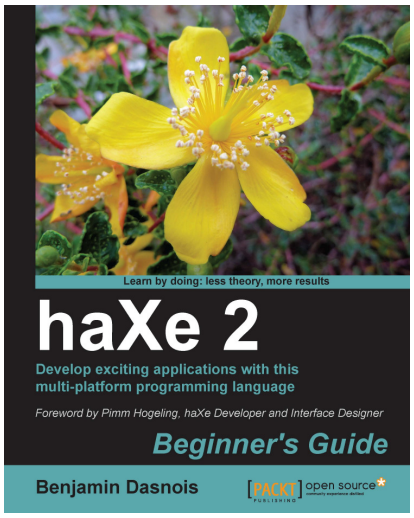
[PACKT] open source
community experience distilled
PUBLISHING

haXe 2 Beginner's Guide

Benjamin Dasnois



Chapter No. 3
"Being Cross-platform with haXe"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.3 "Being Cross-platform with haXe"

A synopsis of the book's content

Information on where to buy this book

About the Author

Benjamin Dasnois has always been fascinated by open source software and as such, has been giving courses about Linux in an IT school in France. In the meantime, Benjamin has been following and working with haXe since its beginning and uses it in his professional life. He now works on an open source project, started the integration of haXe into it, and made it work with the technologies that were already in use.

I would like to thank the entire haXe community, my schoolmates, and former students who really made me discover the contribution I could make by imparting my knowledge or communicating my skills.

haXe 2 Beginner's Guide

haXe is the universal programming language which is completely cross-platform and provides a standard library that remains the same—regardless of platform.

haXe 2 Beginner's Guide will get you up and running with this exciting language and will guide you through its features in the easiest way possible.

haXe has filled the gap in creating multi-platform applications, and this book will fill the gap in learning all you need to know about haXe—even if it is the first time you have heard of it.

This book will enable you to fully realize haXe's potential for translating code from a haXe program into different languages.

Start with learning how to install haXe, work your way up to templating, and finally learn how to make the same code work for multiple platforms. In between, find heaps of tricks and techniques and work with haXe's typing system. Learn about inheritance, go from learning what a parameter is to creating your own parameterized classes, and find out what the fuss is all about regarding the dynamic type.

By the time you are done with this book, you will find yourself writing efficient haXe code for multiple platforms in less time than you can say "compatible".

Here is haXe

haXe started as a web-oriented programming language and has gone a long way since its debut in late 2005. Its main goal was to correct several big issues within web development, but now people and companies are using it to be able to cover several platforms and devices.

Where and for what is haXe used?

Well, it seems like you are interested in haXe, and we will soon write our first haXe program together. However, before doing that, you may be wondering who uses haXe, and for what kind of tasks or applications it is used.

Where is haXe used?

haXe is actually used in several places. There are some companies that are using it for their professional work. That does not necessarily mean that they use haXe for their whole work; many companies that uses haXe are indeed going through a move towards haXe, or are simply using haXe for some parts of their work and do not necessarily plan to use it more. Even though haXe is not difficult to learn, it is really difficult to change the technologies a company is using.

There are also several open source projects that use haXe. For example, haXe libraries are written in haXe, and many of these are open source.

There are also hobbyists, or students, who are learning and using haXe. Learning haXe and participating in the community helps in learning many programming concepts (the community sometime talks about some people that are not in haXe at the moment but may be added later, for example.) It is also advantageous to learn something other than what is usually taught in schools.

What is haXe used for?

haXe is used for a lot of different things. At first, it is mainly used for making website-related things. This can be web-based tools (such as a mind-mapping program), or games in Flash or JavaScript.

It can also be used for writing desktop applications. Some people have been using a great combination of Flash and Neko to achieve this. However, this is not the only way to do things! You can write command-line applications that will not need Flash, or you can use or write another graphic library.

One can write servers with haXe too. The framework provides some helpers to do that. One impressive example of a server written in haXe is haXeVideo available at `http://code.google.com/p/haxevideo/`. This is a video server that allows live streaming of videos to Flash clients. Although it is not ready for production at the moment, it is a great start and can put you on the right track.

haXe can be used to directly target mobile devices such as iOS and Android, but it's also possible to use it to write web-based applications targeting mobile devices. Nowadays, many phones have a browser that is able to run JavaScript code. There are also some runtimes that allow one to create applications based on web technologies running on mobile devices.

