Module Interface Specification for Two and Three Dimensional Dynamic Model of Soil-Water-Structure Interaction

Prepared by:
Brandon Karchewski (karcheba@mcmaster.ca)
Ph.D. Candidate
Department of Civil Engineering

Prepared for:
Dr. Spencer Smith
CES 741 - Development of Scientific Computing Software
Department of Computational Engineering and Science

McMaster University
Hamilton, Ontario, Canada

Ver. DynSWS-MIS-1.0
April 14, 2012

Brandon Karchewski
McMaster University - Dept. of Civil Engineering

Table of Contents

1 Introduction

This section introduces the MIS for DynSWS. Section 1.1 gives the raison d'être of this document. Section 1.2 provides some insight into the transition from the high-level design in the MG (see [1]) to the low-level syntax or interface specification in the MIS. Section 1.3 specifies the scope, to the extent that it differs from that of the SRS (see [2]) and the MG (see [1]). Section 1.4 lists and describes the intended audience of this document.

1.1 Purpose

This purpose of this MIS is to specify the interface syntax for each of the leaf modules presented in the MG for DynSWS. The reason for a document specifying the interface is to separate the concerns of design and implementation for each module. Specifying the syntax independent of the implementation also enables parallelization of the implementation task as the lines of communication between modules are clearly drawn.

1.2 Bridge Between Design and Specification

The MG for DynSWS provides the high-level design of the software product including the secret, service, and (optionally) prefix for each module. Also included in the MG are various examinations of the relationship between modules such as the module hierarchy and the uses hierarchy. The next step in the process is to define the syntax specific to each leaf module; that is, the lowest level modules that will actually be implemented (or used from an external source) in the DynSWS software product. This syntax specification is, as mentioned previously, the topic of this MIS.

1.3 Scope

The scope of the interface specification of DynSWS presented in this MIS is that same as that of the SRS and the MG. That is, this document contains the interface specification for all of the leaf modules in the design of the software product.

1.4 Intended Audience

The three main groups that the MIS is intended for use by are:

UG1. Developers. Users in this group are involved in the actual implementation of the requirements of DynSWS. This will certainly include the author, but may include others in the future if the software product proves useful and the functionality continues to be extended over time. This group can use the MIS as a reference to the low-level interface of DynSWS, which details the syntax and behaviour of each module. Users from this group should follow the interfaces presented herein when modifying the implementation of the modules. If users from this group add to the module hierarchy, they must also update the MIS to reflect these additions.

UG2. Maintainers. Users in this group maintain the software product over time. This may include activities such as performing tests, fixing bugs, and reorganizing the module hierarchy to reflect design modifications. Again, this will initially be just the author, but in the future may include others. If the interface to a module is modified by users in this group, changes should be documented in the MIS.

UG3. Reviewers. Users in this group have the task of ensuring that DynSWS meets all requirements and that the results produced by the software product are correct (insofar as correctness can be determined). This includes the author, but also the author's supervisory committee as they will be responsible for verifying the correctness and accuracy of the model contained in DynSWS. The MIS will be useful for this group in understanding the syntax of each module (inputs, outputs, exceptions) so that it can be reviewed in a systematic manner.

It should be noted that the MIS is not necessarily intended for end user of the software product. The MIS presents the low-level syntax of the implementation without going into detail on the requirements, the design, or the implementation. Readers interested in the requirements specification and the module guide (high-level design) for DynSWS should see references [2] and [1], respectively.

2 Module Interface Specification

2.1.1 File Reading and Writing

The facilities of this module are expected to be provided by the programming language and/or the operating system. However, this section contains limited information for the purpose of documenting other modules that use the File Reading and Writing module.

Exported Types

fileRefT := reference/handle to a file as defined by the programming language

2.1.2.1 Storage Access

The facilities of this module are expected to be provided by the programming language and/or the operating system.

2.1.2.2 Integer Operations

The facilities of this module are expected to be provided by the programming language and/or the operating system.

2.1.2.3 Floating Point Operations

The facilities of this module are expected to be provided by the programming language and/or the operating system.

2.1.2.4 Memory Access

The facilities of this module are expected to be provided by the programming language and/or the operating system. However, a state variable representing available memory is documented here for the purpose of documenting dynamic memory allocation in other modules.

Exported Functions

Table 2.1: Exported function interfaces for Memory Access module

Name	Input	Output	Exceptions
mem_getAvailMem		integer	
mem_allocMem	integer		

Semantics

State Variables

mem : integer

Access Routine Semantics

mem_getAvailMem():

output: *out* := amount of available memory stored in *mem*

exception: none

mem_allocMem(i):

transition: $mem := mem - i$

exception: none

2.2.1 Master Control

Uses

Modules:
Floating Point Operations
Input File Control
Integer Operations
Log Message Control
Output File Control
PDE Solver

Syntax

Exported Constants
N/A

Exported Functions

Table 2.2: Exported function interfaces for Master Control module

Name	Input	Output	Exceptions
dynSWS	string		

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

dynSWS(*fname*):

transition: Call ipt_setFileNames(*fname*,0)

Call log_setFileName(*fname*)
Call log_initLogFile()
Call ipt_loadDomain()
Call ipt_loadBoundary()
Call ipt_loadMaterials()
Call ipt_loadInitVector()
Call ipt_loadInitTensor()
Call ipt_loadKinBC()
Call pde_buildMassMatrix()
Call pde_buildStiffMatrix()
Call pde_buildDampMatrix()
Call pde_buildModMassMatrix
Call pde_buildLoadVector(0)
Call pde_initAcc()
Call opt_setFileNames(*fname*,0)
Call opt_printDisp()
Call opt_printVel()
Call opt_printAcc()
Call opt_printStress()
Call opt_printStrain()
Call opt_printStrainRate()
$\forall itime \in$ [1..dmn_numTimeSteps()]
{
    Call pde_buildLoadVector(*itime*)
    Call pde_incAcc()
    Call pde_incDisp()
    Call pde_incVel()
    Call pde_incStrain()
    Call pde_incStrainRate()
    Call pde_incStress()
    Call pde_updateAcc()
    Call pde_updateDisp()
    Call pde_updateVel()
    Call pde_updateStrain()
    Call pde_updateStrainRate()
    Call pde_updateStress()
    Call opt_setFileNames(*fname*,*itime*)
    Call opt_printDisp()
    Call opt_printVel()
    Call opt_printAcc()

```
        Call opt_printStress()
        Call opt_printStrain()
        Call opt_printStrainRate()
    }
    Call log_closeLogFile()
```

exception: none

Local Functions
N/A

Local Variables
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.1 Input File Control

Uses

Modules:
Body Force Reader
Boundary File Reader
Boundary Data
Domain File Reader
Field Data
Initial Tensor Field Reader
Initial Vector Field Reader
Kinematic BC Reader
Log Message Control
Log Messages
Material File Reader
Material Property Data
Natural BC Reader
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.3: Exported function interfaces for Input File Control module

