Gaudi. A platform agnostic build tool for the Java virtual machine.

Sam Saint-Pettersen,
BSc (Hons) Computing (University of Worcester)

Abstract

This paper looks at the development of a software build tool, written in a combination of Scala and Java to run on the Java virtual machine. It compares Gaudi to other existing tools such as Apache Ant and GNU Make and contrasts the differences between these applications and Gaudi and the software methodology employed for its development.

Additionally, some of the implementation specifics are looked and the results of a short test deployment are examined and any conclusions on the aim of the project and what was actually achieved are acknowledged.

Acknowledgements

I would like to thank my dissertation supervisor, Dr. Hubert. P.H. Shum for his support and guidance offered while I prepared this piece of work. I also extend my thanks towards Dr. Colin B. Price for supporting this choice of independent study, the other numerous people who showed encouragement and interest in *Gaudi* and last but not least thank you to everyone who took part in the test deployment testing .

The image of the hammer on the front of this paper is courtesy of James Bowe (*http://www.flickr.com/photos/jamesrbowe/6371964415*) and was released under Creative Commons Attribution License.

http://gaudi-build.tk

Table of Contents

Introduction to Gaudi

Overview

Gaudi is a platform agnostic build tool written in the Scala (EPFL, 2011) and Java (University of Michigan, 1997) programming languages, which run on a Java virtual machine [JVM]. (Lindholm et al., No date).

As it is a build tool, it is named after an architect; *Antoni Gaudí*, the designer of the famous *Sagrada Familia*. (Craven, No date).

Gaudi can be thought of as being similar to Apache Ant (Apache Foundation, 2011) in that it too abstracts commands related to building software away from the operating system (e.g. `:erase` instead of `rm` on Unix-likes or `del` on Windows); but differs in that:

1. Its build files are based on a JSON format rather than an XML format. (Json.org, No date).

2. It is not tailored to offer advanced features for a particular programming language, unlike Apache Ant which is highly specialised for Java development.

Aims

The aims of Gaudi as a project of this dissertation were to:

- ✓ Implement enough core commands for Gaudi to be able to build *itself*.

- ✓ Implement enough core commands for Gaudi to be able to *configure* the build and Apache Ant in order to build Gaudi. This purpose is currently served by *configure* scripts.

- ✓ Implement server/client functionality for Gaudi to receive commands and relay responses via telnet or SSH to perform build operations or *actions*; each consisting of individual *commands.* Also, to implement the loading and execution of remote (to the program) build files.

- ✓ Implement command extensibility. (E.g. bespoke commands).

- ✓ Implement plug-in system for advanced extensibility.

- ✓ Aim to be compatible with a wide range of JVM implementations.

Strengths

Gaudi has the following strengths over similar build tools:

- ✓ Concise syntax for build files (over more verbose build files of Apache Ant).

- ✓ Written in Java and Scala; it will run on any operating system which runs a compatible Java virtual machine (JVM).

- ✓ Small core executable ("Unbundled" JAR file is ~ 37 kilobytes[1]).

- ✓ Open-source software relying on open-source libraries which may be shared with other applications utilising a Java virtual machine.

Current status

Gaudi can currently be build using Apache Ant, after being configured with made-for-purpose configuration scripts. Within the project source, I have included an example build file runnable with Gaudi to successfully to compile a C++ Hello World program using the GNU C/C++ Compiler [GCC]. (Free Software Foundation, 2012).

[1] The JAR file (Oracle Corporation, 1999) without the libraries included; as of this paper's date. "Unbundled" means *not* packaged with its dependent libraries using *One Jar.* (Tuffs, 2004).

Organisation

I have written the core source code for the program [2] from the ground up; but have also used third-party, open-source libraries available from the Internet. This information is included in the appendices. (*see* Appendix I).

As for the program, I have always been and currently am the sole developer, at least for now during the duration of this project.

Background of project

Gaudi was originally a personal project of mine which I developed while learning the Scala programming language, back in the summer of 2010. During this project I have and may continue to refer to *Practical Development Environments* (Doar, 2005) for better understanding and inspiration on the mechanics of build tools.

Literature Review

Overview

This literature review details the research into different software build tools with an aim into the implementation of an all-new software build tool, Gaudi.

In this literature review, I look at the following:

1. Software development methodologies and how they will apply to the implementation of Gaudi, specifically the application of the Scrum methodology (a subset of Agile).

2. Existing software building solutions and how Gaudi compares to them.

[2] Any small portions of source code taken (almost) verbatim from point of reference are annotated in the source code, and are subject to change in future development.

3. Implementation choices for Gaudi development.

Software development methodologies

I will develop Gaudi using an agile software development philosophy. Agile software development philosophy developed over time to replace the aging and often impractical Waterfall methodology. Whereas the traditional Waterfall methodology confines the developer to build software in a sequential and phased manner, agile allows software to be developed in a more flexible manner which allows for changes in circumstances during development. (Szalvay, 2004).

The Waterfall method "has dominated software development projects for decades". As defined by Winston Royce in 1970, it defines five stages. Each following on from another and each depending on the completion of the stage before. The five stages are: the *requirements* stage, the *design* stage, the *implementation* stage, the *verification* stage and finally the *maintenance* stage. (Serena Software Inc., 2007).

Scacchi, 2001 cites Royce, 1970 in describing the classic software life cycle as being "a simple prescriptive waterfall software phase model, where software evolution proceeds through an orderly sequence of transitions from one phase to the next in order". Such a model is identified as being most useful in structuring and staffing large software development projects in complex organisational settings. (Scacchi, 2001 citing Royce, 1970 and Boehm, 1976).

In the *requirements* stage, functions and constraints of the system/application ("the requirements") for the end user are ascertained. In the Waterfall model, it is important to appreciate that all requirements are defined only at this stage and cannot change without upsetting the work flow. The *design* stage refers to the requirements specifications, from the *requirements* stage, in order to produce a system design. This design serves as an input for the next phase. In the *implementation* stage, from the previously produced system design, the development of code begins. This code is usually written in a modular fashion into units with each unit tested independently for its compliance with its own sub-specifications ("Unit testing"). In the *verification* or *integration* stage, the units are integrated into the full system and tested that all functionally co-ordinate and that the system behaves in compliance with the specifications. After any bugs are ironed out and fully successful testing is achieved, the software is ready for the customer. In the *maintenance* stage, any problems encountered within the system developing during practical use which had not previously

occurred during the development life cycle (stages 1-4) are patched. This stage virtually never ends in the waterfall model and is recurrent. (Parekh, 2011).

The Waterfall model corresponds to the traditional Software Development Life Cycle (SDLC), development. Evolutionary development contrasts with the SDLC in that it is non-linear with only a first set of outline requirements identified at a time, with the assumption that these requirements will change. SDLC requirements are said to be "frozen", as explained earlier, they cannot change. (Avison & Fitzgerald, 2006 citing Pressman, 2004).

The Spiral model adopts the concept of a series of incremental releases or developments, which can be seen as development spirals outwards from the centre, with each spiral cycle representing an iteration of the system. The spiral is drawn in a clock-wise direction. Since these spiral cycles are iterations but also sequential, this model could be described as a reconciliation of both the traditional SDLC and the evolutionary development approaches. In each cycle of the spiral, there are four quadrants – *Planning, Objectives*, *Risks* and *Development.* (Avison & Fitzgerald, 2006 citing Boehm, 1988).

The stages of the Spiral model correspond to the stages in the Waterfall model: planning, analysis (*requirements*) and design and implementation. (The Pennsylvania University, 2008).

The Spiral model evolved over several years from the Waterfall model, based on numerous refinements applied to large government projects. (Boehm, 1988).

Agile development refers to a family of related, iterative and evolutionary development methodologies. Some of these are: Scrum, XP (Extreme Programming), DSDM (Dynamic Systems Development Model) and Lean Software Development. (Waters, 2011).

Contrary to the Waterfall model, because of its iterative processes of rapid prototyping and the minimal documentation involved, agile software methodologies allow developers to produce higher quality software more quickly. (Livermore, 2008).

In fact, one of the most import differences between Agile and Waterfall is that agile development has iterations rather than waterfall's stages. The output of which for agile is working code that can be used when evaluating and responding to changing or evolving user requirements. Additionally, agile development contrasts with Spiral in that after every iteration fully working code rather than a prototype is produced. Agile does not reconcile the traditional SDLC with evolutionary development. (Serena Software Inc., 2007).

Also in pertinence to agile development - Begel & Nagappan, 2007 note that “Agile [software] development methodologies [ASD] have been gaining acceptance in the mainstream software development community”. Begel and Nagappan undertook research at Microsoft to gauge the opinions and/or adoption of agile methodologies across the development, testing and management personnel there. From their web-based survey, they received a seventeen per cent response rate or nearly 500 responses, making it “one of the largest respondent populations for a survey of software development at Microsoft”. From the survey responses, they found that approximately a third of the respondents used ASD in some form. Scrum was found to be the most popular ASD methodology, and ASD as a group of methodologies was noted to be a new phenomenon at Microsoft with most projects having employed it for less than two years. Finally, those people using ASD were found to have an overwhelmingly positive opinion of it. (Begel & Nagappan, 2007).

In developing Gaudi, specifically the Scrum methodology will be used. The Scrum methodology utilises an iterative and incremental approach to “optimize predictability and reduce risk”. (Schwaber & Sutherland, 2011).

Scrum consists of the following principles: *Product Backlog* (a list of tasks to complete is drawn up)*, Product Owner* (the project co-ordinator, him or her in charge)*, Sprint* (setting of a fixed deadline in the foreseeable future)*, Sprint Planning/Sprint Backlog* (estimation of how much can be completed by the deadline)*, Daily Scrum* (check the list every day to ascertain progress towards deadline)*, Sprint Review* (evaluate how work went and discuss if there is anything that could be done differently in the future)*, Iterate all* (repeat the principles). (Waters, 2007).

Existing software build tools and Gaudi

Apache Ant is an open-source build tool available as a command line tool and associated Java library. It is developed by the Apache Foundation. (Apache Foundation, 2011).

Ant was originally designed as an extensible replacement for *make*, but “without make’s wrinkles”. Its build files are written in XML and the core of Ant and its tasks are written in Java. An Ant build file defines the `<project>` and `<target>` elements, which define a project and targets for that project passable on the command line, respectively. Targets will contain one or more `<task>` elements, which are elements to execute actual commands during a build. Ant build files are, by convention, named *build.xml*, and this is the first file in the current working directory Ant will look for on invocation. (Doar, 2005). Because it is written in and has close integration with Java, Ant is commonly used with Java-based projects. (Vogel, 2011).

SCons is a build tool, written in Python, which uses “configuration files” to build software. SCons utilises the use of Python methods or objects to build target files and features a modular design. Like Apache Ant, it is user extensible. (Gagon, 2009).

SCons was conceived as a rewriting of an earlier tool, Cons, into Python. SCons includes support for many compilers, including but not limited to, GCC, the official Java compiler and the Microsoft Visual C++ compiler. Features of SCons include cross-platform build files, automatic dependency checking and signature files to ensure file robustness, parallel build support and programming language capabilities (from Python) such as conditional branching and in-built debugging. The default build file for SCons is called *SConstruct*. (Doar, 2005).

GNU Make is a build tool, developed by the Free Software Foundation, which “controls the generation of executables and other non-source files of a program from the program’s source files.” (Free Software Foundation, 2010).

GNU Make is the Free Software Foundation’s implementation of the historical *make* utility, first written in 1977 by Stuart Feldman. Make is the original build tool and probably the most commonly used, in some implementation such as GNU Make or Microsoft’s NMAKE (Microsoft, 2012). Make uses Makefiles, written in make’s own language (the default file is named *Makefile*) to build software. Makefiles are typically implemented to use the native commands of the host operating system, which makes them susceptible to portability problems. Due to its ubiquitous use and lengthy history, make has inspired various spin-offs based on its concepts for specific languages or tools such as Ruby’s rake. (Doar, 2005).

Gaudi will use a build file format based on the JavaScript Object Notation format (JSON), a popular alternative to XML as a data exchange format. JSON is considered to be considerably less verbose than XML or similarly mark-up based formats. (Shin, No date).

Gaudi aims to be most similar to Ant in its core conceptual design; i.e. The use of an existing data format (JSON as Ant is to XML) and platform portability (abstraction of file manipulation commands away from the host system), but more similar to make in its formatting of make files (JSON files that are similar in resemblance to Makefiles). Like Ant and SCons, Gaudi also aims to be extensible with support for user plug-ins, but this full extensibility will most likely not be complete by the end of this independent study. Gaudi’s default build file will be named *build.json.*

Implementation choices for Gaudi development

Python and Perl were used to write the build configuration scripts. The configuration scripts are responsible for generating the final Ant build file and shell script or batch file necessary to execute a Gaudi build and the shell script or batch file to bootstrap Gaudi as a JAR-based application.

A configuration script is first run to typically check for dependencies, allow customization via parameters and to generate a build file compatible with the current system. This behaviour is typical to programs using a Make variant as the build tool (Veselosky, 2011), but is adopted here with the Ant build tool.

Python and Perl were used as they are widely available across platforms and on the Unix platform often are preinstalled, so therefore are available straight away for a Gaudi build on such platforms. (Perl.org, 2011 & Python Software Foundation, 2011a). Additionally, both Python and Perl abstract file manipulation commands away from the host system, so actions necessary for configuration such as file deletion work across platforms. (Python Software Foundation, 2011b).

Instead of Python or Perl, I could have used Tcl ("tickle" or “tee-see-ell”), another scripting language. (Tcl Developer Xchange, No date a). Like Python or Perl, Tcl also has the advantage of being able to abstract shell commands away from the host operating system. (Hume Integration Software, 2011). Additionally, like Python and Perl, Tcl is typically preinstalled on many Unix-like systems. (Tcl Developer Xchange, No date b).

However, I did not choose Tcl as I was not familiar with it and preferred Python due to my better familiarity with the latter.

I could have also used Autoconf to perform the build configuration duties. Autoconf is a GNU Autotools suite utility that generates a configuration script, *configure* that can be run to check for dependencies and otherwise configure a build for use with the historic *make* utility, the generated *configure* script is a Unix-like Bourne-compatible shell script and will run on Unix-like platforms such as Linux and Mac OS X. (Doar, 2005). I decided instead to use Python and Perl for this purpose, because they would be cross-platform and not dependent on a Unix-like Bourne compatible shell.

Another project which Gaudi aims to be compatible with, Jato VM (an implementation of the Java virtual machine) demonstrates the typical configure and make process in its instructions for building the Java class libraries it depends on. (Enberg et al., 2011).

Apache Ant was chosen as the build tool to build Gaudi, as it is similar in essence to what Gaudi itself is and serves as a model implementation of such a build tool, and is well suited to building a Java-targeted project, which Gaudi is.

Apache Ant also runs on a Java virtual machine (JVM) and as such is widely supported across the most popular desktop operating systems, which beautifully facilitates the building of a cross-platform build tool. Java and by extension other JVM-based languages therefore adhere to the Java platform's "Write Once, Run Anywhere" principle. (Kramer, 1996).

Similarly, to the configuration scripts between writing in scripting languages that abstract away from the host platform, Apache Ant uses file manipulation commands that run across different operating systems system via a Java virtual machine. (Chapman, 2003).