One language to rule them all

If you have already been developing web applications, then you certainly know how painful it is to switch from, say, PHP, to JavaScript and Flash. In fact, even the most basic things can become painful in such a scheme. For example, switching from one language to another can make adding objects to an array or a list difficult and error-prone while developing web applications.

haXe is there to unify the development of the three main parts of web applications; with it, you will be able to write your server-code, your client-code, and your rich-client code with only one language: haXe. However, it is also possible to use it with other languages.

Object-oriented programming

haXe also brings the oriented-object programming concept with the well-known principle of classes to platforms that do not natively support it. This way, you will be able to create a program using the well-known concepts of classes, types (also with generics), enums, and others even if this program runs in the browser.

haXe versions

haXe's version 1 was made available in 2006, but since 2008, we have been using haXe 2. haXe 2 brought several changes both to the language and to the library. Therefore, this is the version we're going to learn together. When you get familiar with haXe, you will understand that haXe is a very fast evolving language and sometimes, even minor versions may bring some important new features. This allows haXe to bring you more and more power as time goes by, but keep in mind that it may sometime (although this is very rare) break your code.

Note that in this book, we are using haXe Version **2.0.6**.

At the beginning of haXe, only the Flash and Neko targets were available. This means that haXe code could only be compiled to be run on Flash (AVM indeed, as that is the name of the virtual machine) or on the Neko virtual machine. Some weeks after, the already-announced Javascript target made its appearance, but with limited support (particularly in regards to closures). Nowadays, the Javascript target is really mature and can be used without any major problems. Later on, new versions of Flash were supported. All of these targets and the base of the compiler were due to Nicolas Cannasse from Motion-Twin. He is the genius mind behind haXe.

The first target that had been created by someone else was the PHP target. Franco Ponticelli released it hoping that people would experience with it, as it is much easier to get a PHP host than a Neko host. Since then, the PHP target has evolved and has become very mature and may be used in a production environment.

In 2009, the C++ generator was developed by Hugh Sanderson. This target creates C++ code from your haXe code. You can then compile it with a C++ compiler, such as GCC. Beware that it is still under heavy work. Hugh Sanderson is famous in the haXe community for making possible the development of games for iPhone and Android with haXe.

Some people are working on implementing other targets. At the time of writing this book, we know about a Java target in development by yours truly.

Also, note that even though it is not really a target on its own, it is now possible to use the Javascript target to write applications that are to be run on NodeJS.

haXe and the new models of web applications

Nowadays, we see many new and exciting web applications that are using a model that is quite new where the following three parts are communicating together:

1. The server.
2. The client-side with Flash.
3. The client-side with Javascript.

It is usually quite difficult to make those three parts communicate because each language has its own structure. With haXe, as you are using the same language everywhere, you do not have to switch between structures.

In addition, haXe has support for its own remoting; this is a way to allow you to communicate between several haXe applications as seamlessly as possible through the network or between a JavaScript and a Flash application running in the same page.

Some people have also implemented haXeRemoting in other languages making it possible to communicate with applications written in those languages.

haXe as an universal language

Nowadays, haXe can be used in order to do much more than just assisting with web applications. It can now be used in order to create, for example, applications for the iPhone or Android devices or for desktops.

In order to do so, a part of the Flash API has been implemented and made available when targeting C++. This way, it is possible to use this well-known API to create games on Windows, Linux, MacOSX, iPhone, and Android.

Getting help

As you will gain experience with haXe, you will surely encounter new problems to solve. This is an interesting part of programming, but if you find yourself really blocked, then there are several ways to get help.

Reading some documentation

There is some documentation available on the haXe website (`http://www.haxe.org/doc`) and you can also find the up-to-date API documentation at `http://www.haxe.org/api`. By the way, the haXe website is a wiki, which means that once you have registered, and when you have enough experience with haXe, you will be able to contribute to the documentation by modifying it or adding some new pages to it. You can also contribute by translating pages into your own language.

Asking questions

At some point, you will certainly want to talk with people about your problems and queries, or maybe share your thoughts about haXe. There are two main places to do that—the haXe forum and the haXe mailing list.

The haXe forum

Once you have registered an account on the haXe wiki, you can use it to post your questions on the haXe forum accessible at `http://www.haxe.org/forum`. This forum has been created primarily for newcomers who are generally more comfortable with such tools. There, you have great chances of finding other newcomers and also some haXe experts to help you.

