table of contents

## part one

Choice of user interface items 2

Case statement – ENBF model 2

Case statement – Railroad diagram 2

Data flow diagram 2

Data dictionary 2

## part two

Test data and expected output 3

Discussion of errors 3

Evidence of error checking 3

Evidence of testing 3

Evidence of debugging 3

## part three

Internal documentation 4

Evidence of readability of code 4

User documentation 4

part one

# choice of user interface items

// tab bar at the top

// status bar at the bottom

// omission of a “send message” button

# case statement - ebnf model

**case statement =**

CASE (<statement>|<variable>)

{WHEN (<constant>|<variable>)

{<statement>}}

END

**variable =**

(<capitalLetter>|<lowercaseLetter>){<capitalLetter>|<lowercaseLetter>|<digit>|.}

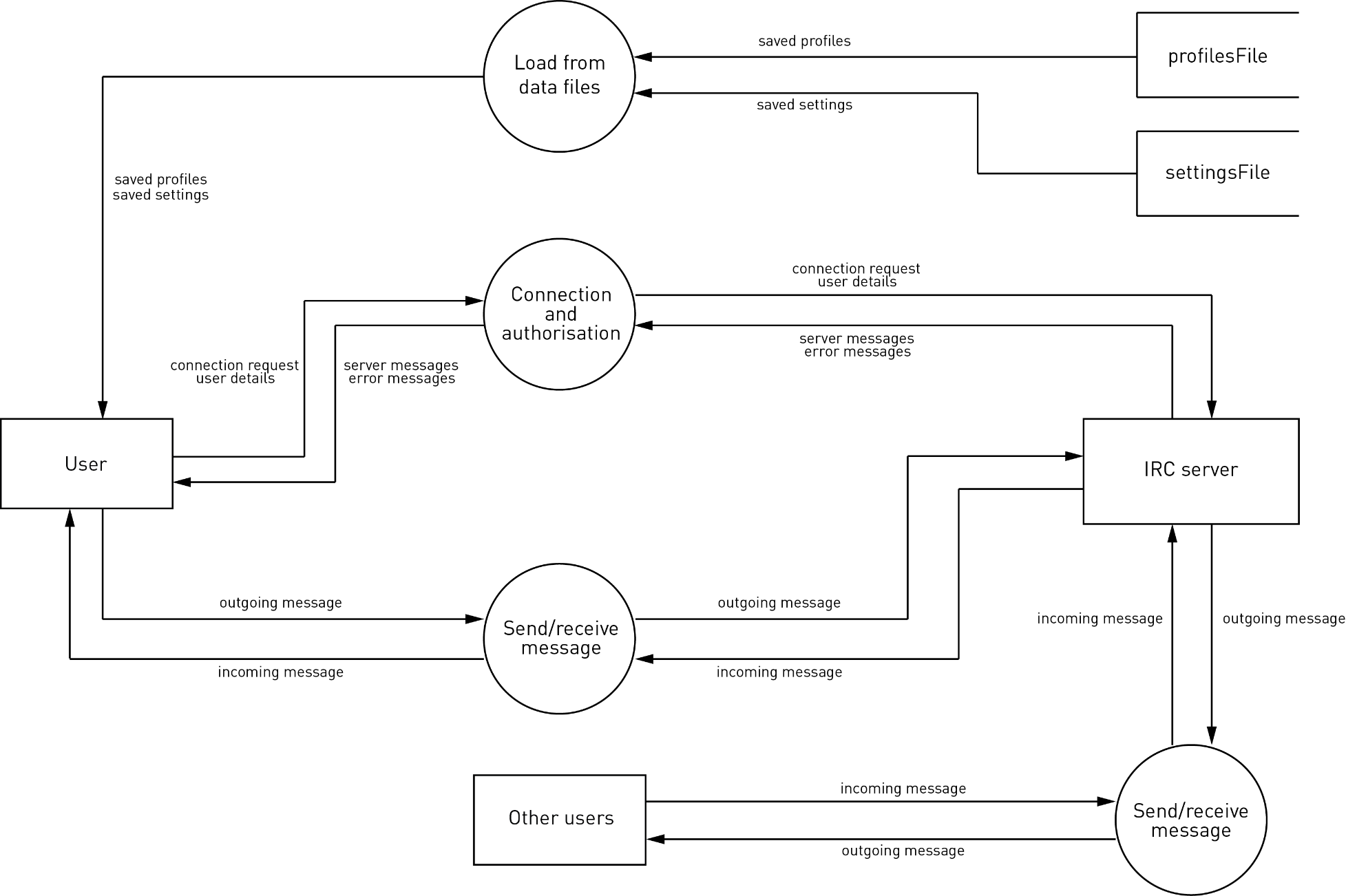
**constant =**

(<string>|{<digit>}|{<digit>}.{digit})

# case statement – railroad diagram

//

# data flow diagram



# data dictionary

//

part two

# test data and expected output

//

# discussion of errors

An error is any unwanted effect of a program that negatively affects its outcome or operation, often fatally. Errors that could occur are categorised into syntax errors, logic errors, and runtime errors.

Syntax errors are the result of incorrect use of properties of programming languages. Effectively, syntax errors occur when the strict rules and guides that a programming language dictates when using it are not followed. As such, syntax errors are generally the result of carelessness rather than that of a flawed algorithm or concept. In this way, syntax errors can occur absolutely anywhere; however, most syntax errors are detected by a compiler or interpreter and thus are usually very easy to resolve.

Logic errors are those that result from a flawed algorithm. Often, the programmer has not thought out the methods behind how the algorithm works thoroughly, and thus any mistake in how the algorithm operates would result in a dramatically different result. Unlike syntax errors, logic errors are rarely detected by a compiler or interpreter, as the computer has no way of knowing that the programmer’s intent was otherwise. Thus, logic errors are usually very difficult to locate and correct, often involving a length debugging process. Logic errors most commonly occur in algorithms where there is a high degree of complexity involved, as the programmer may have had become confused at some point during development and implemented an erroneous algorithm.

Runtime errors, as suggested by their name, occur during runtime. They are most often either the result of unexpected or non-validated input, resulting in illegal operations such as division by zero. Since they only become obvious at runtime, a compiler is normally unable to pre-empt a runtime error, and neither is an interpreter, before the line where the error occurs. Runtime errors can occur as side effects of logic errors and incorrect algorithms. They most often occur when one operation is dependent upon either user input or the output of another operation. Runtime errors are most commonly detected and corrected through the debugging techniques of use of breakpoints and single line stepping; these techniques allow for the checking of the values held within variables at any one point and thus allow the programmer to isolate the error to a particular section of code very quickly.

# evidence of error checking

// use of stubs, flags, debugging output statements

# evidence of testing

// testing of all pathways, desk check, and peer check (one module)

# evidence of debugging

// breakpoint, traces, single line stepping (one module)

part three

# internal documentation

// developer, intrinsic (give examples) (possibly provide a snippet?)

# evidence of readability of code

// provide evidence (code snippets)

# user documentation

// user, technical, installation, online help (give examples)