Clarion To Java

1 Contents

1Contents....1
2Introduction....2
  2.1What is Clarion?....2
  2.2What is Clarion2java?....2
  2.3Why Clarion2java?....2
3Getting Started....3
  3.1Build environment....3
  3.2Run the provided examples....5
  3.3Compile the Provided Examples....7
4Clarion2Java User Guide....10
  4.1Managing dependencies....10
  4.2Images....10
  4.3Default System icons....11
  4.4INI files....11
  4.5Windows Registry....11
  4.6Language Extensions....11
  4.7PostgreSQL backup and restore....13
  4.8Compiling clarion2java for yourself....15
  4.9Performance and Functional Considerations of the Runtime....16
5Known Limitations....20
  5.1Major limitations you may encounter....20
  5.2Minor limitations/unusual features which you may encounter....20
  5.3Other observations and comments about Clarion Language....21

2 Introduction

2.1 What is Clarion?

Clarion is a business oriented, rapid application development language that is owned and supplied by a Soft Velocity (http://www.softvelocity.com/). This project is in no way affiliated with Soft Velocity.

2.2 What is Clarion2java?

clarion2java allows you to take Clarion source code and recompile it as an runnable java program.

Normally when developing Clarion software using Soft Velocity's IDE, you would select the task 'build' in order to compile the clarion application into a runnable Win16/Win32 .exe file.

But with clarion2java – the process changes. You only click on “**Project**” – “**Generate”** to use Clarion source code from the Clarion application, and then you use the clarion2java program to convert this source code from clarion source code into java source code.

clarion2java is **not** what is known to clarion programmers as a template. It does not embed into the IDE and allow you to visually generate java code. It operates on a much lower level and after your templates have finished generating source code.

clarion2java at time of writing is geared towards Clarion versions upto Clarion 6. Clarion 5.5 is the most well tested as this is what original project author uses. It should be possible for clarion2java to also support 7 with some further, minimal work.

2.3 Why Clarion2java?

See the document WhyClarion2Java.pdf

3 Getting Started

3.1 Build environment

3.1.1 Java Development Kit

Clarion2java requires java 6 development kit, release 10 or upwards. The JDK can be downloaded here

http://java.sun.com/javase/downloads/index.jsp

After you have downloaded and installed the JDK – it is recommended that you add the environment variable JAVA_HOME into your user environment to point to the installed directory for java and to include the bin directory within java in your PATH. For example if you installed JDK 1.6 update 18, then your environment should look like this:

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Andrew>echo %JAVA_HOME%
C:\Program Files\Java\jdk1.6.0_18

C:\Documents and Settings\Andrew>echo %PATH%
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;c:\cfd;c:\tools;c:\cygwi
n;c:\cygwin\bin;C:\Program Files\Common-Use Signing Interface\JRE\bin;C:\Program
 Files\Common Files\Intuit\QBPOSSDKRuntime;c:\program files\java\jdk1.6.0_18\bin


C:\Documents and Settings\Andrew>java -version
java version "1.6.0_18"
Java(TM) SE Runtime Environment (build 1.6.0_18-b07)
Java HotSpot(TM) Client VM (build 16.0-b13, mixed mode, sharing)

C:\Documents and Settings\Andrew>javac -version
javac 1.6.0_18

C:\Documents and Settings\Andrew>
```

3.1.2 Maven

The Clarion2Java build environment is geared around a build system called maven. Maven can be downloaded here:

http://maven.apache.org/

Maven simplifies alot of common project concerns such as automated release and dependency resolution, but it comes at the expense of increased complexity, particularly in setup. The learning curve with Maven is exceedingly (and some argue unnecessarily) steep. The example projects that ship with clarion2java provide archetype project files (pom.xml files) which is recommended that you simply copy and modify to change project naming.

Hopefully, for typical usage, detailed understanding of how to use maven will not be necessary. This document will outline usage of Maven via common recopies which you follow. Also included are examples of manual command line compilation.

Install Maven to where you prefer and like Java Development Kit, add it into your PATH environment.

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Andrew>echo %PATH%
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;c:\cfd;c:\tools;c:\cygwi
n;c:\cygwin\bin;C:\Program Files\Common-Use Signing Interface\JRE\bin;C:\Program
 Files\Common Files\Intuit\QBPOSSDKRuntime;c:\program files\java\jdk1.6.0_18\bin
;c:\maven\bin

C:\Documents and Settings\Andrew>mvn -version
Apache Maven 2.2.1 (r801777; 2009-08-07 05:16:01+1000)
Java version: 1.6.0_18
Java home: C:\Program Files\Java\jdk1.6.0_18\jre
Default locale: en_AU, platform encoding: Cp1252
OS name: "windows xp" version: "5.1" arch: "x86" Family: "windows"
C:\Documents and Settings\Andrew>
```

3.1.3 Setting up Maven

Maven manages and downloads resources off the Internet and stores them on your computer in a local repository.