The haXe mailing list

The mailing list was the first place where the community grew. Nowadays, it is where most haXe experts are, but they are also pleased to help newcomers there. On the mailing list, people ask questions to solve their problems, but it is also a place to discuss the language and tools evolution. The mailing list can be joined by going to `http://lists.motion-twin.com/mailman/listinfo/haxe`. If you want to know what is going to happen in the next versions of haXe, this is the place to be.

Some advice

Before you ask a question on the mailing list or on the forum, here is some advice.

- First of all, always be polite and humble, particularly since you are a new comer.
- Do not hesitate to introduce yourself in your first message (telling people who you are will help them to identify you, making you part of the community)
- Always explain what you are trying to achieve. This point is to help people help you. If they know what you are trying to achieve, they will know better how to assist you.
- Keep in mind that not all participants are native English speakers. Indeed, there are chances that most of them are not. Try to express yourself in an easy-to-understand way.
- Stay focused on one problem (or one group of linked problems) per thread. If you have several very different problems, create several threads.
- Always say what platform you are targeting (PHP, Flash, Neko, Javascript, and so on). Some problems may be platform specific.

- If you have a problem with some code, always try to reduce it to the smallest possible snippet that still reproduces the problem.
- If you send some code, always try to make it easily understandable. People who are trying to help are generally happy to do so, but if they have to guess what the variable is because you named it "avhgk" instead of "userName", they will sense that you are not even trying to help them help you.

Reading some blogs

There are many people who write about haXe on their own blog.

Nicolas Cannasse

Nicolas Cannasse, haXe's creator, maintains his own blog at `http://www.ncannasse.fr` where he discusses new and future things in haXe and its ecosystem, Flash, but also about the IT world.

Weblob

`http://www.weblob.net` is Franco Ponticelli's blog. Should I remind you that Franco Ponticelli is the PHP target creator? Although this is a low-traffic blog, you will find many interesting articles about haXe on it.

GameHaXe

Hugh Sanderson, the creator of the C++ target, maintains his blog located at `http://www.gamehaxe.com`. You will find some of his thoughts about haXe, the IT world, and experiments about game development in haXe.

He also explains how one can use haXe to target iOS and Android.

A Bug's Life

The author's blog, located at `http://www.benjamindasnois.com`, is where he talks about haXe. As he is also developing a Java target for haXe, he also writes a lot about it.

Blog.haxe.org

On `http://blog.haxe.org`, you can find an aggregate of several blogs about haXe. Some of them we have already discussed.

Helping the community

When you have some experience with haXe, you will certainly feel the need to participate in the community and help, as you have been helped. There are several ways to do this.

The mailing list and the forum

At first, and that is certainly the most obvious part, you can answer people's questions on the mailing list and on the forum.

If you do so, do not hesitate to give some advice. Share you experience—people who are migrating from the same language as you (if you are migrating) may want to hear about your experience.

Another way of participating on the mailing list is by making some proposals. If you think that you have a great idea that can make haXe even better, or can help newcomers, or everyday development with haXe, then you should propose it. People will tell you what they think about your idea and will eventually help you to make them come true.

The wiki

You can register on the wiki and write on it. There are several things that are needed on the wiki:

- Tutorials
- Updating the documentation according to changes
- Updating the documentation to reflect important things said on the mailing list
- Translating into your own language

Write on your blog or website

If you have a blog or a website, then you can write about haXe and your haXe experience on it. People always want to know how others have made the switch and how well it went. Having several sites talking about haXe helps spread the word too. Moreover, you can explain how you achieved a task and write a tutorial.

Writing libraries or tools

You can also write some libraries or tools and share them. As a developer, you will certainly use someone else's library at some point just because it is useless to rewrite things if they are already written well. So, think that someone else may also be interested in your work and distributing your libraries or tools is a great way to give back to the community.

You can also correct bugs in the haXe library and help maintaining libraries, so that they stay compatible with new versions of haXe.

You can find some libraries on `http://lib.haxe.org`.

Talking about haXe

Maybe, you are part of an association of programmers, or you are a student in an IT oriented school, or even a teacher. If this is the case, then maybe you can organize some conferences to talk about haXe. It is difficult to get people to know about a new and young language, such as haXe, because schools do not teach it and not many sites are talking about those languages.

Some people are already organizing such events (sometimes, even with workshops) and the more there are, the better it will be, because people will at least know about haXe and they can then decide if they want to use it or not (I do believe they will want to use it!).

If haXe is well-known, then you will certainly have more opportunities to use it.

What This Book Covers

Chapter 1, Getting to know haXe. In this chapter, you will be learning about haXe's history along with how to install it. You will also get some important information about how you can interact with the haXe community.

