Getting Started with GPCP

John Gough

September 4, 2004

This document applies to GPCP version 1.3 for JVM (Java Virtual Machine)

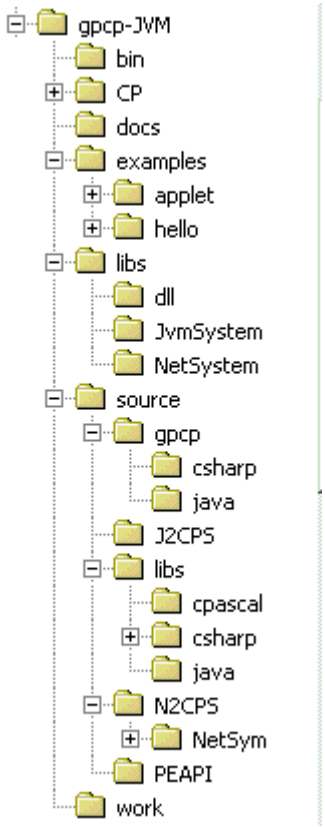

Figure 1: Distribution File Tree

1 Introduction

Gardens Point Component Pascal (*gpcp*) is an implementation of the Component Pascal Language, as defined in the Component Pascal Report from Oberon Microsystems. It is intended that this be a faithful implementation of the report, except for those changes that are explicitly detailed in the release notes. Any other differences in detail should be reported as potential bugs.

The compiler produces either Microsoft.NET intermediate language or Java byte-codes as output. Java byte codes are executed by a Java Virtual Machine (*JVM*). The compiler can be bootstrapped on either platform. These notes refer to the JVM platform. Details on the specifics of this implementation of Component Pascal are found in the release notes that come with the distribution.

2 Installing and Testing the Compiler

Environment

The compiler requires the Java Runtime Environment, running on any compatible platform. The released version of the compiler has been tested against version 1.4.2, but works with earlier version also. *gpcp* is packaged with an InstallShield installer for the Windows platform, or as a **tar**-ed and **gzip**-ed archive for other platforms.

On the Windows platform, the archive is typically expanded into a root directory named `\gpcp` and has several subdirectories. These include the binary files of the compiler, the documentation, the program examples, the library symbol files, and the source code of the compiler. If you plan to install both the *.NET* and the *JVM* version you may wish to place the two distribution trees in distinguished folders such as `\gpcp-CLR` and `\gpcp-JVM`

On *UNIX* platforms the system is usually installed in a shared directory, typically `/usr/local/gpcp`. An environment variable *CPROOT* is defined to point to the directory.

This section describes the steps required to install and try out the compiler

The distribution

The complete distribution tree is shown in Figure 1. The six first-level subdirectories of the distribution are

* **bin** — the binary files of the compiler
* **CP** — the class file tree of the tools and libraries
* **docs** — the documentation, including this file
* **examples** — some example programs
* **libs** — contains the simple library files
* **source** — the source files
* **work** — a working directory to play around with

The bin directory needs to be on your *PATH*. Typical commands to set this variables are —

```
set PATH=%PATH%;C:\gpcp\bin
```

On *UNIX* systems the environment variables would typically be set using commands such as —

```
PATH=$PATH:$CPROOT/gpcp/bin
```

Where *CPROOT* is the root of the *gpcp* distribution. The command file "`cprun`" will pass the environment variable *CPSYM* to the program, and will also set the class path.

The "`CP`" directory is the root of the class-file tree. This directory contains a sub-directory for each module of the system. There are almost 250 class files in the tree, in the initial distribution. However, when you run programs class files in the *local* class file directory take precedence over those in the *CPROOT* directory.

The "`libs`" directory contains the symbol files for the Component Pascal libraries. There are three subdirectories under "`libs`". The first of these is empty in the *JVM* version. The "`JvmSystem`" directory is for the symbols files to interface to the Java runtime. The "`NetSystem`" directory contains the symbol files that allow Component Pascal programs to access the base classes of the *.NET* system. This directory is empty in the *JVM* version.

Running your first program

Go to the "`work`" directory. With your favorite editor create the file (say) "`hello.cp`".

```
MODULE Hello;
  IMPORT CPmain, Console;
BEGIN
  Console.WriteString("Hello CP World");
  Console.WriteLn;
END Hello.
```

Make sure that `\gpcp\bin` is on the executable path.