The local repository is located, by default, in your %HOMEDIR% in a directory called .m2

All the resources that Clarion2Java needs are stored in Maven's central repository. So under most normal circumstances, no additional configuration is required. Maven should be able to locate all the resources it needs in order to compile clarion code into java code out of the box.

The first time you run a maven task you need to be connected to the Internet. Maven will first look in your local repository for resource files, if it cannot find them it will search its repository listing. Once located, resources are locally cached. Meaning that after initial update, you no longer need to be online in order to development work.

You can browse maven central repository, which currently contains 100s of thousands of java opensource projects directly here :

http://repo1.maven.org/maven2/

To browse Clarion2Java resources:

http://repo1.maven.org/maven2/org/jclarion/clarion/

For more sophisticated enterprises, you may wish to run a local maven repository, or use a maven repository mirror. See http://apache.maven.org for details on how to do this.

3.2 *Run the provided examples*

Three precompiled examples are provided.

- A trivial HelloWorld application
- A simple Clarion 5.5 MDI Application Generated program. (Cookbook)
- A simple Clarion 6.0 MDI Application Generated program. (MusicDB)

The examples ship with clarion2java in the jar directory:

```
C:\Documents and Settings\Andrew>cd clarion2java

C:\Documents and Settings\Andrew\clarion2java>cd jar

C:\Documents and Settings\Andrew\clarion2java\jar>dir
 Volume in drive C has no label.
 Volume Serial Number is 8C31-F7EE

 Directory of C:\Documents and Settings\Andrew\clarion2java\jar

14/05/2010  08:22 AM    <DIR>          .
14/05/2010  08:22 AM    <DIR>          ..
23/05/2010  01:19 PM           522,141 cookbook-1.4.jar
23/05/2010  01:11 PM           359,366 clarion-compile-1.19.jar
14/05/2010  04:34 AM            31,881 image4j-0.7.jar
22/11/2009  04:30 PM           510,170 postgresql-8.4-701.jdbc4.jar
23/05/2010  01:20 PM           612,897 musicdb-1.3.jar
23/05/2010  12:52 PM           893,605 clarion-runtime-1.53.jar
23/05/2010  01:19 PM             3,058 helloworld-1.5.jar
23/05/2010  12:28 PM            17,217 clarion-sysruntime-1.11.jar
01/11/2009  05:45 PM         1,130,070 itext-2.1.7.jar
               9 File(s)      4,034,955 bytes
               2 Dir(s)     790,761,472 bytes free
```

3.2.1 HelloWorld

To run HelloWorld, type in:

```
java -jar helloworld-1.5.jar
```

This application will pop up a trivial Hello World Message Box.

You can also double click on the .jar file in Windows explorer to run it.

3.2.2 CookBook

Cook Book is a simple app generated MDI application, using a .DCT file containing three tables. No work has been done at all after App Generation, so the application is not useful. It's purpose is to demonstrate MDI.

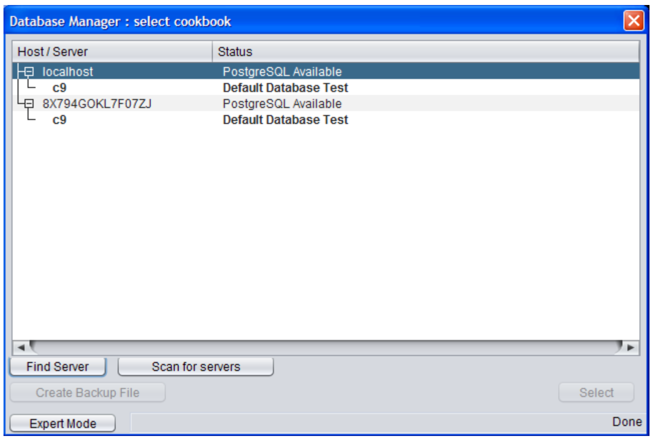
CookBook requires PostgreSQL to be installed, either on your computer or on a nearby computer on your windows network.

To run CookBook, type in:

```
java -jar cookbook-1.4.jar
```

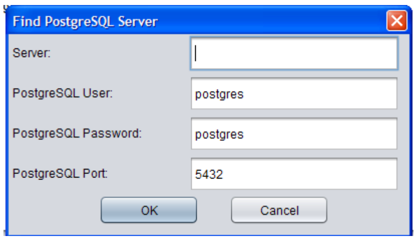
The first time you run cookbook it will popup a screen which allows you to manage your database. This screen is designed to replace the ODBC configuration process.