Name	Input	Output	Exceptions
ipt_setFileNames	string, integer		
ipt_loadDomain			
ipt_loadBoundary			
ipt_loadMaterials			
ipt_loadInitVector			
ipt_loadInitTensor			
ipt_loadKinBC			
ipt_loadNatBC			
ipt_loadBodyAcc			

Semantics

State Variables

timeName : string
nodeName : string
elemName : string
bndName : string
mtlName : string
icDispName : string
icVelName : string
icStressName : string
icStrainName : string
icStrainRateName : string
fixName : string
tracName : string
bodyAccName : string

State Invariants

N/A

Assumptions

1. The function ipt_setFileNames() will always be called before other functions in this module.

Access Routine Semantics

ipt_setFileNames(*fname*,*i*):

transition: *timeName* := path to time step data file
nodeName := path to node data file
elemName := path to body element data file
bndName := path to boundary element data file
mtlName := path to material property data file
icDispName := path to initial displacement data file
icVelName := path to initial velocity data file
icStressName := path to initial stress data file
icStrainName := path to initial strain data file
icStrainRateName := path to initial strain rate data file
fixName := path to kinematic boundary condition data file
tracName := path to natural boundary condition data file for load step i
bodyAccName := path to body acceleration data file for load step i

exception: none

ipt_loadDomain():

transition: Call dmnrdr_initTimeFile(*timeName*)

Call dmn_initTime(dmnrdr_readTimeStep(), dmnrdr_readNumTimeSteps)
Call dmnrdr_closeTimeFile()
Call dmnrdr_initNodeFile(*nodeName*) *nnod* := dmnrdr_readNumNode()
Call dmn_initNode(*nnod*)
$\forall\ i \in [1..nnod]$
{
    coords := dmnrdr_readNode()
    $\forall j \in [1..NDIM]$ { dmn_setCoord(i,j,*coords*[j]) }
}
Call dmnrdr_closeNodeFile()
Call dmnrdr_initElemFile(*elemName*)
nel := dmnrdr_readNumElem()
Call dmn_initElem(*nel*)
$\forall\ i \in [1..nel]$
{
    connect := dmnrdr_readElem()
    $\forall j \in [1..NNODEL]$ { dmn_setConnect(i,j,*connect*[j]) }
}
Call dmnrdr_closeElemFile()

exception: none

ipt_loadBoundary():

transition: Call bndrdr_initFile(*bndName*)

nelb := bndrdr_readNumBoundElem()
Call bnd_init(*nelb*)
$\forall\ i \in [1..nelb]$
{
    connectBound := bndrdr_readBoundElem()
    $\forall j \in [1..NNODELB]$ { bnd_setConnect(i,j,*connectBound*[j])}
}
Call bndrdr_closeFile()

exception: none

ipt_loadMaterials():

transition: Call mtlrdr_initFile(*mtlName*)

nmtl := mtlrdr_readNumMatl()
Call mtl_init(*nmtl*)
$\forall\ i \in [1..nmtl]$
{
    E := mtlrdr_readEmod()
    mtl_setEmod(i,E)
    ν := mtlrdr_readPois()
    mtl_setPois(i,ν)
}
Call mtlrdr_closeFile()

exception: none

ipt_getInitVector():

transition: *nnod* := dmn_numNode()

Call icvrdr_initDispFile(*icDispName*)
$\forall\ i \in [1..nnod]$
{
    disp := icvrdr_readDisp()
    $\forall j \in$ [1..NDIM] { dmn_setDisp(i,j,*disp*[j]) }
}
Call icvrdr_closeDispFile()
Call icvrdr_initVelFile(*icVelName*)
$\forall\ i \in [1..nnod]$
{
    vel := icvrdr_readVel()
    $\forall j \in$ [1..NDIM] { dmn_setVel(i,j,*vel*[j]) }
}
Call icvrdr_closeVelFile()

exception: none

ipt_getInitTensor():

transition: *nel* := dmn_numElem()

Call ictrdr_initStressFile(*icStressName*)
$\forall\ i \in [1..nel]$
{
stress := ictrdr_readStress()
$\forall j \in [1..NTNS]$
{
($j = 1 \rightarrow$ dmn_setStressElem(i,11,*stress*[j])
| $j = 2 \rightarrow$ dmn_setStressElem(i,22,*stress*[j])
| $j = 3 \rightarrow$ dmn_setStressElem(i,33,*stress*[j])
| $j = 4 \rightarrow$ dmn_setStressElem(i,12,*stress*[j])
| $j = 5 \rightarrow$ dmn_setStressElem(i,23,*stress*[j])
| $j = 6 \rightarrow$ dmn_setStressElem(i,31,*stress*[j])
}
}
Call ictrdr_closeStressFile()
Call ictrdr_initStrainFile(*icStrainName*)
$\forall\ i \in [1..nel]$
{
strain := ictrdr_readStrain()
$\forall j \in [1..NTNS]$
{
($j = 1 \rightarrow$ dmn_setStrainElem(i,11,*strain*[j])
| $j = 2 \rightarrow$ dmn_setStrainElem(i,22,*strain*[j])
| $j = 3 \rightarrow$ dmn_setStrainElem(i,33,*strain*[j])
| $j = 4 \rightarrow$ dmn_setStrainElem(i,12,*strain*[j])
| $j = 5 \rightarrow$ dmn_setStrainElem(i,23,*strain*[j])
| $j = 6 \rightarrow$ dmn_setStrainElem(i,31,*strain*[j])
}
}
Call ictrdr_closeStrainFile()
Call ictrdr_initStrainRateFile(*icStrainRateName*)
$\forall\ i \in [1..nel]$
{
strainrate := ictrdr_readStrainRate()
$\forall j \in [1..NTNS]$
{
($j = 1 \rightarrow$ dmn_setStrainRateElem(i,11,*strainrate*[j])
| $j = 2 \rightarrow$ dmn_setStrainRateElem(i,22,*strainrate*[j])
| $j = 3 \rightarrow$ dmn_setStrainRateElem(i,33,*strainrate*[j])
| $j = 4 \rightarrow$ dmn_setStrainRateElem(i,12,*strainrate*[j])
| $j = 5 \rightarrow$ dmn_setStrainRateElem(i,23,*strainrate*[j])

| $j = 6 \rightarrow$ dmn_setStrainRateElem(i,31,*strainrate*[j])
}
}
Call ictrdr_closeStrainRateFile()

exception: none

inctrl_getKinBC():

transition: *nnod* := dmn_numNode()

Call kbcrdr_initFixFile(*fixName*)
$\forall\ i \in$ [1..*nnod*]
{
fix := kbcrdr_readFix()
$\forall j \in$ [1..NDIM] { dmn_setFix(i,j,*fix*[j]) }
}
Call kbcrdr_closeFixFile()

exception: none

inctrl_getNatBC():

transition: *nelb* := bnd_numBoundElem()

