Module Interface Specification for VDisp

Emil Soleymani, Dr. Spencer Smith

May 18, 2022

1 Revision History

Date	**Version**	**Notes**
May 18, 2022	1.0	Initial Draft

2 Symbols, Abbreviations, and Acronyms

# TODO

Contents

3 Introduction

The following document details the Module Interface Specifications for *VDisp*

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at . . . .

4 Notation

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by VDisp.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of VDisp uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, VDisp uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Parameters Module Output Format Module Equation Module Control Module Display Module
Software-Decision Module	Plotting Module

Table 1: Module Hierarchy

6 MIS of Input Parameters Module

6.1 Module

InputFormat

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Types

InputData
Model
Foundation

6.3.2 Exported Access Programs

# TODO: Add exceptions

Name	In	Out	Exceptions
InputData	String	InputData	none

6.4 Semantics

6.4.1 State Variables

TODO: Copy these over from SRS once made

6.4.2 Environment Variables

inputFile: String[] *#f[i] is the ith string in text file f*

6.4.3 Assumptions

# TODO: comments in input file are denoted with a "#"

- Input file is in correct file format
- Input file will be fully parsed, with its corresponding values stored properly in an InputData instance before any attempts to access them

6.4.4 Access Routine Semantics

InputData(s):

- transition: The filename s is first associated with input file f, which is used to modify state variables using the following procedural specification:
    1. Read data sequentially from f to populate the state variables from *# TODO: insert requirement here*
    2. Calculate the derived quantities (all other state variables) as follows:
        - *# TODO: list derived values*
        - *e.x. model:Model is derived from modelOutput:Int*
    3. *# TODO: Come up with constraints on variables (e.x* $nodalPoints > 0$*,* $0.5 \leq poissoinsRatio \leq 1$*) then make local function verifyParams()*
- output: $out :=$ inputData:InputData where inputData has parameters specified from file f
- exception: $exc :=$ a file named s cannot be found OR the format of file f is incorrect $\Rightarrow$ FileError

6.4.5 Local Functions

validFile(S) : $String \rightarrow \mathbb{B}$

- output: $out := true$ iff file S is in valid format
- exception: $exc :=$ a file named s cannot be found OR the format of file f is incorrect $\Rightarrow$ FileError

verifyParams($inputData$) : InputData $\rightarrow \mathbb{B}$

- output: $out :=$ none
- exception: $exc :=$

$$\begin{aligned} \neg(inputData.nodalPoints > 0) &\Rightarrow \text{badNodalPoints} \\ \neg(0 \leq inputData.modelOption \leq 4) &\Rightarrow \text{badModelOption} \\ &\textit{etc...} \end{aligned}$$

6.5 Considerations

The value of each state variable can be accessed through its name (getter). An access program is available for each state variable. There are no setters for the state variables, since the values will be set *(and checked)* by the InputData constructor and not changed for the life of the InputData instance.

7 MIS of Control Module

7.1 Module

main

7.2 Uses

InputFormat, OutputFormat

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

7.4 Semantics

7.4.1 State Variables

None.

7.4.2 Access Routine Semantics

main():

- transition: Modify state of InputFormat module and the environment variables for the *Plot?* and OutputFormat modules by following these steps:
  - Get (inputFile:String) and (outputFile:string) from user
  - *# TODO: List steps*

8 MIS of Output Format Module

8.1 Module

OutputFormat

8.2 Uses

InputFormat, Equation # *TODO: Not specified yet*

8.3 Syntax

8.3.1 Exported Types

OutputData

8.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
OutputData	String	OutputData	FileError
writeOutput	OutputData,String, Function[]	None	TODO
writeHeader	OutputData,String	None	TODO
....	....	....	....
performWriteModel Output	OutputData,String	None	TODO
....	....	....	....

8.4 Semantics

8.4.1 State Variables

inputData:InputData

8.4.2 Environment Variables

outputFile:string

8.4.3 Access Routine Semantics

OutputData(s):

- transition: this.inputData := InputData(s)
- output: *out* := this

writeOutput(outputData, s, funcs):

- transition: The filename s is first associated with output file *out*, which is used to write state variables using the following procedural specification:

  1. Clear any existing contents of file *out*
  2. For each function f in funcs
     - Call function f, passing in outputData as argument
     - Write return value of f(outputData) to file *out*

writeHeader(outputData, s):

- transition: The filename s is first associated with output file *out*, which is used to write state variables using the following procedural specification:

  1. Append "Title: "$\|$outputData.inputData.problemName to file *out*
  2. Append "Nodal Points: "$\|$outputData.inputData.nodalPoints to file *out*
  3. Append "Base Nodal Point Index: "$\|$outputData.inputData.bottomPointIndex to file *out*
  4. Append "Number of Different Soil Layers: "$\|$outputData.inputData.soilLayers to file *out*
  5. Append "Increment Depth: "$\|$outputData.inputData.dx to file *out*

# TODO: Specify rest of the functions

8.4.4 Local Functions

None.

8.5 Considerations

$x\|y$ denotes the concatenation of strings x and y

9 MIS of Module Name

9.1 Module

9.2 Uses

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
accessProg	-	-	-

9.4 Semantics

9.4.1 State Variables

Not all modules will have state variables. State variables give the module a memory.

9.4.2 Environment Variables

This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc.

9.4.3 Assumptions

Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate.

9.4.4 Access Routine Semantics

accessProg():

- transition: if appropriate
- output:
- exception:

A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module.

Modules rarely have both a transition and an output. In most cases you will have one or the other.

9.4.5 Local Functions

As appropriate: These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope.

10 Appendix

Table 2: Possible Exceptions

Message ID	**Error Message**
FileError	Error: File not found
badNodalPoints	Error: Number of nodal points must be > 0
...	...
warnDX	Error: It is recommended that $0.01 \leq dx \leq 3$
...	...
TODO	TODO