Chapter 2, Basic Syntax and Branching. In this chapter, you will learn about the basics of the haXe syntax. You will also learn how to make your programs behave differently depending on the situation.

Chapter 3, Being Cross-platform with haXe. In this chapter, you will learn how your code can be used to create programs that are going to run on several platforms.

Chapter 4, Understanding Types. In this chapter, you will learn about haXe's typing system.

Chapter 5, The Dynamic Type and Properties. In this chapter, you will learn about the Dynamic type. You will also learn how you can give your objects some properties.

Chapter 6, Using and Writing Interfaces, Typedefs, and Enums. In this chapter, you will learn what interfaces, typedefs, and enums are and how you can use them to improve your code.

Chapter 7, Communication Between haXe Programs. In this chapter, you will learn how to get your different haXe programs to communicate together by using haXeRemoting.

Chapter 8, Accessing Databases. In this chapter, you will learn how to access, read, and write to and from databases.

Chapter 9, Templating. In this chapter, you will learn how one can use templating in haXe to create, for example, views.

Chapter 10, Interfacing with the Target Platform. In this chapter, you will learn how you can access native functions and features from the platform you are targeting.

Chapter 11, A Dynamic Website Using Javascript. In this chapter, you will learn how you can use Javascript to create a website.

Chapter 12, Creating a Game with haXe and Flash. In this chapter, you will be guided through the creation of a game.

Appendix. In this section, we provide answers to the pop quiz.

3
Being Cross-platform with haXe

Targeting different platforms with the same code-base.

This chapter is about how to target several platforms from the same haXe code – the thing that haXe is known for.

haXe allows us to target several platforms; so, you may want to take advantage of this feature to be able to use your applications or libraries on several platforms. Unfortunately, there are some drawbacks, but don't worry, we will go through them and see how to work around them.

In this chapter, we will:

- See what is cross-platform in the standard library
- Talk about platform-specific packages
- Learn about their specificities
- Learn about conditional compilation

So, not only are we going to talk about being cross-platform, but also about platform-specific things. So, if you're ready, let's get started!

What is cross-platform in the library

The standard library includes a lot of classes and methods, which you will need most of the time in a web application. So, let's have a look at the different features. What we call standard library is simply a set of objects, which is available when you install haXe.

Object storage

The standard library offers several structures in which you can store objects. In haXe, you will see that, compared to many other languages, there are not many structures to store objects. This choice has been made because developers should be able to do what they need to do with their objects instead of dealing with a lot of structures, which they have to cast all the time.

The basic structures are array, list, and hash. All of these have a different utility:

- Objects stored in an array can be directly accessed by using their index
- Objects in a list are linked together in an ordered way
- Objects in an hash are tied to what are called "keys", but instead of being an `Int`, keys are a `String`

There is also the `IntHash` structure that is a hash-table using `Int` values as keys.

These structures can be used seamlessly in all targets. This is also, why the hash only supports `String` as indexes: some platforms would require a complex class that would impair performances to support any kind of object as indexes.

The Std class

The `Std` class is a class that contains methods allowing you to do some basic tasks, such as parsing a `Float` or an `Int` from a `String`, transforming a `Float` to an `Int`, or obtaining a randomly generated number.

This class can be used on all targets without any problems.

The haxe package

The `haxe` package (notice the lower-case x) contains a lot of class-specific to haXe such as the `haxe.Serializer` class that allows one to serialize any object to the haXe serialization format or its twin class `haxe.Unserializer` that allows one to unserialize objects (that is "reconstruct" them).

This is basically a package offering extended cross-platform functionalities.

The classes in the `haxe` package can be used on all platforms most of the time.

The haxe.remoting package

This package also contains a `remoting` package that contains several classes allowing us to use the haXe remoting protocol. This protocol allows several programs supporting it to communicate easily. Some classes in this package are only available for certain targets because of their limitations. For example, a browser environment won't allow one to open a TCP socket, or Flash won't allow one to create a server.

Remoting will be discussed later, as it is a very interesting feature of haXe.

The haxe.rtti package

There's also the `rtti` package. **RTTI** means **Run Time Type Information**. A class can hold information about itself, such as what fields it contains and their declared types. This can be really interesting in some cases, such as, if you want to create automatically generated editors for some objects.

The haxe.Http class

The `haxe.Http` class is one you are certainly going to use quite often. It allows you to make HTTP requests and retrieve the answer pretty easily without having to deal with the HTTP protocol by yourself. If you don't know what HTTP is, then you should just know that it is the protocol used between web browsers and servers.