Call nbcrdr_initTracFile(*tracName*)
$\forall\ i \in$ [1..*nelb*]
{
trac := nbcrdr_readTrac()
$\forall j \in$ [1..NNODELB] { tns_setTrac(i,j,*trac*[j]) }
}
Call nbcrdr_closeTracFile()

exception: none

inctrl_getBodyAcc():

transition: *nnod* := dmn_numNode()

Call bfcrdr_initBodyAccFile(*bodyAccName*)
$\forall\ i \in [1..nnod]$
{
bodyAcc := bfcrdr_readBodyAcc()
$\forall j \in [1..NDIM]$ { dmn_setBodyAcc(i,j,*bodyAcc*[j]) }
}
Call bfcrdr_closeBodyAccFile()

exception: none

Local Functions
N/A

Local Variables

nnod : integer
coords : sequence [NDIM] of real
nel : integer
connect : sequence [NNODEL] of integer
nelb : integer
connectBound : sequence [NNODELB] of integer
nmtl : integer
E : real
ν : real
disp : sequence [NDIM] of real
vel : sequence [NDIM] of real
stress : sequence [NTNS] of real
strain : sequence [NTNS] of real
strainRate : sequence [NTNS] of real
fix : sequence [NDIM] of boolean
trac : sequence [NNODELB] of surfLoadT
bodyAcc : sequence [NDIM] of real

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.2 Domain File Reader

Uses

Modules:
File Reading and Writing
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.4: Exported function interfaces for Domain File Reader module

Name	Input	Output	Exceptions
dmnrdr_initTimeFile	string		EXIST
dmnrdr_readTimeStep		real	FORMT
dmnrdr_readNumTimeSteps		integer	FORMT
dmnrdr_closeTimeFile			
dmnrdr_initNodeFile	string		EXIST
dmnrdr_readNumNode		integer	FORMT
dmnrdr_readNode		sequence [NDIM] of real	FORMT
dmnrdr_closeNodeFile			
dmnrdr_initElemFile	string		EXIST
dmnrdr_readNumElem		integer	FORMT
dmnrdr_readElem		sequence [NNODEL] of integer	FORMT
dmnrdr_closeElemFile			

Semantics

State Variables

timefile : fileRefT
ndfile : fileRefT
elfile : fileRefT

State Invariants
N/A

Assumptions

1. The function dmnrdr_initNodeFile() will always be called before other functions containing Node in this module.
2. The node data file contains the number of node data entries corresponding to the number of nodes data entry.
3. The function dmnrdr_initElemFile() will always be called before other functions containing Elem in this module.
4. The element data file contains the number of element data entries corresponding to the number of elements data entry.

Access Routine Semantics

dmnrdr_initTimeFile(*fname*):

transition: *timefile* := fileRefT for time data file given by *fname*

exception: *exc* := (time data file does not exist → EXIST)

dmnrdr_readTimeStep():

output: *out* := real read from *timefile*

transition: Advance the reading position in *timefile*

exception: *exc* := (*timefile* does not contain time step in expected format → FORMT)

dmnrdr_readNumTimeSteps():

output: *out* := integer read from *timefile*

transition: Advance the reading position in *timefile*

exception: *exc* := (*timefile* does not contain number of time steps in expected format → FORMT)

dmnrdr_closeTimeFile():

transition: Close the file associated with *timefile*

exception: none

dmnrdr_initNodeFile(*fname*):

transition: *ndfile* := fileRefT for node data file given by *fname*

exception: *exc* := (node data file does not exist → EXIST)

dmnrdr_readNumNode():

output: *out* := integer read from *ndfile*

transition: Advance the reading position in *ndfile*

exception: *exc* := (*nodefile* does not contain number of nodes in expected format → FORMT)

dmnrdr_readNode():

output: *out* := sequence [NDIM] of real read from *ndfile*

transition: Advance the reading position in *ndfile*

exception: *exc* := (node coordinates not in expected format → FORMT)

dmnrdr_closeNodeFile():

transition: Close the file associated with *ndfile*

exception: none

dmnrdr_initElemFile(*fname*):

transition: *elfile* := fileRefT for element data file given by *fname*

exception: *exc* := (element data file does not exist → EXIST)

dmnrdr_readNumElem():

output: *out* := integer read from *elfile*

transition: Advance the reading position in *elfile*

exception: *exc* := (*elfile* does not contain number of elements in expected format → FORMT)

dmnrdr_readElem():

output: *out* := sequence [NNODEL] of integer read from *elfile*

transition: Advance the reading position in *elfile*

exception: *exc* := (element data is not in expected format → FORMT)

dmnrdr_closeElemFile():

transition: Close the file associated with *elfile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.3 Boundary File Reader

Uses

Modules:
File Reading and Writing
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.5: Exported function interfaces for Boundary File Reader module

Name	Input	Output	Exceptions
bndrdr_initFile	string		EXIST
bndrdr_readNumBoundElem		integer	FORMT
bndrdr_readBoundElem		sequence [NNODELB] of integer	FORMT
bndrdr_closeFile			

Semantics

State Variables

bndfile : fileRefT

State Invariants
N/A

Assumptions

1. The function bndrdr_initFile() will always be called before other functions in this module.

2. The boundary element data file contains the number of boundary element data entries corresponding to the number of boundary elements data entry.

Access Routine Semantics

bndrdr_initFile(*fname*):

transition: *bndfile* := fileRefT for boundary element data file given by *fname*

exception: *exc* := (boundary element data file does not exist → EXIST)

bndrdr_readNumBoundElem():

output: *out* := integer read from *bndfile*

transition: Advance the reading position in *bndfile*

exception: *exc* := (*bndfile* does not contain number of boundary elements in expected format → FORMT)

bndrdr_readBoundElem():

output: *out* := sequence [NNODELB] of integer read from *bndfile*

transition: Advance the reading position in *bndfile*

exception: *exc* := (boundary element data is not in expected format → FORMT)

bndrdr_closeFile():

transition: Close the file associated with *bndfile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.4 Material File Reader

Uses

Modules:
File Reading and Writing
Log Message Control
Log Messages

Syntax

Exported Constants
N/A

Exported Functions

Table 2.6: Exported function interfaces for Material File Reader module

Name	Input	Output	Exceptions
mtlrdr_initFile	string		EXIST
mtlrdr_readNumMatl		integer	FORMT
mtlrdr_readEmod		real	FORMT
mtlrdr_readPois		real	FORMT
mtlrdr_closeFile			

Semantics

State Variables

mtlfile : fileRefT

State Invariants
N/A

Assumptions

1. The function mtlrdr_initFile() will always be called before other functions in this module.

2. The material property data file contains the number of material property data entries corresponding to the number of materials data entry.