It also had built-in or readily extendable support for JVM languages. Gaudi is written in Scala and Java. These were the main advantages of using Apache Ant over a Make variant. Make does not offer the close integration of JVM languages and is very reliant on Unix-like shell commands to run effectively – like a shell script, the use of *rm* or similar commands typical only readily available to Unix-like systems and not Windows or other systems. Such commands are not abstracted as they would be by Ant. (Waikato Linux Users Group, 2005).

Implementation

Overview

Design Philosophy

The Gaudi implementation was modelled on that of Apache Ant. Gaudi too abstracts commands away from the host system and lot of interfacing between Gaudi and the host system is offered through the underlying Java virtual machine implementation.

Pragmatic approach to design philosophy

File manipulation for instance is offered partially via the Apache IO Commons [JAR] library. Especially the `FileUtils` class. (Apache Commons, 2012).

The core Gaudi application is implemented as a collection of interoperating classes (instigated as objects) and interfaces written in the Scala and Java programming languages. A UML class diagram of the classes and interfaces which constitute Gaudi are included later on in this paper.

Build file syntax

Gaudi's build file format is based on JavaScript Object Notation [JSON], a data format language derived from JavaScript. (Json.org, No date). Such a build file takes the following format, where italics are specific to a given build file. Highlighted text indicates what is needed to extend the file and the bold ellipses (**…**) indicate where more commands or actions can be added from.

```
{
      "preamble":
            "source": "source files(s) or wildcard for source file(s)"
            "target": "target to produce from source, typically a binary",
            "cc": "compiler or another tool used to generate target"
      },

      "build": [

            { "command": "parameters for command" },
            …
      ] , …
}
```

All build files begin with the *preamble* block which defines:

- The *source* files (s) to use to build the *target*.
- The *target* file(s) to generate from the source using *cc.*
- The *cc* (meaning *C compiler*, a convention adopted from *Makefiles* (Patel, No date)) indicates the compiler or similar tool needed to generate the *target* file(s).

All build files also contain the *build action* block which includes a sequence of instructions. Sequences are indicated with square brackets ("[ ]") and *commands* are included within curly braces ("{ }") and followed by a comma (",") if there are more commands following.

Plug-in file syntax (Groovy programming language)

A plug-in written in the Groovy programming language (Glover, 2004) , takes the following format. Italics indicate a given plug-in's specifics:

```
import org.stpettersens.gaudi.GaudiPlugin

public class PluginName extends GaudiPlugin {

	PluginName() {

		Name = "Name of plug in name"

		Action = "What the plug-in will do; its action."

		Version = "Version number, e.g. 1.0"

		Author = "Name of plug-in author"

		Url = "Url for plug-in documentation/info/download page, etc."

		Initable = true

	}

	public void run() {

		Actions implemented by the plugin. E.g. Gaudi commands…

	}

}
```

Plug-ins reside in the *plug-ins/* subdirectory of the Gaudi installation and when Gaudi is built with plug-in support, they can be invoked like so with the –p switch:

```
gaudi -p pluginName
```

Example plug-ins are included in Appendix VII.

Actions

Actions are just a group of related *commands* in a block other than *preamble.* The default *build* block is an example of an action. For instance, the action *clean* can erase all generated files from a build. It would be invoked like so (assuming we are in the current working directory where the *build.json,* the default searched for Gaudi build , resides and we have a *clean* action defined within the build file. It is not required like *build* but is recommended):

```
Gaudi clean
```

This will invoke the clean action defined in the build file.

Commands

Gaudi implements currently the following commands. Each command is abstracted away from the host operating system in compliance to the *design philosophy* and are handled directly through the Java virtual machine [JVM], except for command that begin with :*x* (e.g. :xstrip) which depend on the existence of a specific program and of course the execution command (:exec) which invokes the specified program such as a compiler or linker.

Gaudi commands are available in two contexts:

1. Build file syntax.
2. Direct invocation syntax.

Build file syntax refers to the way a command will be used within a JSON-based build file as used by Gaudi. For example, to invoke the execution command in a build file – something like this would be written within the build file (where the curly braces indicate a block):

```
{ “exec”: “my_program” }
```

Direct invocation refers to the way a command can be executed directly (e.g. on the command line) as a parameter to Gaudi. For instance, the execution command mentioned for build file syntax could also be invoked like so, when invoking Gaudi itself:

```
gaudi “:exec my_program”
```

Notice that in direct invocation, the colon (“:”) proceeds the command, contrary to build syntax where it follows a quote-enclosed command.

Note that all Gaudi commands in build file syntax take quoted string as parameters enclosed in braces as part of a command block; or two unquoted literals (command and parameter) surrounded within quotes in direct invocation syntax. All commands work in both contexts, but commands such as `:help` are more obvious for use in a direct invocation context, although use in a build file context is also perfectly valid.

Gaudi includes the following commands with their associated parameters (listed in alphabetical order):

- `append: plain_text_file.ext >> single line message / variable(s)` – Appends a single line message and/or variable(s) to a plain text file (typically *.txt or *.log file format) relative to the current working directory by default. Creates file if non-existent; otherwise appends to existing file, keeping existing lines beforehand. Build file syntax example: `{ "append": "output.txt>>Hello there, World!" }`. Since this command appends to a plain text file, it is well suited to logging a build process.

- `clobber: plain_text_file.ext >> single line message / variables(s)` – Similar to append, but writes a single line message and/or variable(s) to a plain text file, overwriting any existent text in an existing file. If the file does not exist, the effect will be the same as `append`. Build file syntax example: `{ "clobber": "output.txt>>This will overwrite any existing text that was in this text file." }`.

- `copy: original file >> file destination` – Copy a file to a file with a new name or to another directory (typically under the current working directory). Build file syntax example: `{ "copy": "my_file.zip">>"folder_of_zips/" }`, which would copy the file *my_file.zip* to the specified of directory *folder_of_zips*.

- `decode: literal encoded string / file containing encoded string` – Decodes a short string encoded with a Caesar-style shift cipher (Savarese & Hart, 1999) to the same plain text file containing the original encoded string. Implemented in Gaudi to work with trivial passwords, et cetera. Decode with `decode` command; see above. Example build file syntax: `{ "decode: "encoded.txt" }`. Re-encode with encode command; *see* below. *TO (PROPERLY) IMPLEMENT.*

- `echo: single line message / variable(s)` – Display a single line message and/or variable(s). This can, and is intended to be, used for documenting your build process output. For instance say what you are attempting to currently accomplish in your build *action* such as *Compiling Hello World program.* Example build file syntax: `{ "echo": "Compiling Hello World program…" }`.

- `encode: plain_text_file.ext >> "short string to encode"` – Encodes a short string with a simple Caesar-style shift cipher to a plain text file. Implemented in Gaudi to work with trivial passwords, et cetera. Decode with `decode` command; see above. Example build file syntax: `{ "encode: "encoded.txt>>caesar" }` .

- `erase: file/wildcard for file` - Erase a non-application (not including shell scripts) file in the current working directory. Example build file syntax: `{ "erase": "file_to_delete.txt" }`.

- `exec: app/external process (and parameters)` – Execute an external application or process. For example, the GNU C++ compiler could be invoked using the example build file syntax: `{ "exec": "g++ hw.cpp -o hw" }`. This will compile the program C++ and link it into an executable *hw* (or *hw.exe* on Windows).

- `:help command` - Like Linux's *man* command (Haas, No date a), this command takes the queried command as a parameter and displays on-screen help on the purpose of that command and examples on how to use it. Direct invocation example: `gaudi ":help echo"`, will instruct the user how to use the echo command. *TO IMPLEMENT.*

- `list: file/folder or wildcard for`– List all folders, files (or by wildcard) for the parameter object existent within the current working directory. Example build file syntax: `{ "list": "*" }` , which would list all files and folders within the current directory because of the asterix (*) wildcard, a common computing convention. (Thomson Reuters, 2009). *TO IMPLEMENT.*

- `mkdir: directory name` – Create a new folder/directory of parameter name if one does not exist in current path; otherwise throw an error message. Example build file syntax: `{ "mkdir": "my_new_folder" }`.

- `notify: notif_subsystem >> message / variable(s)` – Sends a single line message and/or variable(s) to the system notification application/subsystem. Notif_subsystem may be `default`, `growl` (Growl Team, 2012) or `gtk`. Example build file syntax: `{ "notify:" "growl>>"Hello, from Growl notifications." }`, which would display the message with the Grown notification application. This command will be implemented using the notification systems existing APIs for Java. *TO IMPLEMENT.*

- `move: original file >> file destination` – Move/rename a file to a file with a new name or to another directory (typically under the current working directory). Build syntax example: `{ "move" : "my_old_name.zip">>"my_new_name.zip }`, which would move/rename the file *my_old_name.zip* to *my_new_name.zip*. This command is analogous to Unix/Linux's *mv* command (Haas, No date b).

- `rcopy: folder containing files >> destination` – Recursively copy a folder of files (analogous to Windows' *XCOPY* command (Microsoft, No date)) to another directory [destination]. Example build file syntax: `{ "rcopy": "my_cool_src_files/">>"other_cool_code/" }`. *TO IMPLEMENT.*

- `xstrip: executable to strip` – Strip an executable of symbols. This command proceeds with x , a design convention indicating that it is dependent on another external program to Gaudi; in this case the strip (strip.exe on Windows) program, an implementation of which exists within the GCC/MinGW packages (Christias, 1994). Example build syntax: `{ "xstrip": "hw" }`, which will strip the binary hw or hw.exe depending on host operating system.

Classes

The implementation of Gaudi delegates different related areas of functionality to its part of the application. Gaudi implements ten distinct classes (six of which are written in Scala and four of which are implemented in Java); and one interface written in Java.

Each class and its purpose and responsibilities are described below.

1. **GaudiBase** class. As the name suggests this is the base class for the application. The purpose of a base class is to provide expose common functionality (methods) and properties (attributes) to the classes sub-classing it through the object-orientated programming principle of inheritance. (Russell, 2011). The classes that inherit from GaudiBase are sub-classed to provide common methods, but not properties. All attributes in the class are private.

   Any class that inherits from GaudiBase does not also inherit from any other class as Java (and by its relationship with Java - Scala) does not support multiple inheritance. (Bergin, 2000). This class is written in Scala.

   GaudiBase provides commonly needed methods, such as one to write to dump information to a text log and to write to a text file in general (also used by the logging method internally). Both of these methods use the *protected* access modifier.

   Protected access modifiers allow the methods to be used in subclasses. For this reason, the methods in GaudiBase are *protected* as they are intended to be used only by the classes that need them and as such sub-class GaudiBase. In contrast, the *private* access modifier makes the method only accessible within its own class. In Scala, when an access modifier is omitted, the method will be *public* by default. *Public* methods are accessible globally through any instance of the class. (Tutorials Point, No date).

2. **GaudiApp** is a class with a single instance or a singleton. Since this class is written in Scala, it is defined using Scala's singleton definition keyword *object***.** (Odersky et al., 2006).

   GaudiApp contains the entry point (a *public*) method, *main* for the application. Unlike in Java, this method is not declared as static as it appears under a class with only a single instance. Conceptually, the Scala language does not use the *static* keyword as a class with only a single instance is defined with the *object* keyword and contained methods are implicitly the equivalent of static. (Odersky, 2012).

   This class also contains methods that perform operations tied to the command line parameters the application takes. This includes *private* methods to load a build file, run a command (utilizing the *GaudiBuilder* class, see below) and to display occurring errors, version and usage information.

3. **GaudiBuilder** class. As name suggests this class is responsible for building something, specifically the target(s) (the software libraries, executables or documentation, et cetera) outlined in the build file the application uses as input from the source files referenced. Also written in Scala, *GaudiBuilder* defines the methods to extract and execute *actions* and *commands* contained within the loaded build file in order to build the target(s).

   In Gaudi terminology, an *action* is the equivalent of an Ant build file's *target* and *commands* (the equivalent of Ant's *tasks*) are the simple operations that are grouped together to perform an *action.* (Apache Software Foundation, No date).

   For example, a call to the GNU GCC compiler (Free Software Foundation, 2012) with the parameters required to compile a source code file into an executable program could be a *command* included in the *build* action to create a piece of software from source.

   This class also includes all the core *commands* supported by Gaudi, including the I/O commands to manipulate text files (supported by the *GaudiBase* class), to execute external programs, and to provide general output while executing build files amongst other related *commands.*

4. **GaudiForeman** class. In work parlance, the *foreman* is “a person in charge of a particular department, group of workers, et cetera”. (Dictionary.com LLC, 2012). In Gaudi, this analogy is applied to building and the GaudiForeman class is responsible for parsing the JSON build file and ascertaining the build *target*, *preamble* (block of variable definitions at the top of the build file) and *actions.* This class is written in Scala.

5. **GaudiHabitat** is another class with a single instance (singleton). The responsibility of this class is to get the operating system environment Gaudi is running within the Java virtual machine on, and determine how to ascertain paths for files, file extensions for executables, and so forth as appropriate. This class is written in Scala.

6. **GaudiGroovyPlugin** class serves to implement plug-ins written in the Groovy programming language (Glover, 2004) for Gaudi’s fledging plug-in system. Plug-ins written in Groovy can be used to create custom, be-spoke operations for (and thus extend) the Gaudi build tool, such as conversion functionality to transform a Makefile into a Gaudi JSON-based build file. As of the completion of this paper, the plug-in system in Gaudi has not been fully implemented and as such is disabled. For compatibility reasons, this class was written in Java rather than Scala. This class only contains a constructor method as nothing needs to be returned from the method that loads the plug-in code. (Leahy, No date).

7. **GaudiJythonPlugin** class. Similarly, this class partially implements the Gaudi plug-in system. However, as the name suggests, this implements plug-ins written in the Jython programming language. Jython is an implementation of Python to run on the Java virtual machine [JVM] (Pedroni & Rappin, 2002) and is included in Gaudi as it too is written for the JVM. Jython can be used to write plug-ins for Gaudi in the same way as Groovy. As with GaudiGroovyPlugin, also for compatibility reasons, this class was written in Java rather than Scala. This class also only contains a constructor method.

8. **GaudiPluginLoade**r class. In the Gaudi plug-in system, Gaudi plug-in code is stored in an ordinary **.zip* archive (PKZIP Inc., 2007), albeit with the file extension **.gpod*. The code is stored within this archive, in either a Groovy (file extension: **.groovy*) or Jython (**.py*) source file. This class is tasked with extracting the source file from the archive and determining the plug-in type from the source file extension (via *pattern matching* – Wampler & Payne, 2009). Determining on the code base, a *GaudiGroovyPlugin* or

GaudiJythonPlugin object is then invoked with the source code as a parameter. This class was written in Scala.

9. **GaudiPluginSupport** is a single instance class (singleton). This class only contains the Boolean constant Enabled, which by default is *true*. The configuration process for building Gaudi can set this to false. The only purpose of this class is to set whether or not the plug-in system will be used in the Gaudi build.

Interfaces

1. **IGaudiPlugin** interface. The purpose of an interface is to provide an abstraction of a class that needs to be modelled. In an interface, the method signatures are defined, but the implementation within them is not, but deliberately left blank. An interface is part of a block box approach where the programmer cares about what methods need to be available, but the implementation does not need to be known. As Oracle Corporation (1995a) states "In its most common form, an interface is a group of related methods with empty bodies". A bicycle's behaviour could be modelled to include methods to change gear, speed up, and apply breaks. When combined with say, the BMX (CITE) class, a contract could be formed to ensure that the implementation BMX implements the features in the interface common to all bicycles. In such a way, "implementing an interface allows a class to become formal about the behaviour it promises to provide". Interfaces are always *public* as they often are combined with an implementation class. (Oracle Corporation, 1995a).

