS client will learn all it needs to know about the server it's talking to by invoking the service URL and inspecting the response it gets back. The service URL depends on the server you're using, the binding you've chosen, and how the server is deployed. In this case, the server is deployed to a web application under the inmemory context, so the URL will begin with http://localhost:8081/inmemory; and the AtomPub service URL is /atom, so the full service URL is http://localhost:8081/inmemory/atom.

Figure 1.7 To connect to the repository, you must select a binding and specify the service URL.

THE CMIS WORKBENCH CAN CONNECT TO ANY CMIS SERVER We're using the Apache Chemistry InMemory Repository throughout this book because it's freely available, easy to install, and compliant with the CMIS specification. But, as the name implies, it stores all of its data in memory. That would never work for most production scenarios. Real ECM repositories persist their data to a more durable and scalable back end. Typically this is some combination of a relational database and a filesystem. If you have access to an ECM repository like Alfresco, FileNet, SharePoint, or the like, you can use the CMIS Workbench to work with data stored in those repositories. All you need to know is your repository's service URL.

1.3.2 *Try it—browse the repository using the CMIS Workbench*

You now know enough to be able to connect to the server. Follow these steps to use the CMIS Workbench to connect to the server and browse the repository:

1. If the CMIS Workbench isn't running, run it as previously discussed.
2. If the CMIS Workbench isn't displaying the login dialog box, click Connection in the upper-left corner.
3. Specify http://localhost:8081/inmemory/atom as the URL.
4. Take all the other defaults. Click Load Repositories.
5. The InMemory Repository only has one repository. You should see it in the Repositories list. Click Login.

If everything is working correctly, you should see the login dialog box close and the Workbench will display the contents of the repository, as shown in figure 1.8.

Take a few minutes to explore the Workbench. You can't hurt anything. Every time you restart the InMemory Repository, it'll revert to its original state.

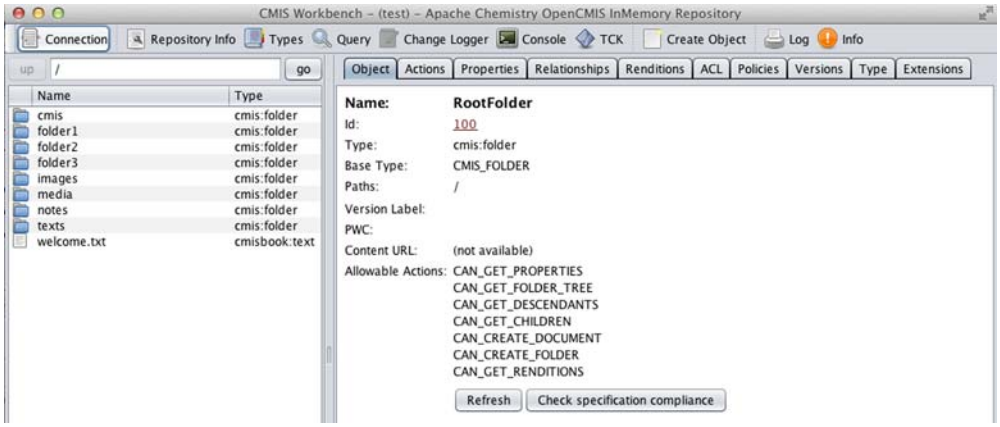

Figure 1.8 **Root folder of the OpenCMIS InMemory Repository**

Here are a few things to notice as you explore:

- As you click objects in the left-hand pane, the right-hand pane updates to provide details on what's selected.
- The right-hand pane has tabs across the top that group different sets of information about the selected object as well as actions you can take on the selected object.
- The items in the menu bar let you do things like change the connection details, inspect repository information, view the types defined on the server, and open a Groovy console. That's where we're headed next.

1.3.3 Try it—run CMIS code in the CMIS Workbench Groovy console

Groovy is a dynamic language that's easy for Java programmers to learn. It can run anywhere Java can run. It's different from Java in a few respects, such as the fact that semicolons are optional in most cases, closures are supported, and regular expressions are natively supported.

> **DON'T KNOW GROOVY? NO PROBLEM!** Don't worry if you don't know Groovy. We picked it for the examples in part 1 of this book because it's easy to learn, it looks similar to Java, it doesn't require a compiler, and the CMIS Workbench features a Groovy console. You'll probably easily grok what's going on as you work through the examples. But if you want to dive into Groovy, you can learn more from the Groovy home page (http://groovy.codehaus.org/) or from *Groovy in Action, Second Edition* (Manning, 2013).