If there is a nearby copy of postgres available, the manager will try to automatically find it.



Host / Server	Status
localhost	PostgreSQL Available
c9	Default Database Test
8X794GOKL7F07ZJ	PostgreSQL Available
c9	Default Database Test

If the system cannot find your copy of PostgreSQL – click on Find Server to type in the exact host and username details to connect to a administrator level user in postgres.

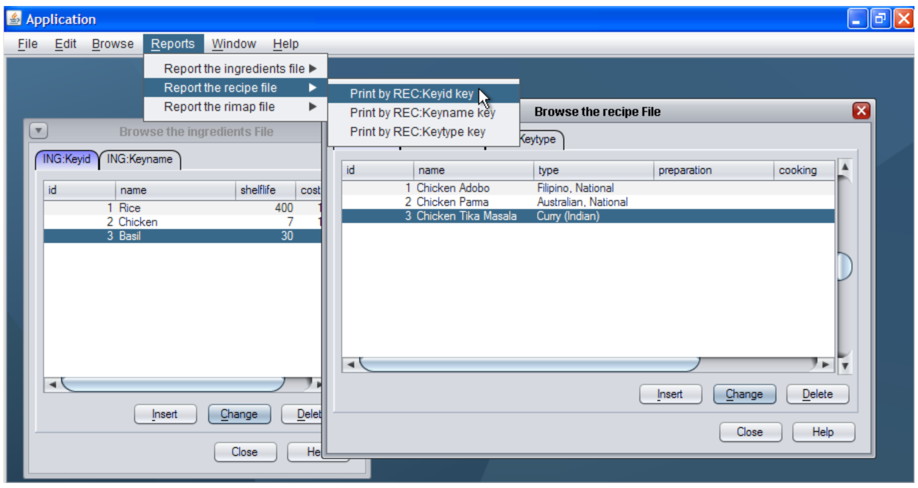


Server:	
PostgreSQL User:	postgres
PostgreSQL Password:	postgres
PostgreSQL Port:	5432

Once you have a postgres server. Do the following:

- Highlight the server
- Click on **Export Mode**
- Click **Create New DB**
- Click **OK** to accept the name **cookbook** as your database name.
    - This will create a database where database, user and password are all set to 'cookbook'
- Now highlight the newly created 'cookbook' database and click **select**

The application should now begin.



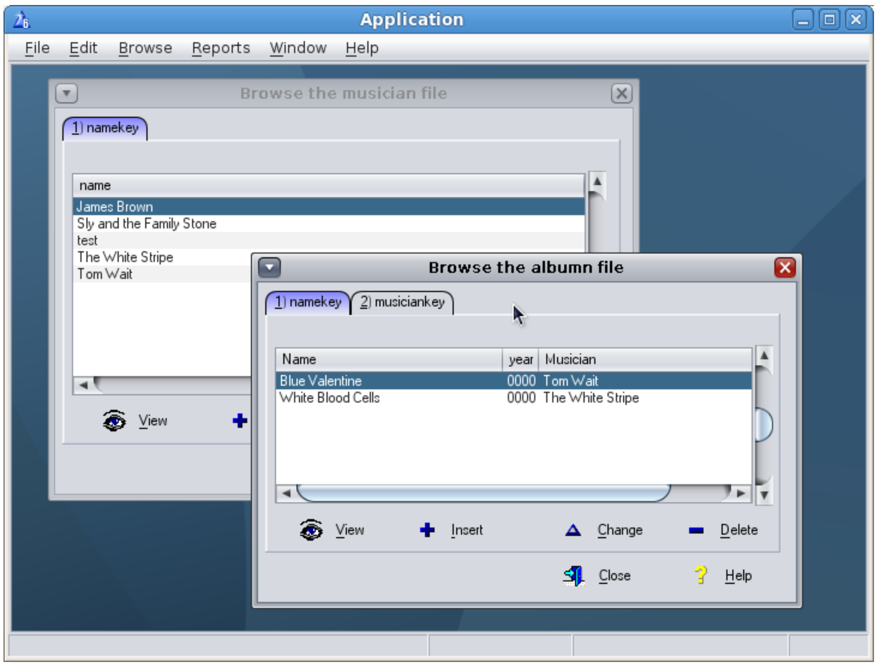
id	name	shelflife	cost
1	Rice	400	1
2	Chicken	7	1
3	Basil	30	

id	name	type	preparation	cooking
1	Chicken Adobo	Filipino, National		
2	Chicken Parma	Australian, National		
3	Chicken Tika Masala	Curry (Indian)		

Note that database details are saved in a file called db.properties. Next time you start cookbook, these details will be reused. If you delete db.properties, the file manager will be re-presented.