   The IGaudiPlugin interface enforces the contract between *GaudiPlugin* and the method signatures defined in the interface itself of what a plugin should implement . This is as follows (shown with UML notation (Pilone, 2006).

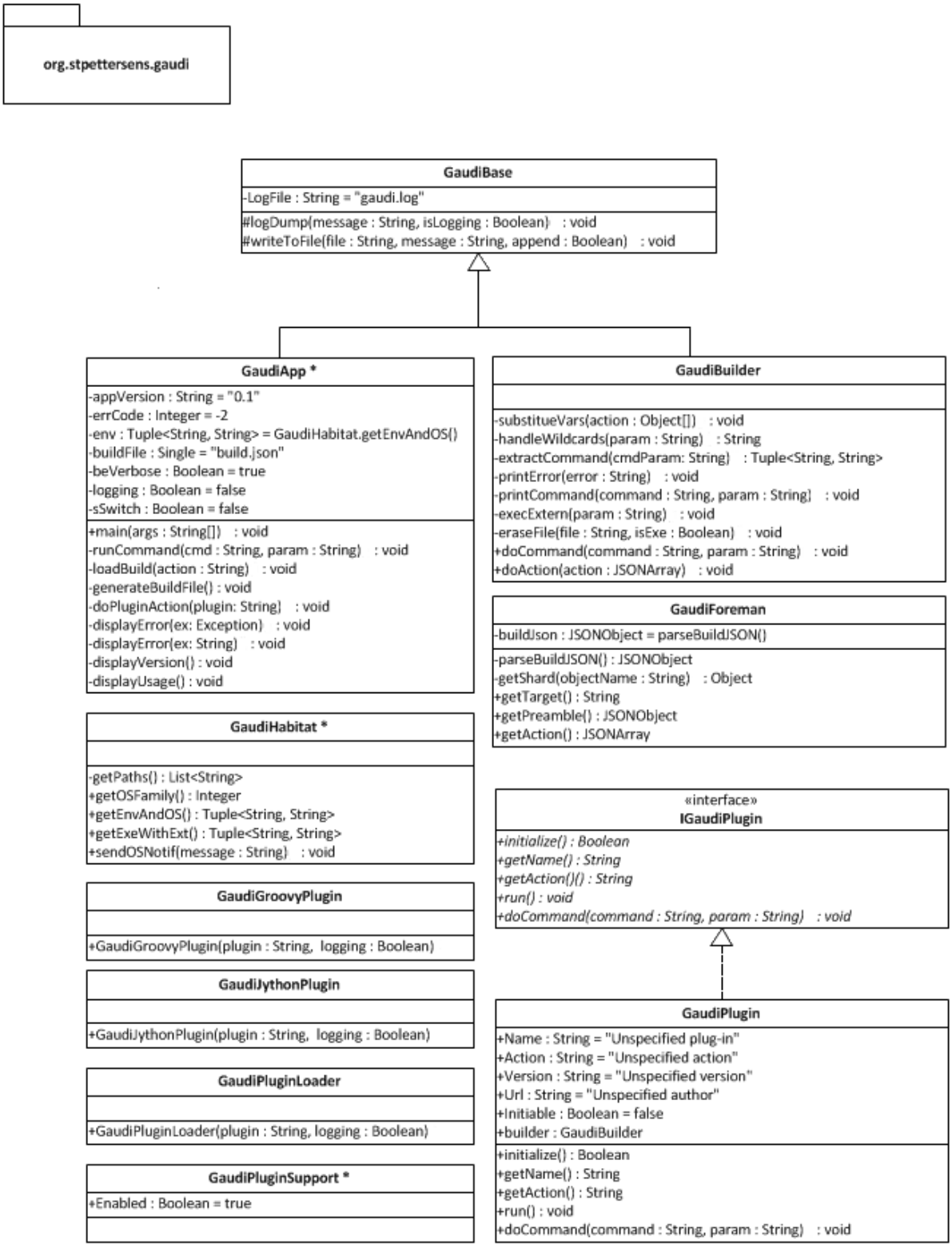
   - `+ initialize() : Boolean` [3] – for initializing the plug-in class in the plug-in system.
   - `+ getName() : String` – retrieve the name of the plugin.
   - `+ getAction() : String` –retrieve the *action* name performed by the plug-in.

[3] The Java type is *boolean* with a lowercase *b.* (Oracle Corporation, 1995b).

- `+ run(): void` – invoke the plug-in *action.*
- `+ doCommand(command : String, param : String) : void` - invoke a Gaudi command, e.g. `super.doCommand(":echo", "hello there!");`

With the *GaudiGroovyPlugin* class, the *GroovyClassLoader* parses a Groovy language class from a plug-in source code file onto a *Class* type, which is instigated as an *Object.* This *Object* is then cast to the *IGaudiPlugin* interface, which allows the Groovy class to become a Gaudi plug-in implementation without the explicit need to understand the underlying implementation of a plug-in in the Gaudi plug-in system. A UML diagram of all the classes follows on the next page.

Figure: UML class diagram for Gaudi application



Test deployment and survey

An Nullsoft Scriptable Install System [NSIS] (Nullsoft, 2011) setup application and **.zip* archive containing a preliminary build of the Gaudi application and a web link to some instructions on how to setup Gaudi and perform a short series of tasks (as in user interactions) to test it were deployed on the World Wide Web for my testing audience. The audience has a choice whether to download the setup application (recommended) or to install manually from the **.zip* archive.

Participants in the test deployment were invited to dowload Gaudi and the GNU C++ compiler [GCC]. They were instructed to follow the instructions to build a sample C++ Hello World program from source code using Gaud by invoking the *build.json* build file provided with the sample program source code from a **.zip* archive they were also invited to download and extract to their computer.

Additionally, a survey was deployed on the service offered by a web-based survey provider (SurveyMonkey, 2012) to collect responses on whether these tasks were straightforward for the testing user or not. Other responses collected were the testers' general computer literacy and development experience, to help offer context to the usability results. This survey was linked to from the wiki page providing links to the required testing files (Gaudi and the sample program, et cetera) and was completed by each participant after attempting the testing deployment tasks.

The testing instructions is included in the appendices (*see* Appendix VI) and the questions posed follow in the results of survey.

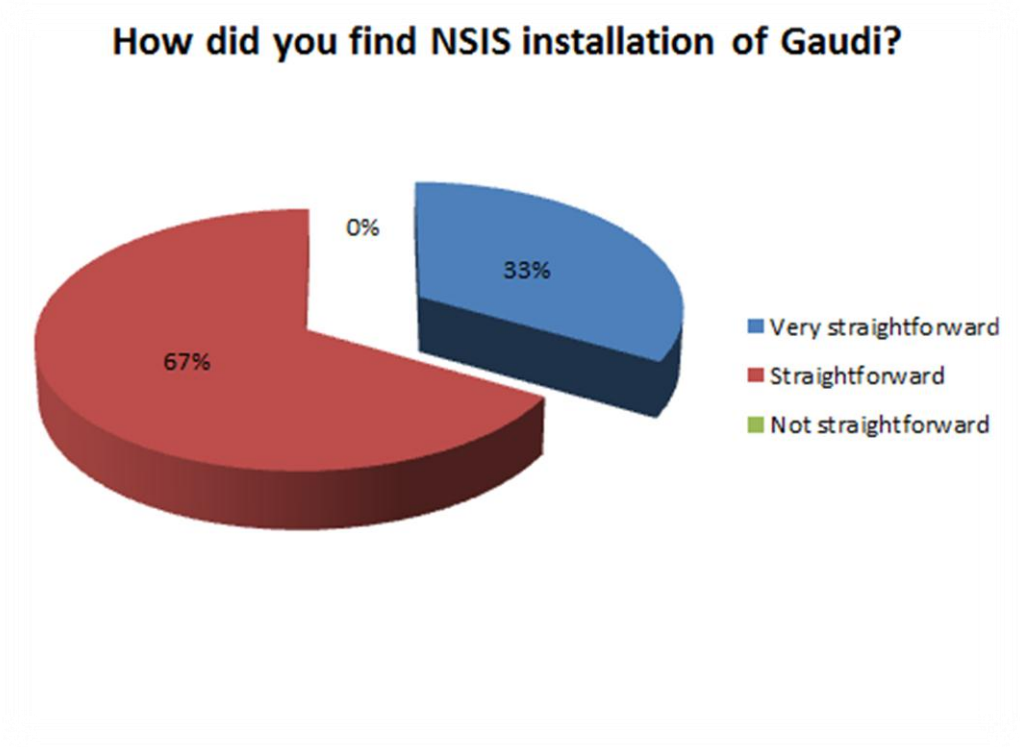
Continued overleaf.

Results of survey

The results of the survey follow and are presented with each question that the participant was asked.

Question 1. If you opted to use the installation program, how straightforward did you find installation of Gaudi? [One choice only answer].

The results were as follows:

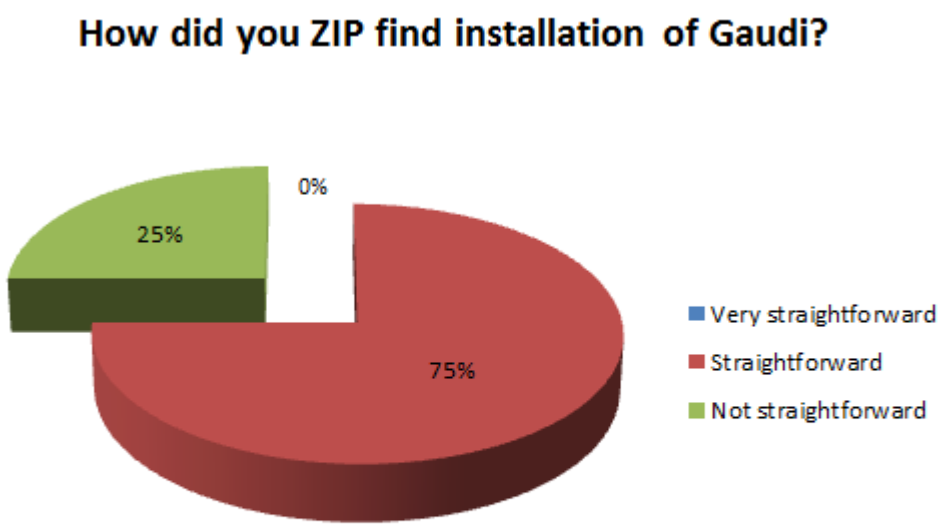


It was evident from the results that the majority of participants found the installation process "straightforward" at sixty-seven per cent. About a third (33%) found it "very straightforward". I was expecting mostly "straightforward" as the result, so this was very encouraging in terms of NSIS deployment of the application.

Continues overleaf.

*Question 2. If you installed the program manually from the *.zip archive, how straightforward did you find installation of Gaudi?* [One choice only answer].

The results were as follows:

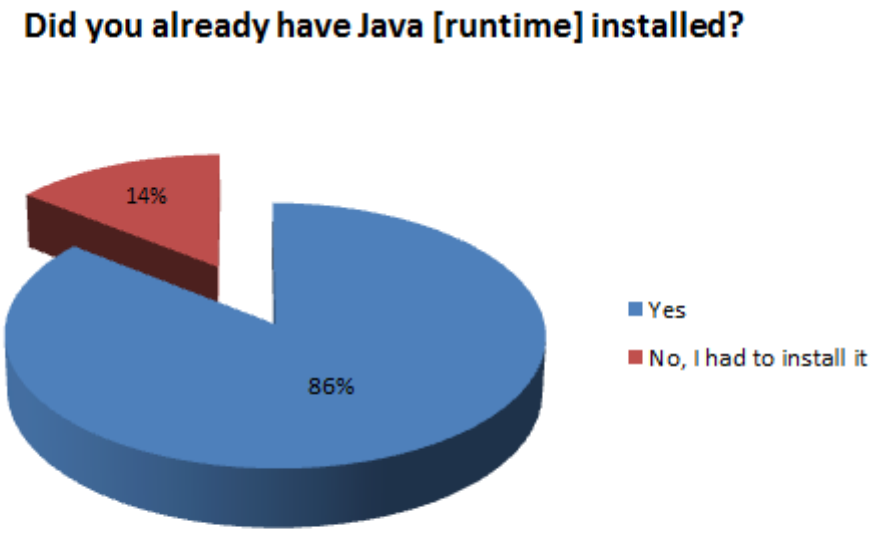


It was evident from the results that the majority of participants found the installation process “very straightforward” at three quarters (75%) A quarter (25%) found it “not straightforward”. No-one responded “very straightforward” (0%). I was expecting mostly “straightforward” as the result. It was surprising how the manual installation outshone the NSIS installation.

Continues overleaf.

Question 3. Did you already have Java (runtime environment) already installed on your system before installing Gaudi? [One choice only answer].

The results were as follows:

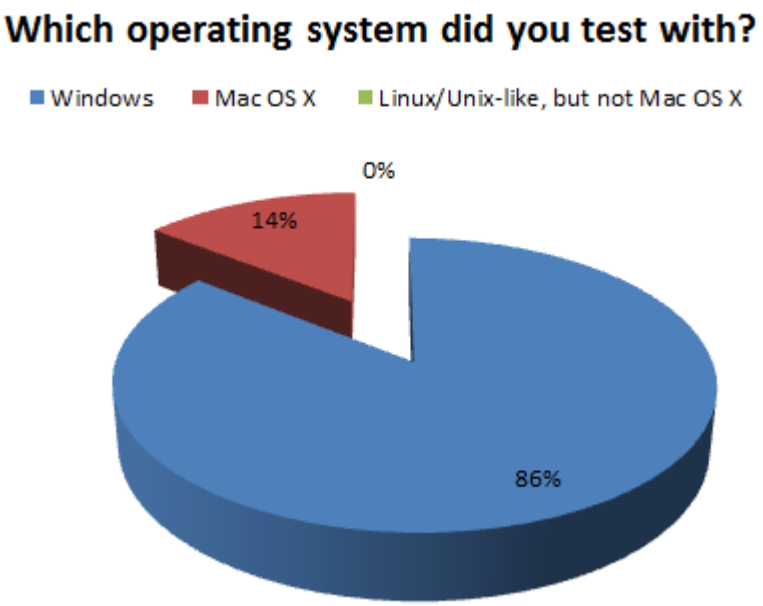


It was evident from the results that the overwhelming majority of participants already had the Java runtime environment installed on their system. ("Yes" at 86%). Fourteen per cent "had to install it first". Due to the ubiquitous nature of Java (Stewart, 2001), the results did not surprise me. Java's ubiquity was one of the, albeit lesser reasons, for the development of Gaudi to target Java.

Continues overleaf.

Question 4. What operating system did you try Gaudi on? [One choice only answer].