The best way to get a feel for Groovy is to jump right in, so let's do that. Follow these steps to write a Groovy script that will display the repository's name:

1. From the CMIS Workbench, click Console, and select Main Template in the submenu.
2. A Groovy console window will be displayed with eight or nine lines of prepopulated code. Delete those lines.
3. Add the following two lines of Groovy:

```
def info = session.getRepositoryInfo()
println "Repository Name: " + info.getName()
```

4. Click the Execute Groovy Script button, which is the little document with the green arrow.

Your code should run without a hitch. The output of the program will be displayed in the bottom half of the Groovy console. It should look something like figure 1.9.

Let's look at a few important things:

- You didn't have to import anything.
- You didn't have to retrieve a session. It was handed to you in a variable called `session` that was already defined. The `session` variable represents a connection to the CMIS repository for the user you provided when you launched the

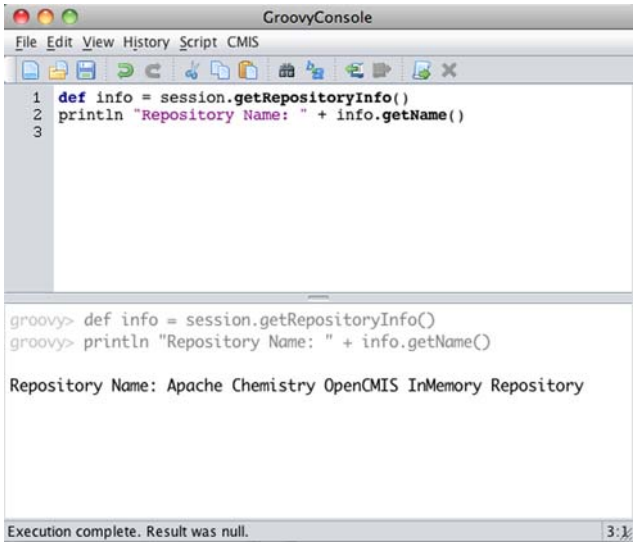

Figure 1.9 Groovy console after running code to retrieve the CMIS server name

Workbench. The object is an instance of `org.apache.chemistry.opencmis.client.runtime.SessionImpl`.

- You could have omitted the "get" and the parenthesis from the no-argument getters. For example, you could have said `session.repositoryInfo` and `info.name`.
- Any time you feel you need some help with the API, you can click CMIS > OpenCMIS Client API Javadoc, and the documentation will open in a browser window.
- When you first click Console in the CMIS Workbench, you'll see a list of Groovy script templates. You the choose - Main Template - and then replace it with your own code. When you have a chance, you might want to take a look at some of the other sample Groovy scripts that are provided.

And that's it. You've written your first CMIS code. We sense some disappointment, though. "I don't feel like I've experienced the true power of CMIS yet," you say. OK, overachiever. Earlier you learned that one of the beauties of CMIS is that, as a developer, once you learn CMIS you should be able to write code that works with any CMIS-compliant repository. You've demonstrated your ability to use the OpenCMIS InMemory Repository. How about an enterprise-grade repository from a completely different vendor?

It so happens that publicly available CMIS servers are waiting for folks like you who are testing client libraries or exploring CMIS. One of them is run by a company called Alfresco Software; its AtomPub service URL is http://cmis.alfresco.com/cmisatom. Unlike the InMemory Repository, you'll need credentials to authenticate with Alfresco. You can use the administrator's account, which is *admin*, and the password is also *admin*. Fair warning: the response time will be significantly slower than what you see with the local InMemory Repository.

SAVE YOUR SCRIPT To save some typing, do a File > Save on your current Groovy script before clicking Connect to specify the Alfresco service URL and credentials. Then, when you open the Groovy console, you can do a File > Open to reopen your script.

Now you know how to install a reference CMIS server and a handy CMIS client. You've had a glimpse of the power of CMIS as you used the same client to talk to two different implementations.

1.4 CMIS considerations

In the next chapter, you'll start to dive into the CMIS specification a little more deeply. But before doing that, let's discuss a few of the limitations of CMIS and how it compares to other content management standards. This will help you decide if CMIS might be right for your next project.

1.4.1 Understanding the limitations of CMIS

Like any industry-wide standard, CMIS has some limitations that may affect your ability to use it for a particular project. Whether or not these limitations affect you depends on your specific requirements.

LIMITED IN SCOPE

Enterprise Content Management systems vary broadly in their capabilities and functionality. Some of the differences are significant, such as whether or not the system has an embedded workflow engine, and others are minor, like whether or not the system supports access control lists (ACLs). The CMIS specification is flexible enough to accommodate differences between implementations: A repository doesn't have to support ACLs and can still be CMIS-compliant, for example. Or one repository might support "unfiled" documents, but another might require that documents always live in a folder.