Setup of MusicDB is very similar to CookBook.

```
java -jar musicdb-1.3.jar
```



name
James Brown
Sly and the Family Stone
test
The White Stripe
Tom Wait

Name	year	Musician
Blue Valentine	0000	Tom Wait
White Blood Cells	0000	The White Stripe

3.3 *Compile the Provided Examples*

Examples ship with the clarion2java bundle in the examples directory.

3.3.1 Hello World

Go into hello world directory and simply run the command mvn. The first time you run mvn, it will take a long time and you will see alot of logging. This is because it is downloading lots of maven core support files, such as the actual compiler (you must be connected to the internet the first time you run this). The maven you downloaded is just a simple skeleton. Downloads are cached in .m2 directory. Subsequent runs are faster.

```
C:\Documents and Settings\Andrew\clarion2java\examples\HelloWorld>mvn
[INFO] Scanning for projects...
[INFO] ----------------------------------------------------------------------
[INFO] Building clarion
[INFO]    task-segment: [assembly:assembly] (aggregator-style)
[INFO] ----------------------------------------------------------------------
[INFO] Preparing assembly:assembly
[INFO] ----------------------------------------------------------------------
[INFO] Building clarion
[INFO] ----------------------------------------------------------------------
   < .....   SNIP .... >
[INFO] ----------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ----------------------------------------------------------------------
[INFO] Total time: 7 seconds
[INFO] Finished at: Mon Apr 12 07:13:07 EST 2010
[INFO] Final Memory: 17M/42M
[INFO] ----------------------------------------------------------------------
C:\Documents and Settings\Andrew\clarion2java\examples\HelloWorld>
```

This will generate a .zip file in the directory target\. Unzip the .zip file and run the helloworld-1.5.jar file placed in there.

Tip: If you make changes to helloworld.clw and include constants such as **event:Accepted**, **prop:hide** etc, you must copy both equates.clw and property.clw from libsrc in your Clarion installation into src\main\clarion directory. If you do not want to do this you can always substitute actual values of these constants in code. Although the code wont be very easy to read.

3.3.2 Cookbook and MusicDB

Cookbook requires some more preparation as you need to copy in your entire clarion libsrc directory, so that clarion2java can compile ABC and other common libraries.

The demo does not ship with these files! You must source them yourself, from a legal copy of Topspeed's Clarion product. For Cookbook you need a libsrc from a Clarion 5.5 installation. For MusicDB you need Clarion 6. It is also recommended that you use clarion IDE to regenerate the source code from the supplied .app file to account for any customisation or variation in your template/library installation.

In examples\cookbook\src\main\clarion – you create a libsrc directory and copy in the contents of you libsrc from clarion.

Once libsrc is copied, then run mvn to compile.

3.3.3 Alternative build systems

If you do not want to use maven, but instead wish to use tooling in a preferred Java IDE or Ant, then you can simply use your preferred method.

Note that this section assumes you know how to use the build environments in question. Clarrion2java, at time of writing only supports maven. This document will not service as a complete tutorial for how to build using ant for example, as it is assumed that you know how to manage ant already.

Clarion2java works by transforming clarion source code into java source code. Clarion2java then relies on maven to invisibly transform that code into java byte code.

You can manually perform the transform step thus. The command is:

```
java -jar clarion-compile-1.19.jar <sourcedir> <main.clw> <targetdir>
```

Using the cookbook example : from the CookBook directory:

```
>java -jar ..\..\jar\clarion-compile-1.19-SNAPSHOT.jar
Compile: <sourceDirectory> <main.clw> <targetDirectory>

>mkdir output

>java -jar ..\..\jar\clarion-compile-1.19.jar  src\main\clarion cookbook.clw output
214 Files loaded
12/04/2010 8:09:33 AM org.jclarion.clarion.compile.expr.VariableExpr assign
WARNING: Not appropriate to use reference assigment for variable:fld
12/04/2010 8:09:33 AM org.jclarion.clarion.compile.scope.Scope matchProcedureImplementation
WARNING: Could not match procedure (ConcatGetComponents) based on typing: do length only match
12/04/2010 8:09:34 AM org.jclarion.clarion.compile.scope.Scope matchProcedureImplementation
WARNING: Could not match procedure (AddTranslation) based on typing: do length only match
12/04/2010 8:09:34 AM org.jclarion.clarion.compile.scope.Scope matchProcedureImplementation
WARNING: Could not match procedure (AddErrors) based on typing: do length only match
12/04/2010 8:09:34 AM org.jclarion.clarion.compile.scope.Scope matchProcedureImplementation
WARNING: Could not match procedure (RemoveErrors) based on typing: do length only match
Wrote 341 java file(s)
Total run time:0.2905s
```

You will then need to compile the contents of output yourself. i.e. the following will work.

```
D:\clarion2java\examples\CookBook\output>javac -cp ..\..\..\jar\clarion-runtime-
1.53.jar clarion\*.java clarion\equates\*.java

D:\clarion2java\examples\CookBook\output>
```

Then you can package and setup Manifest as required. For hints on dependency requirements, see contents of jar\ directory.