Access Routine Semantics

mtlrdr_initFile(*fname*):

transition: *mtlfile* := fileRefT for material property data file given by *fname*

exception: *exc* := (material property data file does not exist → EXIST)

mtlrdr_readNumMatl():

output: *out* := integer read from *mtlfile*

transition: Advance the reading position in *mtlfile*

exception: *exc* := (*mtlfile* does not contain number of materials in expected format → FORMT)

mtlrdr_readEmod():

output: *out* := real read from *mtlfile*

transition: Advance the reading position in *mtlfile*

exception: *exc* := (elastic modulus data is not in expected format → FORMT)

mtlrdr_readPois():

output: *out* := real read from *mtlfile*

transition: Advance the reading position in *mtlfile*

exception: *exc* := (Poisson's ratio data is not in expected format → FORMT)

mtlrdr_closeFile():

transition: Close the file associated with *mtlfile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.5.1 Initial Vector Field Reader

Uses

Modules:
File Reading and Writing
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.7: Exported function interfaces for Initial Vector Field Reader module

Name	Input	Output	Exceptions
icvrdr_initDispFile	string		EXIST
icvrdr_readDisp		sequence [NDIM] of real	FORMT
icvrdr_closeDispFile			
icvrdr_initVelFile	string		EXIST
icvrdr_readVel		sequence [NDIM] of real	FORMT
icvrdr_closeVelFile			

Semantics

State Variables

dispfile : fileRefT
velfile : fileRefT

State Invariants
N/A

Assumptions

1. The function icvrdr_initDispFile() will always be called before other functions containing Disp in this module.

2. The initial displacement data file contains the same number of data entries as the number of geometry nodes.

3. The function icvrdr_initVelFile() will always be called before other functions containing Vel in this module.

4. The initial displacement data file contains the same number of data entries as the number of geometry nodes.

Access Routine Semantics

icvrdr_initDispFile(*fname*):

transition: *dispfile* := fileRefT for initial displacement field data file given by *fname*

exception: *exc* := (initial displacement field data file does not exist → EXIST)

icvrdr_readDisp(i):

output: *out* := sequence [NDIM] of real read from *dispfile*

transition: Advance the reading position in *dispfile*

exception: *exc* := (displacement data is not in expected format → FORMT)

icvrdr_closeDispFile():

transition: Close the file associated with *dispfile*

exception: none

icvrdr_initVelFile(*fname*):

transition: *velfile* := fileRefT for initial velocity field data file given by *fname*

exception: *exc* := (initial velocity field data file does not exist → EXIST)

icvrdr_readVel(i):

output: *out* := sequence [NDIM] of real read from *velfile*

transition: Advance the reading position in *velfile*

exception: *exc* := (velocity data is not in expected format → FORMT)

icvrdr_closeVelFile():

transition: Close the file associated with *velfile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.5.2 Initial Tensor Field Reader

Uses

Modules:
File Reading and Writing
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.8: Exported function interfaces for Initial Tensor Field Reader module

Name	Input	Output	Exceptions
ictrdr_initStressFile	string		EXIST
ictrdr_readStress		sequence [NTNS] of real	FORMT
ictrdr_closeStressFile			
ictrdr_initStrainFile	string		EXIST
ictrdr_readStrain		sequence [NTNS] of real	FORMT
ictrdr_closeStrainFile			
ictrdr_initStrainRateFile	string		EXIST
ictrdr_readStrainRate		sequence [NTNS] of real	FORMT
ictrdr_closeStrainRateFile			

Semantics

State Variables

stressFile : fileRefT
strainFile : fileRefT
strainRateFile : fileRefT

State Invariants
N/A

Assumptions

1. The function ictrdr_initStressFile() will always be called before other functions containing Stress in this module.
2. The initial stress data file contains the same number of data entries as the number of body elements.
3. The function ictrdr_initStrainFile() will always be called before other functions containing Strain in this module.
4. The initial strain data file contains the same number of data entries as the number of body elements.
5. The function ictrdr_initStrainRateFile() will always be called before other functions containing StrainRate in this module.
6. The initial strain rate data file contains the same number of data entries as the number of body elements.

Access Routine Semantics

ictrdr_initStressFile(*fname*):

transition: *stressFile* := fileRefT for initial stress field data file given by *fname*

exception: *exc* := (initial stress field data file does not exist → EXIST)

ictrdr_readStress():

output: *out* := sequence [NTNS] of real read from *stressFile*

transition: Advance the reading position in *stressFile*

exception: *exc* := (stress data is not in expected format → FORMT)

ictrdr_closeStressFile():

transition: Close the file associated with *stressFile*

exception: none

ictrdr_initStrainFile(*fname*):

transition: *strainFile* := fileRefT for initial strain field data file given by *fname*

exception: *exc* := (initial strain field data file does not exist → EXIST)

ictrdr_readStrain(i):

output: *out* := sequence [NTNS] of real read from *strainFile*

transition: Advance the reading position in *strainFile*

exception: *exc* := (strain data is not in expected format → FORMT)

ictrdr_closeStrainFile():

transition: Close the file associated with *strainFile*

exception: none

ictrdr_initStrainRateFile(*fname*):

transition: *strainRateFile* := fileRefT for initial strain rate field data file given by *fname*

exception: *exc* := (initial strain rate field data file does not exist → EXIST)

ictrdr_readStrainRate(i):

output: *out* := sequence [NTNS] of real read from *strainRateFile*

transition: Advance the reading position in *strainRateFile*

exception: *exc* := (strain rate data is not in expected format → FORMT)

ictrdr_closeStrainRateFile():

transition: Close the file associated with *strainRateFile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.6.1 Kinematic BC Reader

Uses

Modules:
File Reading and Writing
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.9: Exported function interfaces for Kinematic BC Reader module

Name	Input	Output	Exceptions
kbcrdr_initFixFile	string		EXIST
kbcrdr_readFix		sequence [NDIM] of boolean	FORMT
kbcrdr_closeFixFile			

Semantics

State Variables

fixfile : fileRefT

State Invariants
N/A

Assumptions

1. The function kbcrdr_initFixFile() will always be called before other functions containing Fix in this module.
2. The kinematic boundary condition data file contains the same number of data entries as the number of nodes.

Access Routine Semantics

kbcrdr_initFixFile(*fname*):

transition: *fixfile* := fileRefT for kinematic boundary condition data file given by *fname*

exception: *exc* := (kinematic boundary condition data file does not exist → EXIST)

kbcrdr_readFix():

output: *out* := sequence [NDIM] of boolean read from *fixfile*

transition: Advance the reading position in *fixfile*

exception: *exc* := (kinematic boundary condition data is not in expected format → FORMT)

kbcrdr_closeFixFile():

transition: Close the file associated with *fixfile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.6.2 Natural BC Reader

Uses

Modules:
Boundary Data
File Reading and Writing
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.10: Exported function interfaces for Natural BC Reader module

Name	Input	Output	Exceptions
nbcrdr_initTracFile	string		EXIST
nbcrdr_readTrac		sequence [NNODELB] of surfLoadT	FORMT
nbcrdr_closeTracFile			

Semantics

State Variables

tracfile : fileRefT

State Invariants
N/A

Assumptions

1. The function nbcrdr_initTracFile() will always be called before other functions containing Trac in this module.
2. The boundary traction data file contains the same number of data entries as the number of boundary elements.