In cases where the differences between repositories are too significant to be covered by one standard definition of a repository, CMIS omits those areas from its scope. Workflow is one example—you won't see anything about workflow in this book, even though workflow is a relatively common feature of ECM systems.

As a developer, you may be able to meet all of the requirements of your application by staying strictly with pure CMIS API calls. But there may be times when you'll have to supplement what CMIS provides with calls to your ECM system's proprietary APIs.

OBJECT MODEL IS BASED ON DOCUMENTS AND FOLDERS

In the next chapter, you'll see that two prominent domain objects covered by the specification are `cmis:document` and `cmis:folder`. That's because the CMIS specification assumes a general document management use case: you're using CMIS to manage documents (files) organized in a hierarchy of folders.

No user or group management

A CMIS repository typically uses named user accounts to control who can authenticate with the repository. But the CMIS specification provides nothing that helps you create user accounts or organize users into groups.

Does this mean your application can't assign ACLs to documents and folders? No. It means that if your application needs to create new users or modify groups of users, CMIS isn't going to help you to do that in a standard way. You'll have to use your repository's API or an LDAP directory to manage users and groups, if that's something your repository supports.

No support for defining content types until CMIS 1.1

You'll learn about content types in chapter 4. For now, realize that content in a CMIS repository belongs to a particular type, like document, folder, image, invoice, or web page. It's quite common for companies to define their own business-specific content types by updating the repository's data dictionary.

The first version of the CMIS specification doesn't provide for creating or updating content types, even if the underlying repository supports this feature natively. This may be a challenge if your application assumes that the types it needs are already configured in the repository's data dictionary. If they don't already exist, you'll have to provide documentation or configuration scripts when you deliver your CMIS application so that the system administrators can update the data dictionary with types to support your application.

Luckily, this is addressed in CMIS 1.1. With CMIS 1.1, your CMIS application can check to see if the required types have been configured, and if not, it can go ahead and create them using code, to avoid the need for manual changes to the data dictionary.

1.4.2 Comparing CMIS to the Java Content Repository (JCR) API

If you've worked with content management repositories for a while, you may already be familiar with the Java Content Repository (JCR) API, which is sometimes referred to as Java Specification Request (JSR) 170. What's the difference between CMIS and JCR? Table 1.2 breaks it down.

Table 1.2 Comparing CMIS and JCR

	JCR	CMIS
Standards body	Java Community Process	OASIS
Date first ratified	June 2005	April 2010
Vendor adoption	Limited. Several vendors provide JCR support in their repositories, but Adobe is the primary driver of the specification.	Many big-name ECM vendors actively participate in the specification and reference implementation, including EMC, IBM, Alfresco, SAP, HP Autonomy Interwoven, Oracle, Microsoft, and several others.

Table 1.2 Comparing CMIS and JCR *(continued)*

	JCR	CMIS
Primary language	Java, although work is being done to expand support to PHP.	Language-neutral. Any language that can speak HTTP can work with CMIS.
Reference implementation	Apache Jackrabbit	Apache Chemistry

It's important to note that CMIS and JCR aren't completely mutually exclusive. A given ECM repository might be compliant with both standards, which would mean developers would be free to choose which standard to use when working with that repository. Work has also been completed recently to bridge the two standards. You could, for example, write CMIS-compliant code that talks to a JCR repository.

1.5 Summary

You should now have a good idea of why the CMIS specification is so important to the ECM industry. After seeing some real-world examples of how you can apply CMIS to make your life easier as a content-centric application developer, you've probably already started thinking about some of the advantages of working with CMIS to build your applications:

- Content-centric applications can be more independent of the underlying content repository because they can access repositories in a standard way instead of through proprietary APIs.
- Developers can ramp up quickly because CMIS reduces the need to learn a proprietary API for every repository that's involved in an application.
- Developers have the freedom to choose what platform, language, or framework is the best fit for their particular constraints, without worrying whether or not it's supported by the repository they're working with, because CMIS is language-neutral.
- Expensive one-off integrations don't have to be built—applications can take advantage of standards-based connectors to CMIS-compliant repositories.

Beyond learning the *why* of CMIS, you rolled up your sleeves and put CMIS to *work*. You now have a working CMIS development environment based on freely available components from the Apache Chemistry project. You'll use this setup for the rest of the examples in part 1.

Now that you have a working development environment, it's time to start learning how to navigate a CMIS repository and what kind of objects you'll find in a CMIS repository once you connect to it. We'll start with two of the fundamental building blocks—folders and documents. On to chapter 2.