For Main: Manifest entry – set this to clarion.Main. The main program code start always compiles as a **static void main(String args[])** method inside **clarion.Main**, irrespective of what the clarion project, or main .clw file name is.

Note that the resulting .java files in the compile process are currently UNIX text files : with \n line delimiters, not \r\n as required by Microsoft OS types. Result is that some text editors, like notepad, will not format java source code nicely.

4 Clarion2Java User Guide

4.1 *Managing dependencies*

In your project pom.xml file are dependencies on clarion2java expressed.

The key entries of interest are:

```
        <plugin>
                <groupId>org.jclarion</groupId>
                <artifactId>clarion-maven-plugin</artifactId>
                <version>1.19</version>
                <executions>
                        <execution>
                                <goals>
                                        <goal>clarion</goal>
                                </goals>
                        </execution>
                </executions>
                <configuration>
                        <mainSourceFile>c8.clw</mainSourceFile>
                </configuration>
        </plugin>
```

This sets a dependency on 1.19 of the clarion maven plugin, which in turn has a dependency on the 1.19 of the clarion compiler. If you want to change the version of the compiler, change this.

```
    <dependency>
        <groupId>org.jclarion</groupId>
        <artifactId>clarion-runtime</artifactId>
        <version>1.53</version>
    </dependency>
```

This sets the dependency on version 1.53 of the clation runtime system which in turn defines dependencies on items such as Postgres JDBC driver, itext PDF tools, image4j (see images below) and other items.

4.2 *Images*

Put all images used by the application into src/main/resources/resources/images directory.

These will be automatically compiled into the .jar file and used from within the application.

Java by default supports JPG, GIF and PNG file formats.

Support for ICO and BMP files is provided courtesy of image4j library.

http://image4j.sourceforge.net/

4.3 Default System icons

The default system icons are stored in clarion-runtime-1.53.jar. These icons come courtesy of a mix of open source icon packages.

- Crystal Project (http://www.everaldo.com/crystal/)
- Tango Icon Themes (http://tango.freedesktop.org/Tango_Desktop_Project)
- Silk Icons (http://www.famfamfam.com/lab/icons/silk/)

Icons in use can be easily browsed via sub version:

http://clarion2java.svn.sourceforge.net/viewvc/clarion2java/runtime/trunk/src/main/resources/resources/images/clarion/

4.4 INI files

Java does not have a concept of an INI file. It instead uses a notion of property files (i.e. db.properties defined above).

Clarion2java maps INI file calls to fetch and update a corresponding .properties file in the current directory. If a properties file does not exist, clarion2java runtime searches c:\windows\ for a corresponding .ini file and imports/converts it to .properties first.

4.5 Windows Registry

Java has no mechanism to access windows registry. The clarion 6 functions regget, regput and regdelete are implemented as calls to windows REG.EXE program.

The result is that registry access in clarion2java is very very slow. It is recommended to avoid using windows registry. Also it means your clarion2java code is not portable to non Windows systems.

4.6 Language Extensions

4.6.1 Java equate

As described earlier, compiling with clarion2java forces the equate JAVA to always evaluate to 1. This can be used in compile/omit blocks to compile in/out java specific concerns.

4.6.2 @java extensions

Clarion2java introduces a new lexical structure : @java-*. These tokens permit importing of java classes, dependencies and inlining of java code.

4.6.2.1 *@java-import 'classfile'*

This command loads in the nominated java file and links it into the compiler as though it is a clarion class file. Only zero parameter constructor classes can be used this way. The class in question must be in the class path of the clarion2java compiler to work.

@java-import can appear anywhere in the code

```
databaseScanner routine
    compile('java-end',java)
    @java-import 'com.c8systems.c9.scanner.Scanner'
    data
scanner &Scanner
    code
    scanner &= new scanner
    scanner.run()
```

4.6.2.2 *@java-code 'java source'(parameters),[modifier]*

This command allows you to insert java code. It can only be used in code blocks.

The parameters provided are clarion expressions and are substituted into the java code by replacing '$' characters in the java source string in the order that they occur.