The results were as follows:

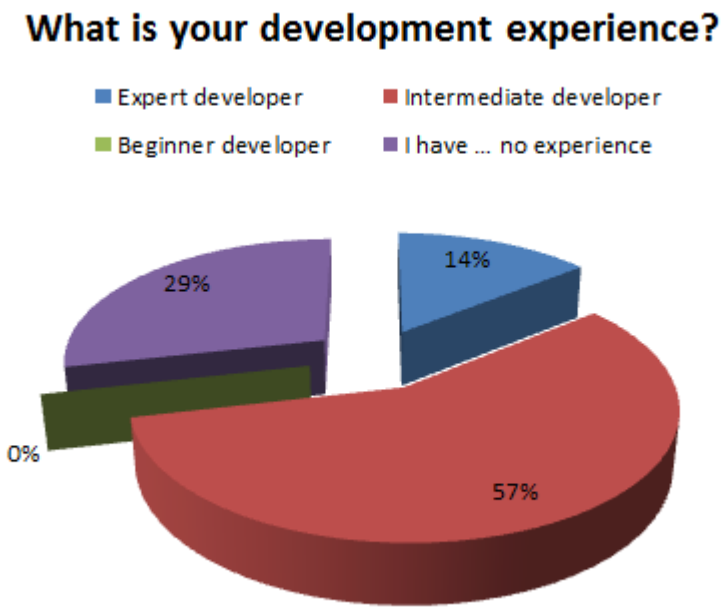


It was evident from the results that the overwhelming majority of participants were using Gaudi on the very popular Windows operating system at eighty-six per cent. Fourteen per cent used Mac OS X, another popular operating system. No-one asked used non-Mac OS-X UNIX-like systems (0%). Due to the prevalent nature of Windows (Refsnes Data, 2012), the results did not surprise me and were in line with my expectations.

Continues overleaf.

Question 5. How experienced are you when it comes to developing software? [One choice only answer].

The results were as follows:

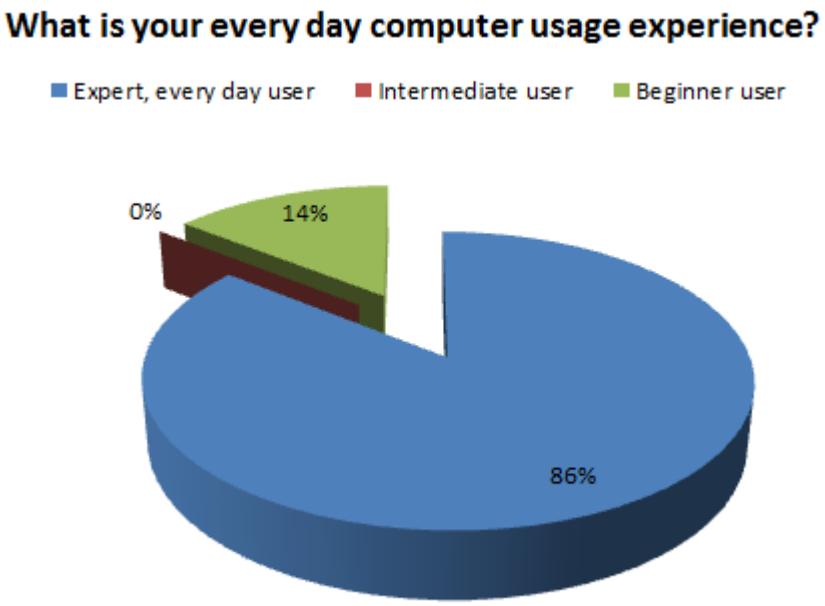


It was evident that the results to this question were more mixed. From the results, the majority of participants classified themselves as an “intermediate developer” at fifty-seven per cent. Next was those without development experience at twenty-nine per cent, then “expert developer” response was at fourteen per cent. Finally, “beginner developer” had no responses (0%). I did not have any expectations for the results of this question.

Continues overleaf.

Question 6. In terms of general computer use, such as word processing and Internet usage, how would you rate your level of experience? [One choice only answer].

The results were as follows:

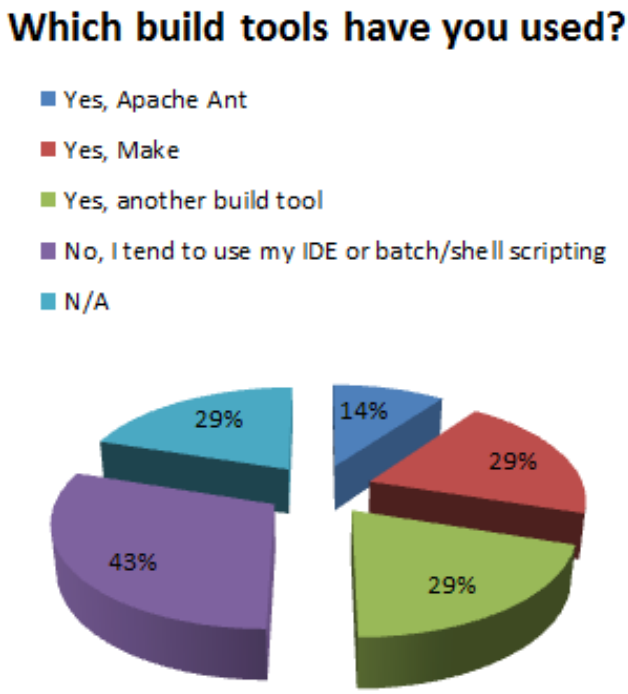


It was evident from the results that the vast majority of participants classified themselves as an "expert, every day user" at eighty-six per cent, next was "beginner user" at fourteen per cent. Intermediate users were at zero per cent. I did expect the majority of respondents to be expert, ever day users, since they could complete the survey online without issue; so these results generally matched my expectations.

Continues overleaf.

Question 7. If you have development experience, have you ever used a software build tool, before trying Gaudi, and if so which one(s)? Tick all that apply, if you previously said you don't have any dev. experience, just answer N/A. [Multiple choice answers].

The results were as follows:

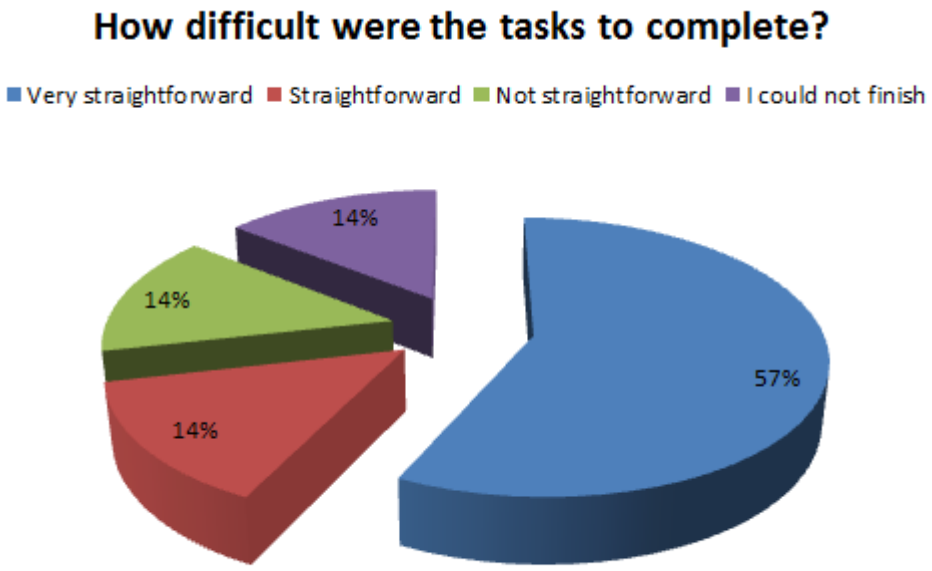


It was evident that the results to this question were more mixed as expected since this question allowed a respondent to select multiple choices. From the results that the majority of participants "tended to use [their] IDE or batch/shell scripting" at forty-three per cent. Joint next was Make, another build tool ant not applicable (N/A); all at twenty-nine per cent respectively. Last was Apache Ant at fourteen per cent. I expected IDE/batch/shell scripting to be the most popular answer and I also expected Make to be more popular than Ant. The results matched my predictions.

Continues overleaf.

Question 8. How straightforward did you find the tasks you were given to complete with Gaudi? [One choice only answer].

The results were as follows:

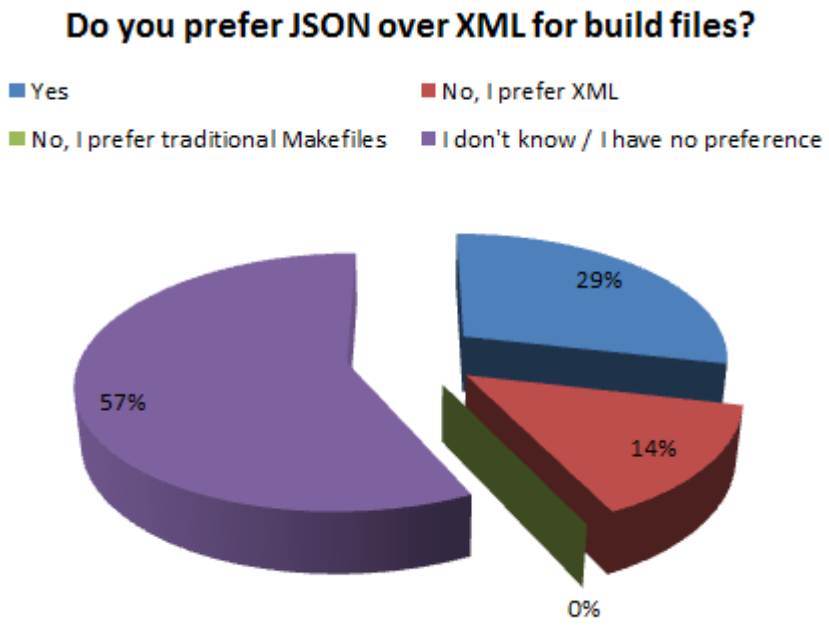


It was evident that the results to this question were more mixed. From the results, the majority of participants found the tasks "very straightforward" to complete. Joint next was "I could not finish", "Not straightforward" and "Straightforward"; all at fourteen per cent respectively. I expected "straightforward" to be the most popular response, so the 57% result for "very straightforward" exceeded my expectations.

Continues overleaf.

Question 9. Do you prefer JSON over XML or traditional Makefiles as a build file format? [One choice only answer].

The results were as follows:



It was evident that the results to this question were more mixed. From the results, the majority of participants had no preference or did not know at fifty-seven per cent. Next was a preference for JSON at twenty-nine per cent. Fourteen per cent expressed a preference for XML (the build file format of choice for Apache Ant) and no-body expressed a preference for traditional Make files a la the Make tool (0%). The overall majority of no preference matched my expectation, but I was happy that JSON proved more popular than XML overall, as it was the build file format I had chosen for Gaudi.

Continues overleaf.

Further internal testing

Using Gaudi, I compiled a similar small program written in Java and another Java program which consisted of several files and needed to be packaged. The build files and source files are included in Appendix VII.

Conclusions

In conclusion, Gaudi was my most ambitious programming project so far and originally started just as hobby project to help me learn the Scala programming language (Odersky et al., 2006). By the submission of this paper as my dissertation or individual study project, Gaudi has implemented or partially implement around twenty commands. Enough to build many programs, including almost Gaudi itself; although I have not yet written a build file for Gaudi to build itself.

From the results of my survey, I can also draw the conclusion that a build tool which uses JSON (Json.org, 2012) as it base format seems to be a popular concept.

As for how the results of implementing this project went, I feel that I have mainly achieved the aims that I set out at the beginning of the dissertation writing period.

I set out the aims I passed and failed below:

- ❖ FAILED. Although Gaudi can superficially theoretically build itself, it cannot yet configure itself without reliance on the *Txtrevise* utility.

- ✓ PASSED. Implemented enough core commands for Gaudi to be able to *configure* the build and Apache Ant in order to build Gaudi. This purpose is currently served by *configure* scripts.

- ❖ FAILED. Did not Implement server/client functionality for Gaudi to receive commands and relay responses via telnet or SSH to perform build operations or *actions*; each consisting of individual *commands.* Also, to implement the loading and execution of remote (to the program) build files.

- ❖ FAILED. Did Implement command extensibility. (E.g. bespoke commands).

- ✓ PARTIALLY PASSED. Implement plug-in system for advanced extensibility.

- ✓ PARTIALLY PASSED. Compatible with a wide range of JVM implementations. Tested with *HotSpot* 32 and 64-bit (*de facto* Oracle/Sun implementations) on Windows and Linux and Apple's Java implementation on Mac OS X. Also worked with *GIJ*, GNU Java virtual machine implementation on Windows and Linux.

References

Apache Commons (2012) *Commons IO Overview.* [Online]. Available from: http://commons.apache.org/io [Accessed 21st April 2012].

Apache Software Foundation (No date) *Using Apache Ant: Writing a Simple Buildfile.* [Online]. Available from: http://ant.apache.org/manual/using.html [Accessed 29th March 2012].

Apache Foundation (2011) *Apache Ant – Welcome.* [Online]. Available from: http://ant.apache.org [Accessed 3rd November 2011].

Avison, D. & Fitzgerald, G. (2006) *Information Systems Development: Methodologies, Techniques & Tools.* 4th edition. London, McGraw-Hill Education.

Begel, A. & Nagappan, N. (2007) *Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study.* [Online]. Microsoft Research. Available from: http://research.microsoft.com/en-us/um/people/abegel/papers/AgileDevatMS-ESEM07.pdf [Accessed 10th October 2011].

Bergin, J. (2000) *Multiple Inheritance in Java.* [Online]. Available from: http://csis.pace.edu/~bergin/patterns/multipleinheritance.html [Accessed 28th March 2012].

Boehm, B. (1988) *A Spiral Model of Software Development and Enhancement.* [Online]. Available from: http://www.dimap.ufrn.br/~jair/ES/artigos/SpiralModelBoehm.pdf [Accessed 10th October 2011].

Chapman, M. (2003) *Apache Ant 101: Make Java builds a snap.* [Online]. Available from: https://www6.software.ibm.com/developerworks/education/j-apant/j-apant-ltr.pdf [Accessed 10th October 2011].

Christias, P. (1994) *strip [man page].* [Online]. University of Edinburgh. Available from: http://unixhelp.ed.ac.uk/CGI/man-cgi?strip [Accessed 18th April 2012].

Craven, J. (No date) *Antoni Gaudí, Spanish Modernist Architect.* [Online]. Available from: http://architecture.about.com/od/architectsaz/p/gaudi.htm [Accessed 25th September 2011].

Dictionary, LLC. (2012) *Foreman [definition].* [Online]. Available from: http://dictionary.reference.com/browse/foreman [Accessed 20th April 2012].

Doar, M. (2005) *Practical Development Environments.* Sebastopol, O'Reilly Media.

Enberg, P., Grabiec, T., Huillet, A., Nossum, V. & Osman, K. (2011) *README [for Jato].* [Online]. Available from: https://github.com/penberg/jato/blob/master/README [Accessed 21st November 2011].

EPFL (École Polytechnique Fédérale de Lausanne) (2011) *Scala [programming language website].* [Online]. Available from: http://www.scala-lang.org [Accessed 26th September 2011].

Free Software Foundation (2010) *GNU Make.* [Online]. Available from: http://www.gnu.org/software/make/ [Accessed 10th November 2011].

Free Software Foundation (2012) *GCC, the GNU Compiler Collection.* [Online]. Available from: http://gcc.gnu.org [Accessed 29th March 2012].