Access Routine Semantics

nbcrdr_initTracFile(*fname*):

transition: *tracfile* := fileRefT for boundary traction data file given by *fname*

exception: *exc* := (boundary traction data file does not exist → EXIST)

nbcrdr_readTrac(i):

output: *out* := sequence [NNODELB] of surfLoadT read from *tracfile*

transition: Advance the reading position in *tracfile*

exception: *exc* := (traction data is not in expected format → FORMT)

nbcrdr_closeTracFile():

transition: Close the file associated with *tracfile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.2.7 Body Force Reader

Uses

Modules:
File Reading and Writing
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.11: Exported function interfaces for Body Force Reader module

Name	Input	Output	Exceptions
bfcrdr_initBodyAccFile	string		EXIST
bfcrdr_readBodyAcc		sequence [NDIM] of real	FORMT
bfcrdr_closeBodyAccFile			

Semantics

State Variables

bodyaccfile : fileRefT

State Invariants
N/A

Assumptions

1. The function bfcrdr_initBodyAccFile() will always be called before other functions containing BodyAcc in this module.
2. The applied body acceleration data file contains the same number of data entries as the number of geometry nodes.

Access Routine Semantics

bfcrdr_initBodyAccFile(*fname*):

transition: *bodyaccfile* := fileRefT for applied body acceleration data file given by *fname*

exception: *exc* := (applied body acceleration data file does not exist → EXIST)

bfcrdr_readBodyAcc(*i*):

output: *out* := sequence [NDIM] of real read from *bodyaccfile*

transition: Advance the reading position in *bodyaccfile*

exception: *exc* := (acceleration data is not in expected format → FORMT)

bfcrdr_closeBodyAccFile():

transition: Close the file associated with *bodyaccfile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.3.1 Output File Control

Uses

Modules:
Boundary Data
Field Data
Log Message Control
Log Messages
Material Property Data
System Constants
Tensor Field Writer
Vector Field Writer

Syntax

Exported Constants
N/A

Exported Functions

Table 2.12: Exported function interfaces for Output File Control module

Name	Input	Output	Exceptions
opt_setFileNames	string, integer		
opt_printDisp			
opt_printVel			
opt_printAcc			
opt_printStress			
opt_printStrain			
opt_printStrainRate			

Semantics

State Variables

dispName : string
velName : string
accName : string
stressNodeName : string
stressElemName : string
strainNodeName : string
strainElemName : string
strainRateNodeName : string
strainRateElemName : string

State Invariants

N/A

Assumptions

1. The function opt_setFileNames() will always be called before other functions in this module.

Access Routine Semantics

opt_setFileNames(*fname*,*i*):

transition: *dispName* := path to displacement data file for time step i
velName := path to velocity data file for time step i
accName := path to acceleration data file for time step i
stressNodeName := path to data file for stresses at nodes for time step i
stressElemName := path to data file for stresses in elements for time step i
strainNodeName := path to data file for strains at nodes for time step i
strainElemName := path to data file for strains in elements for time step i
strainRateNodeName := path to data file for strain rates at nodes for time step i
strainRateElemName := path to data file for strain rates in elements for time step i

exception: none

opt_printDisp():

transition: *nnod* := dmn_numNode()

Call vecwtr_initDispFile(*dispName*)
$\forall\ i \in [1..nnod]$
{
$\forall j \in [1..\text{NDIM}]$ { *disp*[j] := dmn_getDisp(i,j) }
vecwtr_writeDisp(i,*disp*)
}
Call vecwtr_closeDispFile()

exception: none

opt_printVel():

transition: *nnod* := dmn_numNode()

Call vecwtr_initVelFile(*velName*)
$\forall\ i \in [1..nnod]$
{
$\forall j \in [1..\text{NDIM}]$ { *vel*[*j*] := dmn_getVel(*i*,*j*) }
vecwtr_writeVel(*i*,*vel*)
}
Call vecwtr_closeVelFile()

exception: none

opt_printAcc():

transition: *nnod* := dmn_numNode()

Call vecwtr_initAccFile(*accName*)
$\forall\ i \in [1..nnod]$
{
$\forall j \in [1..\text{NDIM}]$ { *acc*[*j*] := dmn_getAcc(*i*,*j*) }
vecwtr_writeAcc(*i*,*acc*)
}
Call vecwtr_closeAccFile()

exception: none

opt_printStress():

transition: *nnod* := dmn_numNode()

Call tnswtr_initStressFile(*stressNodeName*)
$\forall\ i \in [1..nnod]$
{
  $\forall j \in [1..\text{NTNS}]$
  {
    $(j = 1 \rightarrow stress[j] :=$ dmn_getStressNode(i,11)
    $\mid j = 2 \rightarrow stress[j] :=$ dmn_getStressNode(i,22)
    $\mid j = 3 \rightarrow stress[j] :=$ dmn_getStressNode(i,33)
    $\mid j = 4 \rightarrow stress[j] :=$ dmn_getStressNode(i,12)
    $\mid j = 5 \rightarrow stress[j] :=$ dmn_getStressNode(i,23)
    $\mid j = 6 \rightarrow stress[j] :=$ dmn_getStressNode(i,31))
  }
  tnswtr_writeStress(i,*stress*)
}
Call tnswtr_closeStressFile()
Call tnswtr_initStressFile(*stressElemName*)
$\forall\ i \in [1..nel]$
{
  $\forall j \in [1..\text{NTNS}]$
  {
    $(j = 1 \rightarrow stress[j] :=$ dmn_getStressElem(i,11)
    $\mid j = 2 \rightarrow stress[j] :=$ dmn_getStressElem(i,22)
    $\mid j = 3 \rightarrow stress[j] :=$ dmn_getStressElem(i,33)
    $\mid j = 4 \rightarrow stress[j] :=$ dmn_getStressElem(i,12)
    $\mid j = 5 \rightarrow stress[j] :=$ dmn_getStressElem(i,23)
    $\mid j = 6 \rightarrow stress[j] :=$ dmn_getStressElem(i,31))
  }
  tnswtr_writeStress(i,*stress*)
}
Call tnswtr_closeStressFile()

exception: none

opt_printStrain():

transition: *nnod* := dmn_numNode()