Modifiers can be any of the following:

- RETURN
- BREAK
- CONTINUE
- RETURN
- END

The provide hints to clarion2java compiler for purposes to determining code reachability which is very important concern for java source code.

```
@java-code 'System.out.println($.toString());'(SomeVariable)
```

4.6.2.3 *@java-dependency 'classfile'*

This command specifies that the relevant string needs to included as a import declaration in the relevant .java file. This is commonly used in conjunction with java-code.

```
@java-dependency 'java.sql.SQLException;'
@java-code 'throw new SQLException(“An Error”);(),RETURN
```

4.6.2.4 *@java-load 'classfile'*

Java load denotes special classes which insert additional symbols into the compiler's symbol list – so to compliment OPEN(Report), Close(Report) etc we can include CancelPrint(report).

The intention of this is that by default, we want clarion2java to accurately modell the original clation language specification without introducing too many extensions – to keep the language implementation as close to the original as possible. But we want a mechanism that permits loading in of clarion2java extensions for ease of use – to provide programming ease beyond the above defined @java extensions.

For example if you add the following statement:

```
@java-load 'org.jclarion.clarion.compile.ext.Print'
```

This will add new system commands which you can use in your clarion code:

CANCELPRINT (REPORT)	Cancel a print job on an open report.
GETPDF (REPORT), STRING	Print report to a virtual file as PDF

```
@java-load 'org.jclarion.clarion.compile.ext.File'
```

CopyFileIntoMemory(STRING), STRING

Take file denoted by input and load it into memory. The String result donates the name of the virtual file in memory.

```
@java-load 'org.jclarion.clarion.compile.ext.Font'
```

AddFont(STRING)

Add the specified font file as a recognised font. No need to install font into Windows font directory. Only TTF fonts supported.

```
@java-load 'org.jclarion.clarion.compile.ext.Crash'
```

CRASHLOG(STRING)	Add a string entry into crash log
CRASH	Cause system to crash and popup crash dialog

4.7 PostgreSQL backup and restore

The PostgreSQL Manager comes with its own backup and restore utilities. For these to work they need to be configured properly. You need to compile in two files into src/main/resources/resources directory.

They are:

schema.sql this is the output of pg_dump command. Recommended to be run this way:

```
pg_dump <database> -U <user> -O -s
```

For example:

```
pg_dump c8 -U c8 -O -s
```

You can automate this in your pom.xml file:

```
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.1</version>
        <executions>
          <execution>
            <phase>generate-resources</phase>
            <goals>
              <goal>exec</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <executable>pg_dump</executable>
          <outputFile>src/main/resources/resources/schema.sql</outputFile>
          <arguments>
            <argument>c8</argument>
            <argument>-U</argument>
            <argument>c8</argument>
            <argument>-O</argument>
            <argument>-s</argument>
          </arguments>
        </configuration>
      </plugin>
```

Finally – one more file needs to be defined : **schema.properties**

It defines a number of things. See following example:

```
dbtest=select  coalesce((SELECT  pvalue  from  prop  where  pgroup\='GUser'  and
pkey\='Name2'),'*** Empty C8 Database ***')

encoding=UTF8

tables.ignore=client_files

seq.ledger=ledger_primary_key_seq
seq.jobheader=jobheader_dockey_seq

restore.1=select  'Last  Invoice  Date\:  '||coalesce(  (SELECT  max(date)  from
ledger)\:\:varchar,'Empty System')
restore.2=select  'Shop  name\:  '||coalesce((SELECT  pvalue  from  prop  where
pgroup\='GUser' and pkey\='Name2'),'*** Empty C8 Database ***')
```

dbtest defines an SQL string to run against a database to determine whether or not the -database schema is installed already or not. The returned string from this SQL statement is presented to the user in the PostgreSQL manager dialog.

tables.ignore is a comma separated list of tables not to backup.

seq.<table> defines sequences that are relevant to a given table. PostgreSQL backup and restore allows you to backup and restore individual tables. It is necessary for system to understand how sequences relate, if at all.

restore.<n> Defines SQL queries to run on the database to display to the user so the user can see that if they are restoring to an actual database, the relevancy or the data they will be restoring over.

Encoding Defines the encoding to use when creating a new database. If not set then let Postgres decide. (Depends on install of Postgres, the OS and for windows the default codepage).