Gagon, M. (1st June 2009) *AboutSCons.* [Online]. Available from: http://www.scons.org/wiki/AboutSCons [Accessed 10th October 2011 16:04:54 UTC].

Glover, A. (2004) *alt.lang.jre: Feeling Groovy.* [Online]. Available from: http://www.ibm.com/developerworks/java/library/j-alj08034/index.html [Accessed 20th April 2012].

Growl Team (2012) *Growl [website].* [Online]. Available from: http://growl.info [Accessed 21st April 2012].

Haas, J. (No date a) *Linux / Unix Command: man.* [Online]. Available from: http://linux.about.com/od/commands/l/blcmdl1_man.htm [Accessed 24th April 2012].

Haas, J. (No date b) *Linux / Unix Command: mv.* [Online]. Available from: http://linux.about.com/library/cmd/blcmdl1_mv.htm [Accessed 24th April 2012].

Hume Integrated Software (2011) *File Tcl Built-in Commands 8.3.* [Online]. Available from: http://www.hume.com/html84/mann/file.html [Accessed 7th December 2011].

Ippolito, G. (2011) *UNIX for DOS Users.* [Online]. Available from: http://www.yolinux.com/TUTORIALS/unix_for_dos_users.html [Accessed 7th December 2011].

Json.org (No date) *Introducing JSON.* [Online]. Available from: http://json.org [Accessed 3rd March 2012].

Kramer, D. (1996) *The Java Platform: A White Paper.* [Online]. Sun Microsystems. Available from: http://www.math.vu.nl/~eliens/online/@share/documents/java/white/JavaPlatform.pdf [Accessed 3rd October 2011].

Leahy, P. (No date) *The Constructor Method.* [Online]. Available from: http://java.about.com/od/workingwithobjects/a/constructor.htm [Accessed 21st April 2012].

Lindholm, T., Yellin, F., Bracha, G. & Buckley, A. (No date) *The Java Virtual Machine Specification: Java SE 7 Edition.* [Online]. Oracle Corporation. Available from: http://docs.oracle.com/javase/specs/jvms/se7/html/index.html [Accessed 3rd March 2012].

Livermore, J. (2008) Factors that Significantly Impact the Implementation of an Agile Software Methodology. *Journal of Software.* [Online]. 3 (4), 31-36. Available from: http://www.doaj.org/doaj?func=abstract&id=523964&recNo=4&toc=1&uiLanguage=en [Accessed 5th November 2011].

Microsoft (No date) *Xcopy [documentation page].* [Online]. Available from: http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/xcopy.mspx?mfr=true [Accessed 20th April 2012].

Microsoft (2012) *NMAKE [documentation on MSDN].* [Online]. Available from: http://msdn.microsoft.com/en-us/library/dd9y37ha.aspx [Accessed 20th April 2012].

Nullsoft (2011) *Nullsoft Scriptable Install System – Main Page.* [Online]. Available from: http://nsis.sourceforge.net/Main_Page [Accessed 21st April 2012]

Odersky, M., Altherr, P., Cremet, V., Dragos, I., Dubochet, G. Emir, B., McDirmid, S., Micheloud, S., Mihaylov, N., Schinz, M., Stenman, E., Spoon, L. & Zenger, M. (2006) *An Overview of the Scala Programming Language.* 2nd Ed. [Online]. École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. Available from: http://www.scala-lang.org/docu/files/ScalaOverview.pdf [Accessed 28th March 2012].

Odersky, M. (2012) *London Scala Users' Group – Scala: An Introduction.* [Online]. Available from: http://skillsmatter.com/podcast/scala/scala-intro [Accessed 22nd April 2012].

Oracle Corporation (1995a) *The Java Tutorials: What is an Interface?* [Online]. Available from: http://docs.oracle.com/javase/tutorial/java/concepts/interface.html [Accessed 23rd April 2012].

Oracle Corporation (1995b) *The Java Tutorials: Primitive Data Types.* [Online]. Available from: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html [Accessed 24th April 2012].

Oracle Corporation (1999) *JAR File Specification – Introduction.* [Online]. Available from: http://docs.oracle.com/javase/1.4.2/docs/guide/jar/jar.html#Intro [Accessed 26th April 2012].

Parekh, N. (2011) *The Waterfall Model Explained.* [Online]. Available from: http://www.buzzle.com/editorials/1-5-2005-63768.asp [Accessed 2nd January 2012].

Patel, V. (No date) *Makefile Tutorial.* [Online]. Available from: http://viralpatel.net/taj/tutorial/makefile-tutorial.php [Accessed 2nd May 2012].

Pedroni, S. & Rappin, N. (2002) *Jython Essentials.* Sebastopol, O'Reilly Media. [Online]. Available from: http://oreilly.com/catalog/jythoness/chapter/ch01.html [Accessed 20th April 2012].

Perl.org (2011) *Download Perl.* [Online]. Available from: http://www.perl.org/get.html#unix [Accessed 3rd November 2011].

Pilone, D. (2006) *UML 2.0 Pocket Reference.* Sebastopol, O'Reilly Media.

PKZIP Inc. (2007) *.ZIP File Format Specification.* [Online]. Available from: http://www.pkware.com/documents/casestudies/APPNOTE.TXT [Accessed 25th April 2012].

Python Software Foundation (2011a) *Using Python on Unix platforms.* [Online]. Available from: http://docs.python.org/using/unix.html [Accessed 3rd November 2011].

Python Software Foundation (2011b) *shutil – High-level file operations.* [Online]. Available from: http://docs.python.org/library/shutil.html [Accessed 7th December 2011].

Refsnes Data (2012) *OS Platform Statistics.* [Online]. Available from: http://www.w3schools.com/browsers/browsers_os.asp [Accessed 1st May 2012].

Russell, J. (2011) *Tutorial 5 – Inheritance & Polymorphism.* [Online]. Available from: http://home.cogeco.ca/~ve3ll/jatutor5.htm [Accessed 27th March 2012].

Saint-Pettersen, S. (2009) *Sams-Py: The personal code projects of Sam Saint-Pettersen.* [Online]. Available from: http://code.google.com/p/sams-py [Accessed 3rd November 2011].

Savarese, C. & Hart, B. (1999) *Cryptography – The Caesar Cipher.* [Online]. Available from: http://www.cs.trincoll.edu/~crypto/historical/caesar.html [Accessed 23rd April 2012].

Scacchi, W. (2001) *Process Models in Software Engineering.* [Online]. Available from: http://www.ics.uci.edu/~wscacchi/Papers/SE-Encyc/Process-Models-SE-Encyc.pdf [Accessed 2nd November 2011].

Schwaber, K. & Sutherland, J. (2011) *The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game.* [Online]. Available from: http://www.scrum.org/storage/scrumguides/Scrum_Guide.pdf [Accessed 2nd November 2011].

Serena Software, Inc. (2007) *An Introduction to Agile Software Development.* [Online]. Available from: http://www.serena.com/docs/repository/solutions/intro-to-agile-devel.pdf [Accessed 2nd November 2011].

Shin, S. (No date) Introduction to JSON (JavaScript Object Notation) *Presentation.* United States of America. [Online]. Available from: http://www.javapassion.com/ajax/JSON.pdf [Accessed 1st November 2011].

Stewart, B. (2001) *Java as a Teaching Language.* [Online]. Available from: http://www.oreillynet.com/pub/a/oreilly/java/news/teachjava_0101.html [Accessed 1st May 2012].

SurveyMonkey (2012) *SurveyMonkey [website].* [Online]. Available from: http://www.surveymonkey.com [Accessed 21st April 2012].

Szalvay, V. (2004) *An Introduction to Agile Software Development.* [Online]. Available from: http://www.danube.com/docs/Intro_to_Agile.pdf [Accessed 10th November 2011].

Tcl Developer Xchange (No date a) *Tcl Developer Xchange.* [Online]. Available from: http://www.tcl.tk [Accessed 7th December 2011].

Tcl Developer Xchange (No date b) *Tcl/Tk Software*. [Online]. Available from: http://www.tcl.tk/software/tcltk [Accessed 7th December 2011].

The Pennsylvania University (2008) *Lesson 3: Systems Analysis and Design Methodologies.* [Online]. Available from: http://www2.ds.psu.edu/AcademicAffairs/Classes/IST260W/topic01/topic_0110_04.html [Accessed 1st November 2011].

Thomson Reuters (2009) *Web of Science: Wildcards.* [Online]. Available from: http://images.webofknowledge.com/WOK45/help/WOS/ht_wildcd.html [Accessed 30th April 2012].

Tuffs, S. (2004) *Simplify your application delivery with One-JAR.* [Online]. Available from: http://www.ibm.com/developerworks/java/library/j-onejar [Accessed 26th April 2012].

Tutorials Point (No date) *Scala – Access Modifiers.* [Online]. Available from: http://www.tutorialspoint.com/scala/scala_access_modifiers.htm [Accessed 29th March 2012].

University of Michigan (1997) *The Java Programming Language.* [Online]. Available from: http://groups.engin.umd.umich.edu/CIS/course.des/cis400/java/java.html [Accessed 26th September 2011].

Veselosky, V. (2011) *Compiling Linux Software from Source Code.* [Online]. Available from: http://www.control-escape.com/linux/lx-swinstall-tar.html [Accessed 15th November 2011].

Vogel, L. (2011) *Apache Ant – Tutorial.* [Online]. Available from: http://www.vogella.de/articles/ApacheAnt/article.html [Accessed 15th November 2011].

Waikato Linux Users Group (2005). *Ant Vs. Make.* [Online]. Available from: http://wlug.org.nz/AntVsMake [Accessed 7th December 2011].

Wampler, D. & Payne, A. (2009) *Programming Scala.* Sebastopol, O'Reilly Media.

Waters, K. (2007) *Scrum Agile Development: Uncommon Sense.* [Online]. Available from: http://www.allaboutagile.com/scrum-agile-development-uncommon-sense/ [Accessed 15th October 2011].

Waters, K. (2011) *Agile Methodologies.* [Online]. Available from: http://www.allaboutagile.com/agile-methodologies/ [Accessed 15th October 2011].

Bibliography

Hughes, D. (2002) Spiral Model of Software Development at JPL. *Presented to SBIRS Program Office.* United States of America. [Online]. Available from: http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/11132/1/02-3089.pdf [Accessed 2nd November 2011].

Microsoft (2012) *Insert or create footnotes and end notes*. [Online]. Available from: http://office.microsoft.com/en-us/word-help/insert-or-create-footnotes-and-endnotes-HA101854833.aspx#_Toc293388391 [Accessed 20th April 2012].

Nirosh (2011) *Introduction to Object Oriented Programming Concepts (OOP) and more.* [Online]. Available from: http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep#difference [Accessed 23rd April 2012].

Serrano, N. & Ciordia, I. (2004) Ant: automating the process of building applications. *Software, IEEE.* [Online]. 21 (6), 89-91. Available from: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1353230 [Accessed 11th October 2011].

University of California at Berkeley. (No date) How to write an abstract: Links and Tips. [Online]. Available from: http://research.berkeley.edu/ucday/abstract.html [Accessed 1st May 2012].

Vogel, L. (2011) *What is JSON?* [Online]. Available from: http://www.vogella.de/articles/JSON/article.html [Accessed 15th November 2011].

*

Appendices

Appendix I: Credits of third-party libraries used by Gaudi application.

```
JSON.simple (simple Java toolkit for JSON)
Copyright 2010 Yidong Fang et al.
http://code.google.com/p/json-simple

Apache Commons IO library
Copyright 2002-2011 The Apache Software Foundation.
http://commons.apache.org/io

Scala library
Copyright 2002-2012 EPFL, Lausanne, unless otherwise specified.
All rights reserved.
Licensed under a BSD-style license - http://www.scala-lang.org/node/146
http://www.scala-lang.org

Groovy library
Copyright 2003-2010 The respective authors and developers.
This product includes software developed by The Groovy community.
http://groovy.codehaus.org

Jython (Minimal core library)
Copyright (c) 2000-2007 Jython Developers All rights reserved.
Licensed under the Python Software Foundation License
```

```
and Jython 2.0, 2.1 License - http://www.jython.org/license.html
http://www.jython.org
```

Appendix II: Gaudi application source code (Scala, Java)

GaudiApp.scala (181 SLOC, includes comments):

```
/*
Gaudi platform agnostic build tool
Copyright 2010-2012 Sam Saint-Pettersen.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

For dependencies, please see LICENSE file.
*/
package org.stpettersens.gaudi
import scala.io.Source
import scala.util.matching.Regex
import java.io.IOException
```

```
import org.json.simple.{JSONObject,JSONArray}
//

object GaudiApp extends GaudiBase {

  //-------------------------------------------------------------
  val appVersion: String = "0.1"
  val errCode: Int = -2;
  val env: (String, String) = GaudiHabitat.getEnvAndOS()
  //-------------------------------------------------------------
  var buildFile: String = "build.json" // Default build file
  var beVerbose: Boolean = true // Gaudi is verbose by default
  var logging: Boolean = false // Logging is disabled by default
  var messenger = new GaudiMessenger(logging)
  var sSwitch: Boolean = false

  def main(args: Array[String]): Unit = {
      var pSwitch: Boolean = false
        var fSwitch: Boolean = false
        var action: String = "build"
        val pluginPattn: Regex = """([\w\:\\//]+.gpod)""".r
        val filePattn: Regex = """([\w\:\\//]+.json)""".r
        val actPattn: Regex = """([a-z]+)""".r
        val cmdPattn: Regex = """:([a-z]+)\s{1}([\\\/A-Za-z0-9\s\.\*\+\_\-
\>\!\,]+)""".r
        var cmd: String = null
        var param: String = null
```

```
        /* Attempt Gnome GTK integration when operating system is Linux */
        /*
        if(GaudiHabitat.getOSFamily() == 1)
        {
              Gtk.init(args)
        }
        */

        /* Default behavior is to build project following
        build file in the current directory */
        if(args.length == 0) loadBuild(action)

        // Handle command line arguments
        else if(args.length > 0 && args.length < 7) {
              for(arg <- args) {
                    arg match {
                          case "-i" => displayUsage(0)
                          case "-v" => displayVersion()
                          case "-l" => logging = true
                          case "-s" => sSwitch = true
                          case "-b" => generateBuildFile()
                          case "-p" => {
                                if(GaudiPluginSupport.Enabled) pSwitch =
true
                          }
                          case "-q" => beVerbose = false
                          case "-f" => fSwitch = true
                          case pluginPattn(p) => {
```

```
                        if(pSwitch) doPluginAction(arg)
                    }
                    case filePattn(f) => {
                        if(fSwitch) buildFile = arg
                    }
                    case actPattn(a) => action = a
                    case cmdPattn(c, p) => {
                        cmd = c
                        param = p
                    }
                    case _ => {
                        displayError(String.format("Argument (%s
is invalid)", arg))
                    }
                }
            }
            if(sSwitch) {
                messenger.start()
            }
            if(cmd != null) {
                runCommand(cmd, param)
            }
            else loadBuild(action)
        }
        else displayError("Arguments (requires 0-6 arguments)")
    }
    // Just peform a stdin command; really just for testing implemented
commands.
    // E.g. argument ":move a->b"
```

```
  private def runCommand(cmd: String, param: String): Unit = {
        // Create a new builder to run a command
        val builder = new GaudiBuilder(null, sSwitch, beVerbose, logging)
        builder.doCommand(cmd, param)
        System.exit(0)
  }
  // Load and delegate parse and execution of build file
  private def loadBuild(action: String): Unit = {
        var buildConf: String = ""
        try {
              for(line <- Source.fromFile(buildFile).getLines()) {
                    buildConf += line
              }
              // Shrink string, by replacing tabs with spaces;
              // Gaudi build files should be written using tabs
              buildConf = buildConf.replaceAll("\t","")
        }
        catch { // Catch I/O and general exceptions
              case ioe: IOException => displayError(ioe)
              case e: Exception => displayError(e)
        }
        finally {
              // Delegate to the foreman and builder
              val foreman = new GaudiForeman(buildConf)
              val builder = new GaudiBuilder(foreman.getPreamble, sSwitch,
beVerbose, logging)
              if(beVerbose) {
                    println(String.format("[ %s => %s ]",
foreman.getTarget, action))
```

```
            }
            builder.doAction(foreman.getAction(action))
            messenger.end()
      }
}
// Generate a Gaudi build file (build.json)
private def generateBuildFile(): Unit = {
      // TODO
}
// Handle a plug-in
private def doPluginAction(plugin: String): Unit = {
      //new GaudiPluginLoader(plugin, logging)
      System.exit(0)
}
// Display an error
def displayError(ex: Exception): Unit = {
      println(String.format("\nError with: %s.", ex.getMessage))
      logDump(ex.getMessage, logging)
      displayUsage(errCode)
}
// Overloaded for String parameter
def displayError(ex: String): Unit = {
      println(String.format("\nError with: %s.", ex))
      logDump(ex, logging)
      displayUsage(errCode)
}
// Display version information and exit
private def displayVersion(): Unit = {
```

```
        println(String.format("Gaudi v.%s [%s (%s)]", appVersion, env._1,
env._2))

        if(GaudiPluginSupport.Enabled) println("Plug-in support.")

        else println("No plug-in support (-noplugins).")

        System.exit(0)

  }

  // Display usage information and exit

  private def displayUsage(exitCode: Int): Unit = {

        println("\nGaudi platform agnostic build tool")

        println("Copyright (c) 2010-2012 Sam Saint-Pettersen")

        println("\nReleased under the Apache License v2.")

        println("\nUsage: gaudi [-s <port>][-l][-i|-v|-n|-m][-q]")

        println("[-p <plug-in>][-f <build
file>][<action>|\"<:command>\"]")

        println("\n-s: Enable listen on socket (Default: TCP/3082).")

        println("-l: Enable logging of certain events.")

        println("-i: Display usage information and quit.")

        println("-v: Display version information and quit.")

        println("-b: Generate a Gaudi build file (build.json).")

        println("-p: Invoke <plug-in> action.")

        println("-q: Mute console output, except for :echo and errors
(Quiet mode).")

        println("-f: Use <build file> instead of build.json.")

        System.exit(exitCode)

  }

}
```