On a side-note, the ability to make HTTPS requests depends on the platform. For example, at the moment, Neko doesn't provide any way to make one, whereas it's not a problem at all on JS because this functionality is provided by the browser.

Also, some methods in this class are only available on some platforms. That's why, if you are writing a cross-platform library or program, you should pay attention to what methods you can use on all the platforms you want to target.

You should note that on JS, the `haxe.Http` class uses `HttpRequest` objects and as such, they suffer from security restrictions, the most important one being the same-domain policy. This is something that you should keep in mind, when thinking about your solution's architecture.

You can make a simple synchronous request by writing the following:

```
var answer = Http.requestUrl("http://www.benjamindasnois.com");
```

It is also possible to make some asynchronous requests as follows:

```
var myRequest = new Http("http://www.benjamindasnois.com");
myRequest.onData = function (d : String)
                {
                   Lib.println(d);
                }
myRequest.request(false);
```

This method also allows you to get more information about the answer, such as the headers and the return code.

The following is an example displaying the answer's headers :

```
import haxe.Http;
#if neko
import neko.Lib;
#elseif php
import php.Lib;
#end

class Main
{

  static function main()
  {
    var myRequest = new Http("http://www.benjamindasnois.com");
    myRequest.onData = function (d : String)
    {
     for (k in myRequest.responseHeaders.keys())
     {
      Lib.println(k + " : " + myRequest.responseHeaders.get(k));
     }
    };
    myRequest.request(false);
  }

}
```

The following is what it displays:

```
X-Cache : MISS from rack1.tumblr.com
X-Cache-Lookup : MISS from rack1.tumblr.com:80
Via : 1.0 rack1.tumblr.com:80 (squid/2.6.STABLE6)
P3P : CP="ALL ADM DEV PSAi COM OUR OTRo STP IND ONL"
```

```
Set-Cookie : tmgioct=h6NSbuBBgVV2IH3qzPEPPQLg; expires=Thu, 02-Jul-2020
23:30:11
 GMT; path=/; httponly
ETag : f85901c583a154f897ba718048d779ef
Link : <http://assets.tumblr.com/images/default_avatar_16.gif>; rel=icon
Vary : Accept-Encoding
Content-Type : text/html; charset=UTF-8
Content-Length : 30091
Server : Apache/2.2.3 (Red Hat)
Date : Mon, 05 Jul 2010 23:31:10 GMT
X-Tumblr-Usec : D=78076
X-Tumblr-User : pignoufou
X-Cache-Auto : hit
Connection : close
```

Regular expressions and XML handling

haXe offers a cross-platform API for regular expressions and XML that can be used on most targets' target.

Regular expressions

The regular expression API is implemented as the `EReg` class. You can use this class on any platform to match a RegExp, split a string according to a RegExp, or do some replacement.

This class is available on all targets, but on Flash, it only starts from Flash 9.

The following is an example of a simple function that returns true or false depending on if a RegExp matches a string given as parameter:

```
public static function matchesHello(str : String) : Bool
{
   var helloRegExp = ~/.*hello.*/;
   return helloRegExp.match(str);
}
```

One can also replace what is matched by the RegExp and return this value. This one simply replaces the word "hello" with "bye", so it's a bit of an overkill to use a RegExp to do that, and you will find some more useful ways to use this possibility when making some real programs. Now, at least you will know how to do it:

```
public static function replaceHello(str : String) : String
{
   var helloRegExp = ~/hello/;
   helloRegExp.match(str);
   return helloRegExp.replace(str, "bye");
}
```

XML handling

The XML class is available on all platforms. It allows you to parse and emit XML the same way on many targets. Unfortunately, it is implemented using `RegExp` on most platforms, and therefore can become quite slow on big files. Such problems have already been raised on the JS targets, particularly on some browsers, but you should keep in mind that different browsers perform completely differently.

For example, on the Flash platform, this API is now using the internal Flash XML libraries, which results in some incompatibilities.

The following is an example of how to create a simple XML document:

```
<pages>
   <page id="page1"/>
   <page id="page2"/>
</pages>
```

Now, the haXe code to generate it:

```
var xmlDoc : Xml;
var xmlRoot : Xml;
xmlDoc = Xml.createDocument(); //Create the document
xmlRoot = Xml.createElement("pages"); //Create the root node
xmlDoc.addChild(xmlRoot); //Add the root node to the document

var page1 : Xml;
page1 = Xml.createElement("page"); //create the first page node
page1.set("id", "page1");
xmlRoot.addChild(page1); //Add it to the root node

var page2 : Xml;
page2 = Xml.createElement("page");
page2.set("id", "page2");
xmlRoot.addChild(page2);

trace(xmlDoc.toString()); //Print the generated XML
```

Input and output

Input and output are certainly the most important parts of an application; indeed, without them, an application is almost useless.