Call tnswtr_initStrainFile(*strainNodeName*)
$\forall\ i \in [1..nnod]$
{
  $\forall j \in [1..\text{NTNS}]$
  {
    $(j = 1 \rightarrow strain[j] :=$ dmn_getStrainNode(i,11)
    $\mid j = 2 \rightarrow strain[j] :=$ dmn_getStrainNode(i,22)
    $\mid j = 3 \rightarrow strain[j] :=$ dmn_getStrainNode(i,33)
    $\mid j = 4 \rightarrow strain[j] :=$ dmn_getStrainNode(i,12)
    $\mid j = 5 \rightarrow strain[j] :=$ dmn_getStrainNode(i,23)
    $\mid j = 6 \rightarrow strain[j] :=$ dmn_getStrainNode(i,31))
  }
  tnswtr_writeStrain(i,*strain*)
}
Call tnswtr_closeStrainFile()
Call tnswtr_initStrainFile(*strainElemName*)
$\forall\ i \in [1..nel]$
{
  $\forall j \in [1..\text{NTNS}]$
  {
    $(j = 1 \rightarrow strain[j] :=$ dmn_getStrainElem(i,11)
    $\mid j = 2 \rightarrow strain[j] :=$ dmn_getStrainElem(i,22)
    $\mid j = 3 \rightarrow strain[j] :=$ dmn_getStrainElem(i,33)
    $\mid j = 4 \rightarrow strain[j] :=$ dmn_getStrainElem(i,12)
    $\mid j = 5 \rightarrow strain[j] :=$ dmn_getStrainElem(i,23)
    $\mid j = 6 \rightarrow strain[j] :=$ dmn_getStrainElem(i,31))
  }
  tnswtr_writeStrain(i,*strain*)
}
Call tnswtr_closeStrainFile()

exception: none

opt_printStrainRate():

transition: *nnod* := dmn_numNode()

Call tnswtr_initStrainRateFile(*strainRateNodeName*)
$\forall\ i \in [1..nnod]$
{
  $\forall j \in [1..\text{NTNS}]$
  {
    ($j = 1 \rightarrow$ *strainRate*[j] := dmn_getStrainRateNode(i,11)
    | $j = 2 \rightarrow$ *strainRate*[j] := dmn_getStrainRateNode(i,22)
    | $j = 3 \rightarrow$ *strainRate*[j] := dmn_getStrainRateNode(i,33)
    | $j = 4 \rightarrow$ *strainRate*[j] := dmn_getStrainRateNode(i,12)
    | $j = 5 \rightarrow$ *strainRate*[j] := dmn_getStrainRateNode(i,23)
    | $j = 6 \rightarrow$ *strainRate*[j] := dmn_getStrainRateNode(i,31))
  }
  tnswtr_writeStrainRate(i,*strainRate*)
}
Call tnswtr_closeStrainRateFile()
Call tnswtr_initStrainRateFile(*strainRateElemName*)
$\forall\ i \in [1..nel]$
{
  $\forall j \in [1..\text{NTNS}]$
  {
    ($j = 1 \rightarrow$ *strainRate*[j] := dmn_getStrainRateElem(i,11)
    | $j = 2 \rightarrow$ *strainRate*[j] := dmn_getStrainRateElem(i,22)
    | $j = 3 \rightarrow$ *strainRate*[j] := dmn_getStrainRateElem(i,33)
    | $j = 4 \rightarrow$ *strainRate*[j] := dmn_getStrainRateElem(i,12)
    | $j = 5 \rightarrow$ *strainRate*[j] := dmn_getStrainRateElem(i,23)
    | $j = 6 \rightarrow$ *strainRate*[j] := dmn_getStrainRateElem(i,31))
  }
  tnswtr_writeStrainRate(i,*strainRate*)
}
Call tnswtr_closeStrainRateFile()

exception: none

Local Functions
N/A

Local Variables

nnod : integer
nel : integer
disp : sequence [NDIM] of real
vel : sequence [NDIM] of real
acc : sequence [NDIM] of real
stress : sequence [NTNS] of real
strain : sequence [NTNS] of real
strainRate : sequence [NTNS] of real

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.3.2 Vector Field Writer

Uses

Modules:
File Reading and Writing
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.13: Exported function interfaces for Vector Field Writer module

Name	Input	Output	Exceptions
vecwtr_initDispFile	string		
vecwtr_writeDisp	integer, sequence [NDIM] of real		
vecwtr_closeDispFile			
vecwtr_initVelFile	string		
vecwtr_writeVel	integer, sequence [NDIM] of real		
vecwtr_closeVelFile			
vecwtr_initAccFile	string		
vecwtr_writeAcc	integer, sequence [NDIM] of real		
vecwtr_closeAccFile			

Semantics

State Variables
dispfile : fileRefT
velfile : fileRefT
accfile : fileRefT

State Invariants
N/A

Assumptions

1. The function vecwtr_initDispFile() will always be called before other functions containing Disp in this module.

2. The function vecwtr_initVelFile() will always be called before other functions containing Vel in this module.

3. The function vecwtr_initAccFile() will always be called before other functions containing Acc in this module.

Access Routine Semantics

vecwtr_initDispFile(*fname*):

transition: *dispfile* := fileRefT for displacement field output file given by *fname*

exception: none

vecwtr_writeDisp(i,d):

transition: Write node number i to *dispfile*
$\forall j \in$ [1..NDIM] { Write displacement $d[j]$ to *dispfile* }
Advance the writing position in *dispfile*

exception: none

vecwtr_closeDispFile():

transition: Close the file associated with *dispfile*

exception: none

vecwtr_initVelFile(*fname*):

transition: *velfile* := fileRefT for velocity field output file given by *fname*

exception: none

vecwtr_writeVel(i,v):

transition: Write node number i to *velfile*
$\forall j \in$ [1..NDIM] { Write velocity $v[j]$ to *velfile* }
Advance the writing position in *velfile*

exception: none

vecwtr_closeVelFile():

transition: Close the file associated with *velfile*

exception: none

vecwtr_initAccFile(*fname*):

transition: *accfile* := fileRefT for acceleration field output file given by *fname*

exception: none

vecwtr_writeAcc(i,a):

transition: Write node number i to *accfile*
$\forall j \in [1..\text{NDIM}]$ { Write acceleration $a[j]$ to *accfile* }
Advance the writing position in *accfile*

exception: none

vecwtr_closeAccFile():

transition: Close the file associated with *accfile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.3.3 Tensor Field Writer

Uses

Modules:
File Reading and Writing
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.14: Exported function interfaces for Tensor Field Writer module

Name	Input	Output	Exceptions
tnswtr_initStressFile	string		
tnswtr_writeStress	integer, sequence [NTNS] of real		
tnswtr_closeStressFile			
tnswtr_initStrainFile	string		
tnswtr_writeStrain	integer, sequence [NTNS] of real		
tnswtr_closeStrainFile			
tnswtr_initStrainRateFile	string		
tnswtr_writeStrainRate	integer, sequence [NTNS] of real		
tnswtr_closeStrainRateFile			

Semantics

State Variables

stressFile : fileRefT
strainFile : fileRefT
strainRateFile : fileRefT

State Invariants
N/A

Assumptions

1. The function tnswtr_initStressFile() will always be called before other functions containing Stress in this module.

2. The function tnswtr_initStrainFile() will always be called before other functions containing Strain in this module.

3. The function tnswtr_initStrainRateFile() will always be called before other functions containing StrainRate in this module.