GaudiBase.scala (85 SLOC):

```
package org.stpettersens.gaudi
```

```
import java.util.Calendar

import java.text.{DateFormat,SimpleDateFormat}

import java.io.{PrintWriter,FileOutputStream,IOException}


class GaudiBase {


      val caesarOffset: Array[Int] = Array(0x22, 0x1C, 0xA, 0x1F, 0x8,
0x2, 0x24, 0xF, 0x13, 0x25, 0x40, 0x10)

      val LogFile: String = "gaudi.log"


      protected def logDump(message: String, isLogging: Boolean): Unit = {

            val calendar: Calendar = Calendar.getInstance()

            val sdf: SimpleDateFormat = new SimpleDateFormat("MM-dd-yyyy
HH:mm:ss")

            val timeStamp: String = sdf.format(calendar.getTime())

            if(isLogging) {

                  writeToFile(LogFile, String.format("[%s]\r\n%s",
timeStamp, message), true)

            }

      }


      // File writing operations

      protected def writeToFile(file: String, message: String, append:
Boolean): Unit = {

            var out: PrintWriter = null

            try {

                  out = new PrintWriter(new FileOutputStream(file,
append))

                  out.println(message)

            }
```

```
            catch {
                  case ioe: IOException =>  println(ioe.getMessage) // !
//printError(ioe.getMessage)
            }
            finally {
                  out.close()
            }
      }

      // Encode a message in a simple Caesar cipher
      protected def encodeText(message: String) : String = {

            var emessage: String = "";
            var x: Int = 0;
            for(s <- message) {

                  if(x == caesarOffset.length - 1) x = 0;
                  var cv: Int = message.charAt(x).asInstanceOf[Int] -
caesarOffset(x);
                  var ca: Char = cv.asInstanceOf[Char];
                  emessage += ca
                  x += 1
            }
            emessage
      }


      // Decode a message in a simpole Caesar cipher
      protected def decodeText(message: String) : String = {
```

```
            var umessage: String = "";
            var x: Int = 0;
            for(s <- message) {

                  if(x == caesarOffset.length - 1) x = 0;
                  var cv: Int = message.charAt(x).asInstanceOf[Int] +
caesarOffset(x);
                  var ca: Char = cv.asInstanceOf[Char];
                  umessage += ca
                  x += 1
            }
            umessage
      }
}
```

GaudiBuilder.scala (203 SLOC):

```
package org.stpettersens.gaudi
import org.json.simple.{JSONObject,JSONArray}
import org.apache.commons.io.FileUtils._
import org.apache.commons.io.filefilter.WildcardFileFilter
import scala.util.matching.Regex
import java.io._

class GaudiBuilder(preamble: JSONObject, sSwitch: Boolean,
beVerbose: Boolean, logging: Boolean) extends GaudiBase {

      // Define global messenger object
      var messenger = new GaudiMessenger(logging)
```

```
      if(sSwitch) {
            messenger.start()
      }

      // Substitute variables for values
      private def substituteVars(action: Array[Object]): Unit = {
            println(preamble)
      }
      // Handle wild cards in parameters such as *.scala, *.cpp,
      // for example, to compile all Scala or C++ files in the specified
dir
      private def handleWildcards(param: String): String = {
            if(param.contains("*")) {
                val rawParamPattn: Regex = """[\w\d]*\s*(.*)""".r
                var rawParamPattn(raw_param) = param
                  var dir = new File(".")
                  val filePattn: Regex = """\*([\.\w\d]+)""".r
                  var filePattn(ext) = raw_param
                  val filter: FileFilter = new WildcardFileFilter("*" +
ext)
                  var newParam: String = ""
                  val files: Array[File] = dir.listFiles(filter)
                  for(file <- files) {
                        newParam += file
                  }
                  return newParam.replace(".\\", " ")
            }
            param
```

```
	}
	// Extract command and param for execution
	private def extractCommand(cmdParam: String): (String, String) = {
		val cpPattn: Regex =
"""\{\"(\w+)\"\:\"([\\\/\"\w\d\s\$\.\*\,\_\+\-\>\_]+)\"\}""".r
		var cpPattn(cmd: String, param: String) = cmdParam
		(cmd, param)
	}
	// Print an error related to action or command and exit
	private def printError(error: String): Unit = {
		println(String.format("\tAborting: %s.", error))
		logDump(error, logging) // Also log it
		System.exit(-2) // Exit application with error code
	}
	// Print executed command
	private def printCommand(command: String, param: String): Unit = {
		if(beVerbose && command != "echo") {
			println(String.format("\t:%s %s", command, param))
		}
	}
	// Execute an external program or process
	private def execExtern(param: String): Unit = {
		val exe: (String, String) = GaudiHabitat.getExeWithExt(param)
		// ------------------------------------------------------------
------
		logDump(String.format("Executed -> %s %s\n" +
		"Wildcard matched -> %s", exe._1, exe._2,
handleWildcards(exe._2)),
		logging)
```

```
            // ------------------------------------------------------------
------

            if(exe._1 != null) {

                  var p: Process =
Runtime.getRuntime().exec(String.format("%s %s", exe._1,
handleWildcards(exe._2)))

                  val reader = new BufferedReader(new
InputStreamReader(p.getErrorStream()))

                  var line: String = reader.readLine()

                  if(line != null) {

                        val msg = String.format("\t~ %s", line)

                        println(msg)

                        messenger.report(msg)

                  }

            }

      }

      private def eraseFile(file: String, isExe: Boolean): Unit = {

            if(isExe && GaudiHabitat.getOSFamily() == 0) {

                  new File(file.concat(".exe")).delete()

            }

            else new File(file).delete()

      }

      // Execute a command in the action

      def doCommand(command: String, param: String): Unit = {

            // Handle any potential wildcards in parameters

            // for all commands other than :exec and :echo

          val wcc_param: String = handleWildcards(param)


            // Do not print "echo" commands, but do print others

            printCommand(command, param)
```

```
            command match {
                  case "exec" => {
                        execExtern(param)
                  }
                  case "mkdir" => {
                        val aDir: Boolean = new File(param).mkdir()
                        if(!aDir) {
                              printError(String.format("Problem making dir
-> %s", param))
                        }
                  }
                  case "list" => println(String.format("\t-> %s",
wcc_param))
                  case "echo" => println(String.format("\t# %s", param))
                  case "erase" => eraseFile(wcc_param, false) // Support
wildcards
                  case "erasex" => eraseFile(wcc_param, true)
                  case "xstrip" => {
                        var p: String = wcc_param;
                        if(GaudiHabitat.getOSFamily() == 0) {
                              p = wcc_param.concat(".exe")
                        }
                        execExtern(String.format("strip %s", p)) // Relies
on strip command.
                  }
                  case "copy" => {
                        // Explicit the first time about this being a
string array
                        val srcDest: Array[String] = param.split("->")
```

```
                        copyFile(new File(srcDest(0)), new
File(srcDest(1))) // Via Apache Commons IO
                  }
                  case "rcopy" => {
                        // Implicit...
                        "Recur. copy" // TODO
                  }
                  case "move" => {
                        // Et cetera...
                        val srcDest = param.split("->")
                        moveFile(new File(srcDest(0)), new
File(srcDest(1))) // Via Apache Commons IO
                  }
                  // Append message to a file
                  // Usage in buildfile: { "append": file>>message" }
                  // Equivalent to *nix's echo "message" >> file
                  case "append" => {
                        val fileMsg = param.split(">>")
                        writeToFile(fileMsg(0), fileMsg(1), true)
                  }
                  case "clobber" => {
                        val fileMsg = param.split(">>")
                        writeToFile(fileMsg(0), fileMsg(1), false)
                  }
                  case "notify" => {
                        GaudiHabitat.sendOSNotif("null")
                  }

                  case "encode" => {
```

```
                        val fileMsg = param.split(">>")
                        writeToFile(fileMsg(0), encodeText(fileMsg(1)),
true)
                  }

                  case "decode" => {

                        val fileMsg = param.split("<<")

                        writeToFile(fileMsg(0), decodeText(fileMsg(1)),
true)
                  }

                  case "help" => {

                        val onCmd = param.split(" ")
                        // TODO!
                        //
                  }
                  case _ => {
                        // Implement extendable commands
                        printError(String.format("%s is an invalid
command", command))
                  }
            }
      }

      // Execute an action
```

```
	def doAction(action: JSONArray): Unit = {
		try {
			val actionArray = action.toArray()
			//substituteVars(actionArray)
			for(cmdParam <- actionArray) {
				val cpPair = extractCommand(cmdParam.toString)
				doCommand(cpPair._1, cpPair._2)
			}
		}
		catch {
			case ex: Exception => {
				println(String.format("\t[%s]", ex.getMessage))
				printError("Encounted an invalid action or
command")
			}
		}
	}
}
```

GaudiForeman.scala (60 SLOC):

```
package org.stpettersens.gaudi
import org.json.simple.{JSONValue,JSONObject,JSONArray}
import scala.util.matching.Regex

class GaudiForeman(buildConf: String) {

	// Parse build config into build JSON object on initialization
	val buildJson: JSONObject = parseBuildJSON()
```

```
	private def parseBuildJSON(): JSONObject = {
		val buildObj: Object = JSONValue.parse(buildConf)
		buildObj.asInstanceOf[JSONObject]
	}
	// Get sub-object 'shard' from build JSON object
	private def getShard(objectName: String): Object = {
		var shardStr: String = "n"
		try {
			val objPattn = new Regex(objectName)
			shardStr =
JSONValue.toJSONString(buildJson.get(objectName))
		}
		catch {
			case ex: Exception => {
				GaudiApp.displayError("Instructions (Badly
formatted JSON)")
			}
			// TODO: PREVENT NULL POINTER EXCEPTIONS FROM OCCURING
		}
		JSONValue.parse(shardStr)
	}
	// Get target from parsed preamble
	def getTarget(): String  = {
		val targetStr =
JSONValue.toJSONString(getPreamble().get("target"))
		targetStr.replaceAll("\"", "")
	}
	// Get the preamble from build object
	def getPreamble(): JSONObject = {
```

```
            getShard("preamble").asInstanceOf[JSONObject]
      }
      // Get an execution action
      def getAction(action: String): JSONArray = {
            getShard(action).asInstanceOf[JSONArray]
      }
}
```

GaudiHabitat.scala (101 SLOC):

```
package org.stpettersens.gaudi
import java.io.File
import scala.util.matching.Regex
import scala.collection.mutable.ListBuffer
import scala.collection.immutable.List

object GaudiHabitat {

      private def getPaths(): List[String] = {
            val pathVar: String = System.getenv("PATH")
            val winPathPattn: Regex = """([\:\w\d\s\.\-\_\\]+)""".r
            val nixPathPattn: Regex = """([\w\d\s\.\-\_\/]+)""".r
            var pathPattn: Regex = null

            if(getOSFamily() == 0) {
                  pathPattn = winPathPattn
            }
            else if(getOSFamily() == 1) {
```

```
            pathPattn = nixPathPattn
      }
      val paths = new ListBuffer[String]()
      for(p <- pathPattn.findAllIn(pathVar)) {
            paths += p
      }
      paths.toList
}
def getOSFamily(): Int = {
      var osFamily: Int = -1
      val env: (String, String) = getEnvAndOS()
      val osFamilyPattn: Regex = """(\w+)[\s\d\w]*""".r
      var osFamilyPattn(osName) = env._2
      osName match {
            case "Windows" => osFamily = 0
            case "Linux" => osFamily = 1
            case "Macintosh" => osFamily = 2
      }
      osFamily
}
def getEnvAndOS(): (String, String) = {
      (
      String.format(
      "%s %s",
      System.getProperty("java.vm.name"),
      System.getProperty("java.version")
      ),
      System.getProperty("os.name")
```

```
      )
}
def getExeWithExt(command: String): (String, String) = {
      var pathTerm: String = null
      if(getOSFamily() == 0) {
            pathTerm ="\\"
      }
      else if(getOSFamily() == 1 || getOSFamily() == 2) {
            pathTerm = "/"
      }
      val exts = new Array[String](5)
      exts(0) = ".exe"
      exts(1) = ".bat"
      exts(2) = ".cmd"
      exts(3) = ".sh"
      exts(4) = ""
      val exePattn: Regex = """([\w\d\+\-\_]+)\s*(.*)""".r
      var exePattn(exe, param) = command
      var ext: String = null
      val paths: List[String] = getPaths()
      for(path <- paths) {
            for(ext <- exts) {
                  if(new File(path+pathTerm+exe+ext).exists()
                  && new File(path+pathTerm+exe+ext).isFile()) {
                        return (path+pathTerm+exe+ext, param)
                  }
            }
      }
```

```
            (null, null)
      }

      def sendOSNotif(message: String): Unit = {

            val base = new GaudiBase()
            // TO IMPLEMENT...
      }
}
```