From the command line, type

```
> cprun gpcp hello.cp
```

the system should respond

```
Created file CP\Hello\Hello.class
#gpcp: <Hello> no errors
> _
```

The file "`Hello.cps`" will have been created in the working directory. The "`.cps`" extension is the symbol file that declares the publicly accessible facilities of the program. By default *gpcp* will create a subdirectory in the working directory named "`CP`", with subdirectories for each module compiled. One or more class files will be created and placed in the subdirectories. In our example there is only one class file produced.

You may now run the program by the command "`cprun Hello`". Note carefully that the base class file has name "`Hello`", and the "`cprun`" command is case sensitive.

The examples

The example programs are in three sub-directories under the examples directory. The folder "`hello`" holds some simple command line programs. "`HelloWorld.cp`" is an elaborate version of the "`hello world`" canonical program. "`Nqueens.cp`" is a recursive backtracking version of the N-Queens problem solved for all board sizes from 8 to 13. "`Hennessy.cp`" is a version of the Hennessy integer benchmarks.

A file "`README.txt`" gives instructions for compiling and running each of the programs.

The folder `Applet3` has an example applet written in *Component Pascal*. The applet calculates spoke-lengths for bicycle wheels with various geometries. It is an example of use of the Abstract Window Toolkit, and how to write an applet in a language other than Java.

3 Browsing Modules

The *Browse* tool has been included with this release. This tool can show the exported interface for modules in either text or html format. Details on the use of this tool can be found in the Release Notes.

4 Reporting Bugs

If you find a bug

If you find what you believe is a bug, please send a report to gpcp@qut.edu.au with the detail of the event. It would be particularly helpful if you can send the code of the shortest program which can illustrate the error.

If the compiler crashes

The compiler has an outer-level exception rescue clause (you can see this in the body of procedure "`CPascal.Compile()`") which catches any exceptions raised during any per-file compilation attempt. The rescue code displays a "`<<compiler panic>>`" message on the console, and attempts to create a listing in the usual way. In most cases the rescue clause will be able to build an error message from the exception call chain, and will send this both to the screen and to the listing file.

In almost all cases, the compiler panic will be caused by failed error recovery in the compiler, so that the other error messages in the listing will point to the means of programming around the compiler bug. Nevertheless, it is important to us to remove such bugs from the compiler, so we encourage users who turn up error of this kind to send us a listing of a (hopefully minimal) program displaying the phenomenon.

In order to see how such a rescue clause works, here is an example of a program that deliberately causes a runtime error. When the program is run, the error is caught at the outer level and an error message is generated. After generating the error message, there is still the option of aborting the program with the standard error diagnostics. This is done by re-raising the same exception, and this time allowing the exception to propagate outwards to the invoking command line processor.

```
MODULE Crash;
  IMPORT CPmain, Console, RTS;

  TYPE OpenChar = POINTER TO ARRAY OF CHAR;
  VAR  p : OpenChar;

  PROCEDURE Catch;
  BEGIN
    p[0] := "a";(* line 9 *)
  RESCUE (exc)  (* exc is of type RTS.NativeException *)
    Console.WriteString("Caught Exception: "); Console.WriteLn;
    Console.WriteString(RTS.getStr(exc)); Console.WriteLn;
   (* THROW(exc) *)(* line 13 *)
  END Catch;

BEGIN
  Catch()(* line 17 *)
END Crash.
```

When this program is compiled and run, the following is the result —

```
> cprun gpcp Crash.cp
Created file CP\Crash\Crash.class
#gpcp: <Crash> No errors
> cprun Crash
Caught Exception:
    java.lang.NullPointerException> _
```

If the detailed stack trace is required, the exception may be re-raised by calling the non-standard built-in procedure *THROW*(), with the incoming exception as argument. The comment in the source shows where to place the call. If this is done then a full stack trace may be produced. For this example it is just —

```
Caught Exception:
    java.lang.NullPointerException
Exception in thread "main" java.lang.NullPointerException
   at CP.Crash.Crash.Catch(Crash.cp:9)
   at CP.Crash.Crash.main(Crash.cp:17)
```

You may care to note that the stack trace produced by the system starts at the line at which the exception was originally thrown, rather than the point at which it was explicitly re-thrown. This behaviour is different on the *.NET* platform.

Read the Release Notes!

There are a number of extensions to the language, many of these have been introduced so that Component Pascal programs will to be able to access all of the facilities of the *Java Runtime Environment*. Read the release notes to find out about all of these! In particular, from version 1.2.4 there is built-in support for extensible arrays (vectors).

Posting to the Mail Group

There is a discussion group for users of *gpcp*. You may subscribe by sending an email to GPCP-subscribe@yahoogroups.com. The development team monitor traffic on the group, and will post update messages to the group.