If you think about how the different targets supported by haXe work and how the user may interact with them, you will quickly come to the conclusion that they use different ways of interacting with the user, which are as follows:

- JavaScript in the browser uses the DOM
- Flash has its own API to draw on screen and handle events
- Neko uses the classic input/output streams (`stdin`, `stdout`, `stderr`) and so do PHP and C++

So, we have three different `main` interfaces: DOM, Flash, and classic streams.

The DOM interface

The implementation of the DOM interface is available in the `js` package. This interface is implemented through typedefs. Unfortunately, the API doesn't provide any way to abstract the differences between browsers and you will have to deal with them in most cases by yourself.

This API is simply telling the compiler what objects exist in the DOM environment; so, if you know how to manipulate the DOM in JavaScript, you will be able to manipulate it in haXe. The thing that you should know is that the document object can be accessed through `js.Lib.document`.

The `js` package is accessible only when compiling to JS.

The Flash interface

In a way that is similar to how the Flash is implemented in the `flash` and `flash9` packages, the `js` package implements the DOM interface. When reading this sentence, you may wonder why there are two packages. The reason is pretty simple, the Flash APIs pre and post Flash 9 are different.

You also have to pay attention to the fact that, when compiling to Flash 9, the `flash9` package is accessible through the `flashpath` and not through `flash9`.

Also, at the time of writing, the documentation for `flash` and `flash9` packages on `haxe.org` is almost non-existent; but, if you need some documentation, you can refer to the official documentation.

The standard input/output interface

The standard input/output interface refers to the three basic streams that exist on most systems, which are as follows:

1. `stdin` (most of the time the keyboard).
2. `stdout` (the standard output which is most of the time the console, or, when running as a web-application, the stream sent to the client).
3. `stderr` (the standard error output which is most of the time directed to the console or the log file).

Neko, PHP and C++ all make use of this kind of interface. Now, there are two pieces news for you: one good and one bad.

The bad one is that the API for each platform is located in a platform-specific package. So, for example, when targeting Neko, you will have to use the `neko` package, which is not available in PHP or C++.

The good news is that there is a workaround. Well, indeed, there are three. You just have to continue reading through this chapter and I'll tell you how to handle that.

Platform-specific packages

You've already understood that there are some platform-specific packages, let's have an overview of them.

JavaScript

The platform-specific package for JavaScript is the `js` package. It basically contains only `typedefs` representing the DOM and has a layout that's not comparable to any one of the other platform-specific package.

Flash

As we've already mentioned, Flash has two platform-specific packages: the first one is `flash` and it is used when targeting Flash up to Flash 8, or else, you use the `flash9` package (but access it by writing `flash`). Those packages contain definitions for the external classes that are natives in Flash. Their layouts are not comparable to any other platform-specific packages.

Neko

Neko has its own package named `neko`. It contains a lot of useful classes allowing for example, sockets communication, file-system access, console input and output, and threads management.

Its layout is comparable to the layouts of the C++ and PHP packages.

PHP

The `php` package is a PHP-specific package. It allows us to manipulate files, write, and read from the console and open sockets.

Its layout is comparable to the layouts of the C++ and Neko packages.

C++

C++ has its specific package named `cpp`. It allows one to access the filesystem, read, and write from the console and open sockets.

Its layout is comparable to the layouts of the `neko` and `php` packages.

Conditional compilation

haXe allows you to do conditional compilation. This means that some parts of the code can be compiled and others can be ignored depending on the flags you give to the compiler and depending on the platform you're targeting.

Conditional compilation instructions always begin with the `#if` instruction.

Conditional compilation depending on flags

You can define a flag on the command line by using the `-D` switch. So, to define the `myway` flag, you would use the following: `-D myway`.