Access Routine Semantics

tnswtr_initStressFile(*fname*):

transition: *stressFile* := fileRefT for stress field output file given by *fname*

exception: none

tnswtr_writeStress(i,s):

transition: Write element number i to *stressFile*
$\forall j \in$ [1..NTNS] { Write stress $s[j]$ to *stressFile* }
Advance the writing position in *stressFile*

exception: none

tnswtr_closeStressFile():

transition: Close the file associated with *stressFile*

exception: none

tnswtr_initStrainFile(*fname*):

transition: *strainFile* := fileRefT for strain field output file given by *fname*

exception: none

tnswtr_writeStrain(i,s):

transition: Write element number i to *strainFile*
$\forall j \in$ [1..NTNS] { Write strain $s[j]$ to *strainFile* }
Advance the writing position in *strainFile*

exception: none

tnswtr_closeStrainFile():

transition: Close the file associated with *strainFile*

exception: none

tnswtr_initStrainRateFile(*fname*):

transition: *strainRateFile* := fileRefT for strain rate field output file given by *fname*

exception: none

tnswtr_writeStrainRate(i,s):

transition: Write element number i to *strainRateFile*
$\forall j \in [1..NTNS]$ { Write strain rate $s[j]$ to *strainRateFile* }
Advance the writing position in *strainRateFile*

exception: none

tnswtr_closeStrainRateFile():

transition: Close the file associated with *strainRateFile*

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.4.1 Log Message Control

Uses

Modules:
File Reading and Writing
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Functions

Table 2.15: Exported function interfaces for Log Message Control module

Name	Input	Output	Exceptions
log_setFileName	string		
log_getFileName		string	
log_initLogFile			
log_printLogMsg	messageT, senderT		
log_closeLogFile			

Semantics

State Variables

logName : string
logFile : fileRefT

State Invariants
N/A

Assumptions

1. The function log_setFileName() will always be called before the function log_initLogFile().
2. The function log_initLogFile() will always be called before the function log_printLogMsg().

Access Routine Semantics

log_setFileName(*fname*):

transition: *logName* := path to log message file given by *fname*

exception: none

log_getFileName():

output: *out* := *logName*

exception: none

log_initLogFile():

transition: *logFile* := fileRefT to the file given by *logName*

exception: none

log_printLogMsg(*msg*,*sdr*):

transition: Write msg_getMsg(*msg*) to *logFile*
Write msg_getSdr(*sdr*) to *logFile*
Advance writing position in *logFile*

exception: none

log_closeLogFile():

transition: Close the file associated with *logFile*

exception: none

Local Functions
N/A

Local Variables
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.2.4.2 Log Messages

Uses

System Constants

Syntax

Exported Types

messageT := { OK, ALLOC, DIMEN, EXCEED, EXISTS, FORMT, POSDEF, POSIT, SZE, TYP }
senderT := { BFCRDR, BNDDAT, BNDRDR, BSYMAT, CNSMAT, DMNRDR, DNSMAT, FLDDAT, ICTRDR, ICVRDR, KBCRDR, LINSLV, MTLDAT, MTLRDR, NBCRDR, TNSWTR, VECTOR, VECWTR }

Exported Functions

Table 2.16: Exported function interfaces for Log Messages module

Name	Input	Output	Exceptions
msg_getMsg	messageT	string	
msg_getSdr	senderT	string	

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics

msg_getMsg(*code*):

output: *out* := (*code* = ALLOC → "Failed to allocate memory"
| *code* = DIMEN → "Size of data structure does not match size of target data structure"
| *code* = EXCEED → "Data value exceeds a defined minimum or maximum"
| *code* = EXISTS → "File does not exist",
| *code* = FORMT → "File data is not in expected format"
| *code* = POSDEF → "Matrix is not positive definite"
| *code* = POSIT → "Index exceeds size of data structure"
| *code* = SZE → "Specified size of data structure exceeds minimum or maximum allowable size"
| *code* = TYP → "Specified material property does not correspond to material type")

exception: none

msg_getSdr(*code*):

output: *out* := (*code* = BFCRDR → "Body Force Reader"
| *code* = BNDDAT → "Boundary Data"
| *code* = BNDRDR → "Boundary File Reader"
| *code* = BSYMAT → "Banded Symmetric Matrix"
| *code* = CNSMAT → "Constitutive Matrix"
| *code* = DMNRDR → "Domain File Reader"
| *code* = DNSMAT → "Dense Matrix"
| *code* = FLDDAT → "Field Data"
| *code* = ICTRDR → "Initial Tensor Field Reader"
| *code* = ICVRDR → "Initial Vector Field Reader"
| *code* = KBCRDR → "Kinematic BC Reader"
| *code* = LINSLV → "Linear Solver"
| *code* = MTLDAT → "Material Property Data"
| *code* = MTLRDR → "Material File Reader"
| *code* = NBCRDR → "Natural BC Reader"
| *code* = TNSWTR → "Tensor Field Writer"
| *code* = VECTOR → "Vector Data Type"
| *code* = VECWTR → "Vector Field Writer"

exception: none

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.3.1 System Constants

Uses
N/A

Syntax
Exported Constants

Table 2.17: Exported constants for System Constants module, part 1 of 2

Name	Type	Value
MAXLEN	integer	200
ONE_THIRD	real	$\frac{1}{3}$
NDIM	integer	2
NTNS	integer	3
NNODEL	integer	3
NNODELB	integer	2
MAX_NODES	integer	2000
MAX_DOFS	integer	6000
MAX_ELEMENTS	integer	5000
MAX_BOUNDELS	integer	2000
MAX_MATERIALS	integer	30
MAX_TIMESTEPS	integer	10000
E_MIN	real	0
E_MAX	real	1×10^{11}
NU_MIN	real	0
NU_MAX	real	0.499
RHO_MIN	real	0
RHO_MAX	real	1×10^{11}

Table 2.18: Exported constants for System Constants module, part 2 of 2

Name	Type	Value
COORD_MIN	real	-1×10^{11}
COORD_MAX	real	1×10^{11}
DISP_MIN	real	-1×10^{11}
DISP_MAX	real	1×10^{11}
VEL_MIN	real	-1×10^{11}
VEL_MAX	real	1×10^{11}
ACC_MIN	real	-1×10^{11}
ACC_MAX	real	1×10^{11}
SIG_MIN	real	-1×10^{11}
SIG_MAX	real	1×10^{11}
STR_SMALL	real	1×10^{-2}
STR_MIN	real	-STR_SMALL
STR_MAX	real	STR_SMALL
DTIME_MIN	real	1×10^{-11}
DTIME_MAX	real	1×10^{4}

Exported Types
N/A

Exported Functions
N/A

Semantics

State Variables
N/A

State Invariants
N/A

Assumptions
N/A

Access Routine Semantics
N/A

Local Functions
N/A

Local Types
N/A

Local Constants
N/A

Considerations
N/A

2.3.2.1 Field Data

Uses

Modules:
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
Material Property Data
Memory Access
System Constants

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 2.19: Exported function interfaces for Field Data module, part 1 of 2