GaudiGroovyPlugin.java (61 SLOC):

```
package org.stpettersens.gaudi;
import groovy.lang.GroovyClassLoader;
import java.io.*;

// NOTE: The GaudiGroovyPlugin implementation code is written in
// Java rather than Scala for compatibility reasons.(Specifically casting
issues).
public class GaudiGroovyPlugin {

      @SuppressWarnings("unchecked")
      GaudiGroovyPlugin(String plugin, boolean logging) throws Exception {

            //GaudiLogger logger = new GaudiLogger(logging);

            try {
                  GroovyClassLoader gcl = new GroovyClassLoader();
                  Class pluginClass = gcl.parseClass(new File(plugin));
```

```
                Object aPlugin = pluginClass.newInstance();
                IGaudiPlugin gPlugin = (IGaudiPlugin) aPlugin;
                Object init = gPlugin.initialize();
                Boolean bInit = (Boolean) init; // Cast init value to
Boolean *object*


                if(bInit) {
                      // On successful initialization, display feedback
and run the plug-in
                      String name = gPlugin.getName().toString();
                      String action = gPlugin.getAction().toString();
                      System.out.println("Initialized Groovy-based plug-
in.\n");
                      //logger.dump(String.format("Initialized Groovy-
based plug-in -> %s.", name));
                      System.out.println(String.format("[ %s => %s ]",
name, action));
                      gPlugin.run();


                }
                else {
                      // Otherwise, display the standard did not load
feedback
                      System.out.println("Error with: Groovy-based Plug-
in (Failed to load).");
                      //logger.dump("Failed to load plug-in.");
                }
            }
            catch(Exception e) {
                System.out.println("\n\tError in plug-in code:\n\t\t" +
e);
```

```
                  //logger.dump("Error in plug-in code.");
            }
      }
}
```

GaudiJythonPlugin.java (59 SLOC):

```
package org.stpettersens.gaudi;
import org.python.core.*;
import org.python.util.*;

//NOTE: The GaudiJythonPlugin implementation code is written in
//Java rather than Scala for consistency with GaudiGroovyPlugin.
public class GaudiJythonPlugin {

      GaudiJythonPlugin(String plugin, boolean logging) {

            GaudiLogger logger = new GaudiLogger(logging);

            try {
                  PySystemState.initialize();
                  PythonInterpreter jyIntp = new PythonInterpreter();
                  jyIntp.execfile(plugin);
                  PyObject pluginObj = jyIntp.get("plugin");
                  PyObject init = pluginObj.__getattr__("Initable");

                  if(init.toString() == "True") {
                        // On successful initialization, display feedback
and run the plug-in
```

```
                        String name =
pluginObj.__getattr__("Name").toString();

                        String action =
pluginObj.__getattr__("Action").toString();

                        System.out.println("Initialized Jython-based plug-
in.\n");

                        logger.dump(String.format("Initialized Jython-
based plug-in -> %s.", name));

                        System.out.println(String.format("[ %s => %s ]",
name, action));

                        PyObject runMethod = pluginObj.__getattr__("run");

                        runMethod.__call__();

                  }

                  else {

                        // Otherwise, display the standard did not load
feedback

                        System.out.println("Error with: Jython-based Plug-
in (Failed to load).");

                        logger.dump("Failed to load plug-in.");

                  }

            }

            catch(Exception e) {

                  System.out.println("\n\tError in plug-in code:\n\t\t" +
e);

                  logger.dump("Error in plug-in code.");

            }

      }

}
```

GaudiPluginLoader.scala (41 SLOC):

```
package org.stpettersens.gaudi
```

```
import java.io.File;
import scala.util.matching.Regex

class GaudiPluginLoader(plugin: String, logging: Boolean) {

	// First of all, extract plugin code from the plugin archive (Zip
File)
	//val unpacker = new GaudiPacker(plugin, logging) /* Redundent code
*/
	//val codeFile = unpacker.extrZipFile()
	// NEED TO REDO EXTRACTION CODE.

	// Second step is to load pass the code onto the right script
handler by
	// file extension - *.groovy is Groovy code; *.py is Jython code.
	val langPattnGvy: Regex = """([\w\:\\//]+.groovy)""".r
	val langPattnJyt: Regex = """([\w\:\\//]+.py)""".r

	codeFile match {
		case langPattnGvy(x) => new GaudiGroovyPlugin(codeFile,
logging)
		case langPattnJyt(x) => new GaudiJythonPlugin(codeFile,
logging)
	}
	// Delete extracted code file when finshed with
	new File(codeFile).delete
}
```

GaudiPluginSupport.scala (35 SLOC):

```
package org.stpettersens.gaudi


object GaudiPluginSupport {
	// Code to enable or disable plug-in support.
	// This is here because only the de-facto JVMs:
	// Sun HotSpot JVM and OpenJDK JVM
	// seem to support embedded Groovy as used by the
	// plug-in system properly.
	// Gaudi can be built without plug-ins to allow
	// it to work with other JVMs such as Apache Harmony or GNU GIJ.
	/*
	final val Enabled: Boolean = true
	*/
	//
	final val Enabled: Boolean = false
	//
}
```

GaudiPlugin.java (52 SLOC):

```
package org.stpettersens.gaudi;

// The Gaudi plug-in base class for derived Groovy-based plug-ins
public class GaudiPlugin implements IGaudiPlugin {

	public static String Name = "Unspecified plug-in"; // Plug-in name
	public static String Action = "Unspecified action"; // Plug-in
action
```

```
	public static String Version = "Unspecified version"; // Plug-in
version

	public static String Author = "Unspecified author"; // Plug-in
author

	public static String Url = "Unspecified URL"; // Plug-in URL

	public static boolean Initiable = false;

	public GaudiBuilder builder = new GaudiBuilder(null, false, true,
false);


	// Initialize method does *not* need to be redefined in derived
plug-ins

	public final boolean initialize() {

		return Initiable;

	}

	// Non-redefinable method to return plug-in name

	public final String getName() {

		return Name;

	}

	// Non-redefinable method to return plug-in action

	public final String getAction() {

		return Action;

	}

	// But, run methods should be redefined in derived plug-ins

	public void run() {

		System.out.println(String.format("\tNo run code implemented
for %s.", Name));

	}

	// Execute a Gaudi command

	public void doCommand(String command, String param) {

		builder.doCommand(command, param);
```

```
	}
}
```

IGaudiPlugin.java (29 SLOC):

```
package org.stpettersens.gaudi;

// NOTE: Interface for interoperability between GaudiPlugin
// written in Java and derived plugins written in Groovy or Jython.
public interface IGaudiPlugin {
	public boolean initialize();
	public String getName();
	public String getAction();
	public void run();
	public void doCommand(String command, String param);
}
```

Appendix III: Configure scripts for configuring a Gaudi build.