Then, you use it in the following way:

```
#if myway
//Code here compiled only if myway is defined
#else
//Code here compiled in the other case
#end
```

There is also a `not` operator:

```
#if !myway
//Code here compiled only myway is not defined
#end
```

Conditional compilation depending on the target

Conditional compilation depending on the target basically works in the same way, except that the name of the target you are compiling to automatically gets set. So, you will have something like the following:

```
#if cpp
//C++ code
#elseif neko
//Neko code
#elseif php
//PHP code
#else
//Code for other targets
#end
```

By the way, did you notice the presence of `#elseif` in the code? It could prove to be pretty useful if you need to create a different implementation of the same feature depending on the platform.

The remap switch

The compiler has a `remap` switch that allows us to rename a package, for example, imagine the following code:

```
//Some previous code
myPack.Lib.println("Hello remapping!");
//Some code following
```

This would obviously not work because the `Lib` class is not in the `myPack` package. However, if you compile with the switch `--remap myPack:neko` when compiling to Neko, this will work. The reason is simple: anywhere you have used the package name `myPack`; the compiler will replace it with Neko, therefore using the correct package.

This can be used to make some cross-platform code.

On the other hand, it has a big draw-back: when you remap a package, all of the modules it contains are remapped. This actually means that you won't be able to use something that is platform-specific inside this package.

Coding cross-platform using imports

We've already seen the `import` keyword that allows us to import all the classes of a module. This keyword can be used to write some cross-platform code. Indeed, it is the preferred way to do so, when working with packages that have the same layout. It has the advantage of being easily readable and does not mean adding three lines every time you want to access a platform-specific package.

So, here's a simple example: imagine you want to target Neko and PHP and want to use the `println` method available in `neko.Lib.println` and `php.Lib.println` (these methods allow you to write a line of text on the `stdout`). As you can see here, regarding what we are going to use, the `neko` and `php` package have the same layout (indeed, they have the same layout for quite a lot of their classes). So, we can write the following code:

```
#if php
import php.Lib;
#elseif neko
import neko.Lib;
#end

class Main

{

   public static function main()

   {

      Lib.println("Hello World");

   }

}
```

In this example, we can use the `php.Lib` or `neko.Lib` class by simply using its name (Lib), as the good one is imported depending on the current compilation target. If we didn't do that, the code would have looked like the following:

```
class Main
{
   public static function main()
   {
      #if php
      php.Lib.println("Hello World");
      #elseif neko
      neko.Lib.println("Hello World");
      #endif

   }

}
```

As you can see, this is pretty difficult to read although here we have almost no logic and only one instruction.

Time for action – Welcoming the user on Neko & PHP

We are going to write a simple program that asks the user their name and prints a little welcome message along with the date. This program should work with the same codebase on Neko and PHP.

The following are the steps you should follow to write this program:

1. Identify what classes you will need to print text and read from the keyboard.
2. Identify what classes are in a platform-specific package.
3. Add imports for those classes.
4. Run and test.
5. You should get the following output:

The following is the final code you should have produced:

```
#if neko
import neko.Lib;
import neko.io.File;
#elseif php
import php.Lib;
```

```
import php.io.File;
#end

class Main
{

   static function main()
   {
      //Print a prompt
      Lib.print("Please enter your name: ");
      //Read the answer
      var name = File.stdin().readLine();
      //Print the welcome message
      Lib.println("Welcome " + name);
      //Store the date
      var date = Date.now();
      //Concatenate a message with the two elements from the date
        object
      Lib.println("Time is: " + Std.string(date.getHours()) + ":" +
        Std.string(date.getMinutes()));
   }

}
```

What just happened?

I'm pretty sure that with the commented code, you do understand what's going on, but let's clarify each step we have asked you to think about:

- Needed classes
  - To write to the console, we will use the `neko.Lib` class
  - To read from the keyboard, we will use the `neko.io.File` class
  - To know the date and time, we will use the `Date` class
- Identifying platform-specific classes
  - The `neko.Lib` and `neko.io.File` classes obviously are platform-specific; their corresponding classes for PHP respectively are `php.Lib` and `php.io.File`
  - The `Date` class is platform-independent and therefore, we won't need to do anything special with it

- Imports

  We just have a section to do the good important according to the target platform:

```
#if neko
import neko.Lib;
import neko.io.File;
#elseif php
import php.Lib;
import php.io.File;
#end
```

  This one is pretty short, but sometimes you may have many of those classes.

Pop quiz – Writing cross-platform code

1. What is the main drawback of using conditional compilation directly inside your code?
   a. It is more difficult to read
   b. It takes more time to compile
   c. It is less efficient at runtime
   d. It simply doesn't work
2. How does one access the `flash9` package when targeting Flash9+?
   a. Use the `–remap flash9:flash` switch
   b. Simply access it by writing flash9
   c. Access it by writing flash; the compiler knows what to do
   d. Use an `import` at the beginning of the file

Have a go hero – Handle XML

Now, let's sum-up all we've done and more particularly the XML part.