4.8 Compiling clarion2java for yourself

Use subversion to checkout the 4 main libraries:

```
svn co https://clarion2java.svn.sourceforge.net/svnroot/clarion2java/sysruntime/trunk
svn co https://clarion2java.svn.sourceforge.net/svnroot/clarion2java/runtime/trunk
svn co https://clarion2java.svn.sourceforge.net/svnroot/clarion2java/compiler/trunk
svn co https://clarion2java.svn.sourceforge.net/svnroot/clarion2java/plugin/trunk
```

Then in each directory – to compile – simply run

```
mvn
```

Special note with compiling runtime library. The test cases in there can take a long time to run and they have not been tested with Windows systems. There are a large number of tests that test things such as font sizing, entry and string positioning and keyboard and mouse input. The tests were designed to be run under linux and may not pass on windows. So to disable tests, run the following:

```
mvn -D maven.test.skip=true
```

4.8.1 Compiling your code against self compiled clarion2java

Say for example you make a change in runtime.

In runtime pom.xml you will see something like this:

```
  <groupId>org.jclarion</groupId>
  <artifactId>clarion-runtime</artifactId>
  <packaging>jar</packaging>
  <version>1.53-SNAPSHOT</version>
```

In order for your code to compile against this – you need to change your dependency on clarion-runtime to **1.53-SNAPSHOT.** Same applies for plugin, compiler and the sys-runtime.

Keep in mind that any changes you make to clarion2java source code itself are subject to clarion2java's licence. Clarion2java is licenced under LGPL ("Lesser"), so any changes must also be released under the LGPL licence.

Note that this licence obligation does **not** apply for your clarion code or any other library code you write and reference using '@java-' extensions.

Any code you write in clarion or write in java and reference using '@java-' is yours to do as you please. But if you modify clarion2java itself and then distribute the compiled result of that code, you must also distribute the source code changes under the L-GPL licence.

4.9 *Performance and Functional Considerations of the Runtime*

4.9.1 OVER() attribute

Clarion2Java supports the OVER() attribute but it incurs a substantial performance overhead in Java and under some circumstances the cost will be enough to have a visible impact on the application.

Java is a memory managed language and it does not permit you to access or manipulate memory directly or overlay variables with an OVER() equivalent. The details of the clarion memory model are completely invisible to the programmer.

Clarion2java solves this by setting up a system where changes to variables are monitored and then when a change occurs it propagates to related OVER() variables in such a way as though they were overlaid in a memory system mimicking x86/Clarion memory model.

The result is slow software because memory architecture of x86 is emulated. It is recommended to avoid using this. For java – implementing assignments and variable copies will under most circumstances be faster than using OVER. This is particularly noticeable in batch processing arrangements. Such as overlaying a DOS File with a single string record over some sort of record delimiter group.

4.9.2 AUTO, DISPLAY and Windows

Clarion2java uses change detection capabilities to determine when window controls should be refreshed : resulting in possibly faster window performance.

As such the AUTO property is redundant and ignored by clarion2java.

The STRING and LIST control, and a few others, though will defer redraw until the top of the ACCEPT loop. So if you are in the middle of an accept LOOP, issuing multiple changes to variable that is USE()'d by a STRING control will not cause the display to update immediately. Only when the ACCEPT loop cycles through.

But if you issue a DISPLAY, then any changes pending to be redrawn are flushed immediately.

4.9.3 THREAD() attribute

Clarion2java has strong native thread support and calling START() will result in a new native thread being invoked.

Clarion2java supports THREAD sensitive variables, by setting THREAD attribute.

4.9.4 THREADED FILES

Clarion2java threaded files behave differently than clarion.

In clarion, only the buffer is threaded. File state is still global.

But in clarion2java – the file state is also threaded. This means that you can have multiple threads performing file operations without the threads having to 'handover' file state by saving then restoring file state.

The consequence of this is that every thread will establish a new connection to the database.

4.9.5 PostgreSQL Optimisations

PostgreSQL connector implements a number of optimisations not present in clarion 5.5 ODBC driver.

4.9.5.1 Cursors

The JDBC driver will be setup to preferentially use cursors for SET/NEXT/PREVIOUS iterators. Cursors are valuable because otherwise the entire SQL result set is sent back to the client – costing both memory and performance. Cursors incur a cost though – it requires maintenance of an open transaction frame. The cost is minimal.

For ODBC Postgres – you achieved same result with setting Declare/Fetch. But Declare/Fetch is blind and dumb and applies for all SELECT statements. With clarion2java – GET() selects will not use cursors: only SET/NEXT/PREVIOUS will.

4.9.5.2 Limits

If BUFFER() is called on the file, then PostgreSQL clarion2java will implement “LIMIT X” on SELECT statements for SET/NEXT/PREVIOUS in addition to usage of cursors. Once the result set runs out – a new SQL statement is invisibly executed to continue on with the NEXT/PREVIOUS scanning.

This provides additional performance over cursors because it influences PostgreSQL cost based query planner to select a query that returns first number of records – which is typically only what you want. The first 20 or so records to fill the browse queue. Oracle has a concept to fulfill this – a query goal which defaults to 'FIRST 1000 records'. PostgreSQL has no equivalent, and it assumes the entire result set is the goal and cost planner plans accordingly.