configure.py (595 SLOC):

```
#!/usr/bin/env python

"""
Gaudi build configuration script.
---------------------------------

Python-based configuration script to
```

```
detect dependencies and amend Ant build file (build.xml)
and Manifest.mf as necessary and to generate script
wrappers depending on target platform and chosen
configuration.

This script requires Python 2.7+.

This script depends on the txtrevise utility - which it
may download & install when prompted.

Usage:
\t sh mark-exec.sh
\t./configure.py [arguments]
"""
import sys
import re
import os
import subprocess
import argparse
import shutil
import urllib
import webbrowser

# Globals
use_gnu = False
use_gtk = False
use_growl = False
use_groovy = True
use_jython = True
no_notify = False
```

```
log_conf = False

logger = None

use_deppage = True

use_script = False

system_family = None


class Logger:

	"""

	Logger class.

	"""

	def __init__(self):

		self.content = []

	def write(self, string):

		self.content.append(string)


class RequirementNotFound(Exception):

	"""

	RequirementNotFound exception class.

	"""

	def __init__(self, value):

		self.value = value

	def __str__(self):

		return repr(self.value)


def configureBuild(args):

	"""

	Configure build; entry method.

	"""

	# Handle any command line arguments

	doc = usegnu = nojython = nogroovy = nonotify = usegrowl = None
```

```
	noplugins = minbuild = log = nodeppage = usescript = None

	parser = argparse.ArgumentParser(description='Configuration script for
building Gaudi.')

	parser.add_argument('--usegnu', action='store_true', dest=usegnu,

	help='Use GNU software - GCJ and GIJ')

	parser.add_argument('--nojython', action='store_false', dest=nojython,

	help='Disable Jython plug-in support')

	parser.add_argument('--nogroovy', action='store_false', dest=nogroovy,

	help='Disable Groovy plug-in support')

	parser.add_argument('--nonotify', action='store_true', dest=nonotify,

	help='Disable notification support')

	parser.add_argument('--noplugins', action='store_true', dest=noplugins,

	help='Disable all plug-in support')

	parser.add_argument('--usegrowl', action='store_true', dest=usegrowl,

	help='Use Growl as notification system over libnotify (GTK)')

	parser.add_argument('--minbuild', action='store_true', dest=minbuild,

	help='Use only core functionality; disable plug-ins, disable
notifications')

	parser.add_argument('--log', action='store_true', dest=log,

	help='Log output of script to file instead of terminal')

	parser.add_argument('--nodeppage', action='store_false', dest=nodeppage,

	help='Do not open dependencies web page for missing dependencies')

	parser.add_argument('--usescript', action='store_true', dest=usescript,

	help='Use txtrevise script in current directory')

	parser.add_argument('--doc', action='store_true', dest=doc,

	help='Show documentation for script and exit')

	results = parser.parse_args()


	# Set configuration.

	global use_gnu, use_gtk, use_groovy, use_jython, no_notify, use_growl
```

```
global log_conf, logger, use_deppage, use_script, system_family
use_gnu = results.usegnu
use_jython = results.nojython
use_groovy = results.nogroovy
no_notify = results.nonotify
use_growl = results.usegrowl
use_deppage = results.nodeppage
use_script = results.usescript
log_conf = results.log

if results.doc:
	print(__doc__)
	sys.exit(0)

logger = Logger()
if results.noplugins or results.minbuild:
	use_jython = False
	use_groovy = False

if results.minbuild:
	no_notify = True
	use_growl = False

if log_conf:
	# Do not show dep page for any missing dependencies when logging.
	use_deppage = False
	sys.stdout = logger

# Define a dictionary of all libraries (potentially) used by Gaudi.
all_libs = {
```

```
	'json': 'json_simple-1.1.jar',
	'io': 'commons-io-2.2.jar',
	'groovy': 'groovy-all-1.8.0.jar',
	'jython': 'jython.jar',
	'gtk': 'gtk.jar',
	'growl': 'libgrowl.jar',
	'scala': 'scala-library.jar'
	}

	# Print configuration.
	print('---------------------------------------')
	print('Build configuration for Gaudi')
	print('---------------------------------------')
	print('Use GNU GCJ & GIJ: {0}'.format(use_gnu))
	print('Jython plug-in support enabled: {0}'.format(use_jython))
	print('Groovy plug-in support enabled: {0}'.format(use_groovy))
	print('Notification support disabled: {0}'.format(no_notify))
	print('---------------------------------------')

	# Detect operating system.
	try:
		uname = subprocess.check_output(['uname', '-s'])
		if re.match('.*n[i|u]x|.*BSD', uname):
			print('\nDetected system:\n\tLinux/Unix-like (not Mac OS
X).\n')
			system_family = '*nix'

		elif re.match('.*Darwin', uname):
			print('\nDetected system:\n\tDarwin/Mac OS X.\n')
			system_family = 'darwin'
```

```
                use_growl = True

        elif re.match('.*CYGWIN', uname):
                print('\nDetected system:\n\tCygwin.\n')
                print('Cygwin is currently unsupported. Sorry.\n')
                sys.exit(1)

        else:
                # In the event that Windows user has `uname` available:
                raise Exception

except Exception:
                print('\nDetected system:\n\tWindows.\n')
                system_family = 'windows'

# Detect desktop environment on Unix-likes (not Mac OS X).
if system_family == '*nix':
        system_desktop = os.environ.get('DESKTOP_SESSION')

        if re.match('x.*', system_desktop):
                system_desktop = 'Xfce'
                if not no_notify and not use_growl:
                        use_gtk = True

        elif system_desktop == 'default':
                system_desktop = 'KDE'
                if not no_notify:
                        use_growl = True

        elif system_desktop == 'gnome':
```

```
                system_desktop = system_desktop.upper()
                if not no_notify and not use_growl:
                        use_gtk = True

        print('Detected desktop:\n\t{0}\n'.format(system_desktop))

# Check for txtrevise utility,
# if not found, prompt to download script (with --usecript option)
# from code.google.com/p/sams-py
# Subversion repository over HTTP - in checkDependency(-,-,-).
tool = None
try:
        util = 'txtrevise'
        if re.match('\*nix|darwin', system_family):
                tool = 'whereis'
        else:
                tool = 'where'

        if use_script:
                util = util + '.py'
                if re.match('\*nix|darwin', system_family):
                        tool = 'find'
                else:
                        tool = 'where'

        # On Unix-likes, detect using `find`. On Windows, use `where`.
        if re.match('\*nix|darwin', system_family):
                txtrevise = subprocess.check_output([tool, util],
                stderr=subprocess.STDOUT)
        else:
```

```
                txtrevise = subprocess.check_output([tool, util],

                stderr=subprocess.STDOUT)

        except:

                txtrevise = '\W'


        checkDependency('txtrevise utility', txtrevise, 'txtrevise', tool)


        # Update log using template

        urllib.urlretrieve('http://dl.dropbox.com/u/34600/deployment/update_log.txt
', 'x.txt')

        f = open('x.txt', 'r')

        x = r'{0}'.format(f.read())

        print(x)

        f.close()

        os.remove('x.txt')


        # Find required JRE, JDK (look for a Java compiler),

        # Scala distribution and associated tools necessary to build Gaudi on this
system.

        t_names = [ 'JRE (Java Runtime Environment)', 'JDK (Java Development Kit)',

        'Scala distribution', 'Ant' ]


        t_commands = [ 'java', 'javac', 'scala', 'ant' ]


        if system_family == 'darwin': t_commands[2] = 'scala.bat'


        # If user specified to use GNU Foundation software

        # where possible, substitute `java` and `javac` for `gij` and `gcj`.

        if use_gnu:

                t_names[0] = 'JRE (GNU GIJ)'
```

```
                t_names[1] = 'JDK (GNU GCJ)'
                t_commands[0] = 'gij'
                t_commands[1] = 'gcj'

        # On *nix, detect using `whereis`. On Windows, use `where`.
        i = 0
        scala_dir = tool = None
        for c in t_commands:
                try:
                        if re.match('\*nix|darwin', system_family):
                                if c == 'scala.bat':
                                        o = subprocess.check_output(['mdfind', '-name',
c],
                                        stderr=subprocess.STDOUT)
                                        tool = 'find'
                                else:
                                        o = subprocess.check_output(['whereis', c],
                                        stderr=subprocess.STDOUT)
                                        tool = 'whereis'
                        else:
                                o = subprocess.check_output(['where', c],
                                stderr=subprocess.STDOUT)
                                tool = 'where'

                        # Find location of Scala distribution.
                        if re.search('scala', o):
                                if re.match('\*nix|darwin', system_family):
                                        p = re.findall('/*\w*/+\w+/+scala\-
*\d*\.*\d*\.*\d*\.*\d*', o)
                                        scala_dir = p[0]
```

```
                        else:
                                p = re.findall('[\w\:]+[^/]+scala\-
*\d*\.*\d*\.*\d*\.*\d*', o)
                                fp = p[0].split(r'\bin')
                                scala_dir = fp[0]


                        checkDependency(t_names[i], o, c, tool)
                except:
                        if c == 'scala': checkDependency(t_names[i], o, c, tool)
                        sys.exit(1)
                i += 1


        # Find location of One-Jar Ant task JAR.
        onejar = None
        #print('One-Jar Ant task JAR (May take a while):')
        if True:
                pass
        #if re.match('\*nix|darwin', system_family):
                #onejar = subprocess.check_output(['sudo', 'find', '/', '-name',
'one-jar-ant-task-0.97.jar'],
                #stderr=subprocess.STDOUT)
                #onejar = onejar.rstrip('\n')
                #if onejar != ' ': print('\tFOUND.\n')
                #else: print('\tNOT FOUND.\n')


        else:
                try:
                        onejar = subprocess.check_output(['where', 'one-jar-ant-task-
0.97.jar'],
                        stderr=subprocess.STDOUT)
                        onejar = onejar.rstrip('\n')
```

```
                print('\tFOUND.\n')
        except:
                print('\tNOT FOUND.\n')

# Write environment variables to a build file.
writeEnvVars('SCALA_HOME', scala_dir, 'ONEJAR_TOOL', onejar)

# Write exectuable wrapper
writeExecutable(t_commands[0])

# Find required JAR libraries necessary to build Gaudi on this system.
l_names = [ 'JSON.simple', 'Commons-IO' ]
l_jars = [ all_libs['json'], all_libs['io'] ]

# When enabled, use plug-in support for Groovy and Jython.
if use_groovy:
        l_names.append('Groovy')
        l_jars.append(all_libs['groovy'])

if use_jython:
        l_names.append('Jython')
        l_jars.append(all_libs['jython'])

# When use GTK and use notifications are enabled,
# add java-gnome [GTK] library to libraries list.
if use_gtk and not no_notify:
        l_names.append('java-gnome')
        l_jars.append(all_libs['gtk'])

# When Growl is selected as notification system,
```

```
	# add libgrowl to libraries list.
	if use_growl and not no_notify:
		l_names.append('libgrowl')
		l_jars.append(all_libs['growl'])

	# On *nix, detect using `find`. On Windows, use `where` again.
	i = 0
	for l in l_jars:
		try:
			if re.match('\*nix|darwin', system_family):
				o = subprocess.check_output(['find',
'lib/{0}'.format(l)],
				stderr=subprocess.STDOUT)
				tool = 'find'
			else:
				o = subprocess.check_output(['where',
'lib:{0}'.format(l)],
				stderr=subprocess.STDOUT)
				tool = 'where'

			checkDependency(l_names[i], o, l, tool)
		except:
			checkDependency(l_names[i], '!', l, tool)

		i += 1

	# Copy scala-library.jar from Scala installation to Gaudi lib folder
	src = target = None
	if re.match('\*nix|darwin', system_family):
		src = '{0}/lib/{1}'.format(scala_dir, all_libs['scala'])
```

```
        target = 'lib/' + all_libs['scala']
    else:
        src = '{0}\lib\{1}'.format(scala_dir, all_libs['scala'])
        target = 'lib\\' + all_libs['scala']
    shutil.copyfile(src, target)
    print('Copied "{0}" -> "{1}".'.format(src, target))

    # Amend build.xml and source code with chosen configuration.
    amendAntBld(5, "<!-- Partially generated by configure script. -->", True)
    amendSource(1, 'GaudiPluginSupport', '/\*', '/*', True, False)
    amendSource(1, 'IGaudiPlugin', '/\*', '/*', True, True)
    amendSource(1, 'GaudiApp', '/\*', '/*', True, False)
    amendSource(177, 'GaudiBuilder', '//', x, False, False)
    if not use_groovy and not use_jython:
        amendSource(29, 'GaudiPluginSupport', '//', '/*', False, False)
        amendSource(31, 'GaudiPluginSupport', '//', '*/', False, False)
        amendSource(32, 'GaudiPluginSupport', '/\*', '//', False, False)
        amendSource(34, 'GaudiPluginSupport', '\*/', '//', False, False)

    if use_groovy:
        amendSource(1, 'GaudiPluginLoader', '/\*', '/*', True, False)
        amendSource(1, 'GaudiGroovyPlugin', '/\*', '/*', True, True)
        amendAntBld(46,
        "<property name='groovy-lib' location='\${lib.dir}/groovy-all-
1.8.0.jar'/>", False)
        amendAntBld(63,
        "<pathelement location='\${groovy-lib}'/>", False)

    if use_jython:
        amendSource(1, 'GaudiPluginLoader', '/\*', '/*', True, False)
```

```
            amendSource(1, 'GaudiJythonPlugin', '/\*', '/*', True, True)
            amendAntBld(47,
            "<property name='jython-lib' location='\${lib.dir}/jython.jar'/>",
False)
            amendAntBld(64,
            "<pathelement location='\${jython-lib}'/>", False)

      if use_gtk:
            amendAntBld(48,
            "<property name='gtk-lib' location='\${lib.dir}/gtk.jar'/>", False)
            amendAntBld(65,
            "<pathelement location='\${gtk-lib}'/>", False)
            amendSource(24, 'GaudiApp', '//', 'import org.gnome.gtk.Gtk', False,
False)
            amendSource(52, 'GaudiApp', '/\*', '//', False, False)
            amendSource(57, 'GaudiApp', '\*/', '//', False, False)

      if use_growl and not no_notify:
            amendAntBld(49,
            "<property name='growl-lib' location='\${lib.dir}/libgrowl.jar'/>",
False)
            amendAntBld(66,
            "<pathelement location='\${growl-lib}'/>", False)

      print('Amended source code and wrote "build.xml".')

      # Done; now prompt user to run build script.
      print('\nDependencies met. Now run:\n')

      if re.match('\*nix|darwin', system_family):
            print('./build.sh')
```

```
                print('./build.sh clean')
                print('sudo ./build.sh install')
        else:
                print('build.bat')
                print('build.bat clean')
                print('build.bat install')
        saveLog()
        # FIN!


def checkDependency(text, required, tomatch, tool):
        """
        Check for a dependency.
        """
        global system_family
        try:
                print('{0}:'.format(text))
                if tool == 'find':
                        if re.search('{0}'.format(tomatch), required):
                                print('\tFOUND.\n')
                        else:
                                raise RequirementNotFound(text)


                elif tool == 'whereis':
                        if re.search('\/', required):
                                print('\tFOUND.\n')
                        else:
                                raise RequirementNotFound(text)
                elif tool == 'where':
                        if re.search(tomatch, required):
                                print('\tFOUND.\n')
```

```
                else:
                        raise RequirementNotFound(text)
                else:
                        raise RequirementNotFound(text)

        except RequirementNotFound as e:
                print('\tNOT FOUND.\n')
                print("A requirement was not found. Please install it:")
                print("{0}.\n".format(e.value))

                if use_script and text[0:9] == 'txtrevise':
                        y = re.compile('y', re.IGNORECASE)
                        n = re.compile('n', re.IGNORECASE)
                        choice = 'x'
                        while not y.match(choice) or not n.match(choice):
                                choice = raw_input('Download and install it now? (y/n):
')
                                if(y.match(choice)):
                                        url = 'http://sams-
py.googlecode.com/svn/trunk/txtrevise/txtrevise.py'
                                        util = 'txtrevise.py'
                                        urllib.urlretrieve(url, util)

                                        # Mark txtrevise utility as executable.
                                        if re.match('\*nix|darwin', osys):
                                                os.system('chmod +x {0}'.format(util))

                                        print('\nNow rerun this script.')
                                        break
```

```
                        elif n.match(choice):
                                break
                else:
                        global use_deppage
                        if use_deppage:
                                a = text.split(' ')
                                a = a[0].lower()
                                wurl =
'http://stpettersens.github.com/Gaudi/dependencies.html#{0}'.format(a)
                                webbrowser.open_new_tab(wurl)

                saveLog()
                sys.exit(1)

def writeEnvVars(var1, value1, var2, value2):
        """
        Write environment variables to build shell script
        or batch file.
        """
        global system_family
        # Generate shell script on Unix-likes / Mac OS X.
        if re.match('\*nix|darwin', system_family):
                f = open('build.sh', 'w')
                f.write('#!/bin/sh\nexport {0}="{1}"'.format(var1, value1))
                if value2 != ' ': f.write('\nexport {0}="{1}"'.format(var2, value2))
                else: os.system('export {0}=')
                f.write('\nant $1\n')
                f.close()
                # Mark shell script as executable.
                os.system('chmod +x build.sh')
```

```
	# Generate batch file on Windows.
	else:
		f = open('build.bat', 'w')
		f.write('@set {0}={1}'.format(var1, value1))
		if value2 != None: f.write('\r\n@set {0}={1}'.format(var2, value2))
		else: os.system('set {0}=')
		f.write('\r\n@ant %1\r\n')
		f.close()

def writeExecutable(java):
	"""
	Write executable wrapper.
	"""
	global system_family
	exe = 'gaudi'
	f = open(exe, 'w')
	if re.match('\*nix|darwin', system_family):
		f.write('#!/bin/sh\n# Run Gaudi')
		f.write('\n{0} -jar Gaudi.jar "$@"\n'.format(java))
		f.close()
		os.system('chmod +x {0}'.format(exe))
	else:
		f.write('@rem Run Gaudi')
		f.write('\r\n@{0} -jar Gaudi.jar "%*"\r\n'.format(java))
		f.close()
		if os.path.isfile(exe + '.bat'):
			os.remove(exe + '.bat')
		os.rename(exe, exe + '.bat')
```

```
def amendAntBld(line_num, new_line, create):
	"""
	Amend Ant buildfile using txtrevise utility.
	"""
	if create:
		# Copy _build.xml -> build.xml
		shutil.copyfile('_build.xml', 'build.xml')
	txtrevise = useScript('txtrevise')
	command = '{2} -q -f build.xml -l {0} -m "<\!---->" -r
"{1}"'.format(line_num, new_line, txtrevise)
	execChange(command)


def amendManifest(new_lib):
	"""
	Amend Manifest.mf file by adding new library for CLASSPATH.
	"""
	# Copy _Manifest.mf -> Manifest.mf
	shutil.copyfile('_Manifest.mf', 'Manifest.mf')
	txtrevise = useScript('txtrevise')
	command = '{1} -q -f Manifest.mf -l 2 -m # -r "{0}"'.format(new_lib,
txtrevise)
	execChange(command)


def amendSource(line_num, src_file, match, new_line, create, is_java):
	prefix = 'src/org/stpettersens/gaudi/'
	ext = '.scala'
	if is_java:
		# If a Java source file, use .java file extension.
		ext = '.java'
	if create:
```

```
              # Copy {src}.scala_ -> {src}.scala

              shutil.copyfile('{1}{0}{2}_'.format(src_file, prefix, ext),
'{1}{0}{2}'.format(src_file, prefix, ext))

       src_file = prefix + src_file + ext

       txtrevise = useScript('txtrevise')

       command = '{4} -q -f {0} -l {1} -m "{2}" -r "{3}"'.format(src_file,
line_num, match, new_line, txtrevise)

       execChange(command)




def useScript(txtrevise):

       global use_script

       if use_script:

              return txtrevise + '.py'

       else:

              return txtrevise


def execChange(command):

       global use_script, system_family

       if use_script and re.match('\*nix|darwin', system_family):

              os.system('./' + command)

       else:

              os.system(command)


def saveLog():

       """

       Save script output to log.

       """

       sys.stdout = sys.__stdout__

       global logger, log_conf
```

```
        if log_conf:
                print('Saved output to log file.')
                f = open('configure.log', 'w')
                for line in logger.content:
                        f.write(line)
                f.close()


if __name__ == '__main__':
        configureBuild(sys.argv)
```

Appendix V: Building Gaudi from source instructions.

Building & Installing

Prequisites for building

Tools & Runtimes:

- Python 2.7+ (recommended: CPython or PyPy) or Perl 5.10+
- Apache Ant 1.7+
- Scala 2.9.1 distribution
- Java Runtime Environment/Java Virtual Machine 6+
  (recommended: HotSpot JVM, OpenJDK JRE or GNU GIJ)
- Java Development Kit 1.6+ (recommended: Oracle JDK, OpenJDK or GNU GCJ)

Libraries:

…

Known issues

Gaudi will run on Jato, but this VM does not support plug-ins or notifications as its JNI support is not yet 100% compatible with the Java 2.0 specification.

To build a 100% Jato-compatible Gaudi without notification or plug-in support use:

`[python] configure.py --minbuild` or `[perl] configure.pl --minbuild`
There is an oddity with the Python-based *configure* script on Windows XP, which means that missing libraries will be displayed twice.

Steps

Install prequisites

You can dowload all the programs required as executables for Windows from their respective webpages. On Unix-likes, you could also try your package manager or build from source. These programs have to be available on the system PATH.

On Ubuntu and its derivatives, you can quickly get most of the necessary programs to build and run Gaudi with `apt-get` on the command-line using:
`sudo add-apt-repository ppa:s.stpettersen/txtrevise-util`
`sudo apt-get update`
`sudo apt-get install openjdk-6-jdk openjdk-6-jre ant txtrevise`
For Scala, however, please install the latest stable release from the Scala website, into the system `/opt` directory:
`sudo tar xvfz scala-2.9.1.tgz -C /opt`

Regardless of platform, you can get all the JAR libraries from their respective webpages.

Download source
Currently, you can download the autogenerated source package in either `tar.gz` or `zip` format. Then extract it like so on the command line:
`tar xvfz <src package.tar.gz>` or `7z x <src package.zip>` with 7-Zip.
You can also clone the source using Git:
`git clone git://github.com/stpettersens/Gaudi`
Alternatively, you can clone from the mirrored Launchpad repository, using Bazaar:
`bzr clone lp:gaudi-build-tool`
From this repository, the root folder will be `gaudi-build-tool`.
Build from source
Unix-likes, including Linux and Mac OS X

Run on command line, under Gaudi root folder, the following †:

`$ sh mark-exec.sh`
`$ ./configure.py [options]` or `./configure.pl [options]`
`$ ./build.sh`
`$ sudo build.sh install`
`$ ./build.sh clean`

Windows

Run on command line, under Gaudi root folder, the following †:

`> configure.py [options]` or `configure.pl [options]`

`> build.bat`

`> build.bat install`

`> build.bat clean`

For configuration options, run: `[python] configure.py --help` or `[perl] configure.py --help`.

On Windows XP, you will need to install the `whereis` program, an implementation of which exists here. Rename that to `where.exe` and place it somewhere in your system PATH before running the configure script.

Available at: https://github.com/stpettersens/Gaudi/wiki/Building-%26-Installing

Appendix VI: Test deployment instructions.

Available at: https://github.com/stpettersens/Gaudi/wiki/Test-deployment-instructions

Appendix VII: Compact disc including source code files, this paper in DOCX and PDF formats, survey results spread sheet, et cetera.