Imagine you want to create a tool, which allows you to create pages. Each page will be composed of one or several layers and each layer will have an ID.

You want to save and load a page in an XML file that will look like the following:

```
<page name="My First Page">
   <layer id="layer1">
      [content]
   </layer>
   <layer id="layer2">
```

```
        [content]
    </layer>
</page>
```

For the sake of this example, we will not generate the content parts.

Time for action – Reading from the XML file

We are going to create a function to read from an XML file. So, let's proceed step by step.

1. First, create the `Layer` class as follows:

```
class Layer
{
    public var id : String;

    public function new()
    {}
}
```

2. Now create the `Page` class as follows:

```
class Page
{
    public var name : String;
    public var layers : List<Layer>;

    public function new()
    {
    }
}
```

3. Now, let's create a function to create a page from an XML file. Add the following function to the `Page` class:

```
public static function fromXMLFile(path : String) : Page
{
    var nPage = new Page();
    var xmlDoc = Xml.parse(neko.io.File.read(path, false).
      readAll().toString());
    nPage.name = xmlDoc.firstElement().get("name");

    return nPage;
}
```

4. As you can see, it is not yet complete. We have to parse a layer from the XML file, so let's do it now. Add the following function in the `Layer` class:

```
public static function fromXMLNode(node : Xml)
{
   var nLayer : Layer;
   nLayer = new Layer();
   nLayer.id = node.get("id");

   return nLayer;
}
```

5. Now, in the `fromXMLFile` function, let's add some code to iterate over the nodes named `layers`, and parse them using the `fromXMLNode` function:

```
public static function fromXMLFile(path : String) : Page
{
   var nPage = new Page();
   var xmlDoc = Xml.parse(neko.io.File.read
     (path, false).readAll().toString());
   nPage.name = xmlDoc.firstElement().get("name");
   for(l in xmlDoc.firstElement().elementsNamed("layer"))
   {
     nPage.layers.add(Layer.fromXMLNode(l));
   }
   return nPage;
}
```

What just happened?

As you see, we are simply expecting our document to respect the structure that we have defined; therefore, it was pretty easy to parse our XML file.

Time for action – Writing to an XML file

1. We want to be able to write an XML file. To write the XML file, we will follow the same idea. Add the following function in the `Layer` class:

```
public function toXMLNode() : Xml
{
   var nXml = Xml.createElement("layer");
   nXml.set("id", this.id);
   return nXml;
}
```

2. Now, add the following code in the `Page` class:

```
public function toXMLFile(path : String)
{
   var xmlDoc = Xml.createDocument();
   var pageNode = Xml.createElement("page");
   pageNode.set("name", this.name);
   xmlDoc.addChild(pageNode);

   for(l in layers)
   {
      pageNode.addChild(l.toXMLNode());
   }

   neko.io.File.write(path, false).writeString(xmlDoc.toString());
}
```

You can now save a page and all its layers by calling `toXMLFile`.

What just happened?

We have created a simple of the function in order to be able to save our page as an XML file by calling `toXMLFile`.

Testing our sample

The following is a sample `main` function if you want to try this code. It may also help you to understand how to use it:

```
public static function main(): Void
{
   trace('Hello World');
   trace(Page.fromXMLFile("/Users/benjamin/example.xml"));
   trace(Page.fromXMLFile("/Users/benjamin/example.xml"));

   var p = new Page();
   p.name = "Page de test";

   var l1 = new Layer();
   l1.id="l1";
   var l2 = new Layer();
   l2.id="l2";

   p.layers.add(l1);
   p.layers.add(l2);

   p.toXMLFile("/Users/benjamin/page1.xml");
}
```

Making it cross-platform

This code works on the Neko target. Using what we've learned about using imports to make code cross-platform, you can make this code work on PHP very easily. That's something you should try on your own to learn!

Summary

In this chapter, we've learned about how to do some cross-platform programming with haXe.

Specifically, we covered what is in the Standard Library, how to use conditional compilation and other ways to write cross-platform code, and how to handle XML files.

Now, let's move on to our next chapter in which we will talk about the typing system!

Where to buy this book

You can buy haXe 2 Beginner's Guide from the Packt Publishing website:
`http://www.packtpub.com/haxe-2-beginners-guide/book`

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com