This becomes quite obvious when doing queries for composite keys. See in a moment.

In order for this optimisation to work, the KEY used to scan must be a unique key.

4.9.5.3 Composite Key optimisations

If BUFFER() is called on the file, and key is a composite key and the first column of the Key is an integer, a further optimisation will be attempted.

Consider database schema involving supplier and stocked part. You may have:

```
supplier (
   id : long
   name : string
)
part (
    supplier_id :long
    partnumber : string
)
```

Now a typical index on part will be supplier_id and partnumber.

I may define SET thus

```
clear(part)
part.supplier_id=1
set(part.partkey,part.partkey)
loop
    next(part)
    . . .
```

Now without BUFFER, or with Clarion 5.5 ODBC driver, the resultant SQL will be something like this.

```
SELECT * FROM part WHERE supplier_id>1 OR (supplier_id=1 AND partnum>='')
```

But with clarion2java it will optimise to this:

```
SELECT * FROM part WHERE supplier_id=1 AND partnum>=''
```

And when the result set is exhausted, it will then seamlessly execute this to get the rest:

```
SELECT * FROM part WHERE supplier_id>=2
```

For postgres in particular, and when combined with LIMIT functionality above – can result in must faster SQL operations.

4.9.6 LOGOUT/Rollback/Commit support

At time of writing, login, begin and commit are currently not supported and are silently ignored. They can be implemented using Prop.SQL i.e.

```
logout:      File{PROP:SQL}='BEGIN'
commit:      File{PROP:SQL}='COMMIT'
rollback:    File{PROP:SQL}='ROLLBACK'
```

5 Known Limitations

5.1 Major limitations you may encounter

Includes() is lexical, not syntactic

In clarion, includes is a lexical function – not a syntactic one. The contents of file that includes references is blindly inserted into the file being compiled. For a number of reasons, including performance, I decided not to do this with clarion 2 java. Instead includes in clarion for 4 java has explicit syntactic meaning and there are limits where includes can be used. Specifically:

- In a data section to include global variables etc
- in a map or module section to include external functions
- in a code section to import procedure/method definitions

If this proves to be a significant limitation – can consider adding @java-lex-includes as a new statement to do lexical includes.

A notable exception to this is Clarion 6 syntax of include('file','section'). These are handled lexically, where the include inserted into the stream. Precisely same way clarion works.

Layering of components in a window

Appears as though windows have three 'layers' - and that items like buttons etc always overlay ontop of strings which in turn always overlay ontop of panels and boxes. Not yet implemented in java. Tried rapidly prototyping a solution - but it messed with other things - and not prepared right now to try and fix this one. Workaround involves explicitly ordering items in Window structure to the precise order you want.

5.2 Minor limitations/unusual features which you may encounter

Java unreachable code detection

Java enforces unreachable code tests at compile time, whereas clarion does not. This compiler will resolve most issues of this nature. One area that will not work properly is unreachable code analysis for loops where continue/break reference labels. Unreachable code analysis always assumes that continue/break break the immediate code block they reside in only.

Solution is to make sure when using label based cycle and break statements in clarion that procedures in question do not have unreachable code blocks. Problem is fixable in clarion2java compiler but it is a complicated problem.

STEP BY loops

cannot resolve complex loop by statements where the direction of the loop (+ve or -ve) is not obvious from lexical analysis of the statement. Fairly easy to fix at java runtime should need arise.

use variable substitutes

Clarion will permit object definition to supplant control variable number for certain keywords - i.e. following are both perfectly fine in clarion: select(harley_file_reader) and select(?harley_file_reader) where harley_file_reader is a QUEUE object in a LIST control.

Now could build this exception case into runtime java to resolve controls where ClarionObject instead of a int is provided - but do we really want to?

queues

Clarion allows get(QUEUE,+QUEUE) - what does this even mean? Does it mean find record that matches perfectly?

deep assignment on classes

:=: assignments involving classes will not do deep recursive copies - only shallow copies - because generally use it for cloning classes only. Could be easily fixed by tracking and flagging relevant classes with a 'mergable' interface. When implemented implements void mergeIn(ClarionGroup) and ClarionGroup mergeOut() to assist with extraction of symbols etc. Any other class in a marked class is also deeply marked.

Properties on types and classes?

Clarion allowed me to do the following : FileManager{prop:label} - in some dead code. What could that even possibly translate into?

5.3 Other observations and comments about Clarion Language

Visibility

Clarion does not respect private, protected, public modifiers. Either does clarion2java.