Name	Input	Output	Exceptions
fld_initTime	real, integer		
fld_cleanTime			
fld_timeStep		real	
fld_numTimeSteps		integer	
fld_initNode	integer		ALLOC, SZE
fld_cleanNode			
fld_numNode		integer	
fld_getCoord	integer, integer	real	POSIT
fld_setCoord	integer, integer, real		EXCEED, POSIT
fld_getFix	integer, integer	boolean	POSIT
fld_setFix	integer, integer, boolean		POSIT
fld_initDof			SZE
fld_numDof		integer	
fld_getDof	integer, integer	integer	POSIT

Table 2.20: Exported function interfaces for Field Data module, part 2 of 2

Name	Input	Output	Exceptions
fld_getDisp	integer, integer	real	POSIT
fld_setDisp	integer, integer, real		EXCEED, POSIT
fld_getVel	integer, integer	real	POSIT
fld_setVel	integer, integer, real		EXCEED, POSIT
fld_getAcc	integer, integer	real	POSIT
fld_setAcc	integer, integer, real		EXCEED, POSIT
fld_getBodyAcc	integer, integer	real	POSIT
fld_setBodyAcc	integer, integer, real		EXCEED, POSIT
fld_getStressNode	integer, integer	real	POSIT
fld_setStressNode	integer, integer, real		EXCEED, POSIT
fld_getStrainNode	integer, integer	real	POSIT
fld_setStrainNode	integer, integer, real		EXCEED, POSIT
fld_getStrainRateNode	integer, integer	real	POSIT
fld_setStrainRateNode	integer, integer, real		EXCEED, POSIT
fld_initElem	integer		ALLOC, SZE
fld_cleanElem			
fld_numElem		integer	
fld_volElem	integer	real	POSIT
fld_getConnect	integer, integer	integer	POSIT
fld_setConnect	integer, integer, integer		POSIT
fld_getMaterial	integer	integer	POSIT
fld_setMaterial	integer, integer		POSIT
fld_getStressElem	integer, integer	real	POSIT
fld_setStressElem	integer, integer, real		EXCEED, POSIT
fld_getStrainElem	integer, integer	real	POSIT
fld_setStrainElem	integer, integer, real		EXCEED, POSIT
fld_getStrainRateElem	integer, integer	real	POSIT
fld_setStrainRateElem	integer, integer, real		EXCEED, POSIT

Semantics

State Variables

dTime : real
nTime : integer
nodes : set of nodeT
elements : set of elementT
ndof : integer
inod : integer
iel : integer

State Invariants

$|nodes| \leq$ MAX_NODES
$|elements| \leq$ MAX_ELEMENTS
$ndof \leq$ MAX_DOFS

Assumptions

1. The function fld_initNode() will always be called before other functions in Table 2.19.
2. The function fld_initElem() will always be called before other functions in Table 2.20.
3. The function fld_initDof() will be called after all nodes are set.
4. The function fld_initDof() will be called before fld_getDof().

Access Routine Semantics

fld_initTime(*dt*,*n*):

transition: $dTime := dt$
$nTime := n$

exception: none

fld_cleanTime():

transition: $dTime := 0.0$
$nTime := 0$

exception: none

fld_timeStep():

output: $out := dTime$

exception: none

fld_numTimeSteps():

output: $out := nTime$

exception: none

fld_initNode($nnod$):

transition: Allocate memory for $nnod$ nodeT objects in $nodes$

$ndof := 0$
$inod := 0$
$\forall n \in nodes$
{
  $inod := inod + 1$
  $n.num := inod$
  $\forall j \in$ DIM_SUB
  {
    $n.p.x_j := 0.0$
    $n.fix.fix_j := false$
    $n.dof.n_j := 0$
    $n.d.u_j := 0.0$
    $n.v.v_j := 0.0$
    $n.a.a_j := 0.0$
    $n.body.a_j := 0.0$
  }
  $\forall j \in$ TNS_SUB
  {
    $n.\sigma.\sigma_j := 0.0$
    $n.\varepsilon.\varepsilon_j := 0.0$
    $n.\dot{\varepsilon}.\dot{\varepsilon}_j := 0.0$
  }
}

exception: $exc :=$ ($nnod \notin$ [1..MAX_NODES] $\rightarrow$ SZE
| amount of memory required for $nnod$ nodeT objects $<$ mem_getAvailMem() $\rightarrow$ ALLOC)

fld_cleanNode():

transition: Deallocate memory for $nodes$ $ndof := 0$

exception: none

fld_numNode():

output: $out := |nodes|$

exception: none

fld_getCoord(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.p.x_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT})$

fld_setCoord(i,j,x):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.p.x_j := x\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT} \mid x < \text{COORD\_MIN} \rightarrow \text{EXCEED} \mid x > \text{COORD\_MAX} \rightarrow \text{EXCEED})$

fld_getFix(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.fix.fix_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT})$

fld_setFix(i,j,f):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.fix.fix_j := f\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT})$

fld_initDof():

transition: $\forall i \in [1..|nodes|]$

$$
\begin{array}{l}
\{ \\
\quad (\forall n \in nodes \\
\quad \{ \\
\qquad n.num = i \rightarrow \\
\qquad\quad \forall j \in \text{DIM\_SUB} \\
\qquad\quad \{ \\
\qquad\qquad (\neg n.fix.fix_j \rightarrow ndof, n.dof.n_j := ndof + 1, ndof + 1) \\
\qquad\quad \} \\
\qquad \} \\
\quad \}) \\
\}
\end{array}
$$

exception: $exc := (ndof > \text{MAX\_DOFS} \rightarrow \text{SZE})$

fld_numDof():

output: $out := ndof$

exception: none

fld_getDof(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.dof.n_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT})$

fld_getDisp(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.d.u_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT})$

fld_setDisp(i,j,u):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.d.u_j := u\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT} \mid u < \text{DISP\_MIN} \rightarrow \text{EXCEED} \mid u > \text{DISP\_MAX} \rightarrow \text{EXCEED})$

fld_getVel(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.v.v_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT})$

fld_setVel(i,j,v):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.v.v_j := v\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT} \mid v < \text{VEL\_MIN} \rightarrow \text{EXCEED} \mid v > \text{VEL\_MAX} \rightarrow \text{EXCEED})$

fld_getAcc(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.a.a_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT})$

fld_setAcc(i,j,a):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.a.a_j := a\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT} \mid a < \text{ACC\_MIN} \rightarrow \text{EXCEED} \mid a > \text{ACC\_MAX} \rightarrow \text{EXCEED})$

fld_getBodyAcc(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.body.a_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT})$

fld_setBodyAcc(i,j,a):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.body.a_j := a\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{DIM\_SUB} \rightarrow \text{POSIT} \mid a < \text{ACC\_MIN} \rightarrow \text{EXCEED} \mid a > \text{ACC\_MAX} \rightarrow \text{EXCEED})$

fld_getStressNode(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.\sigma.\sigma_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT})$

fld_setStressNode(i,j,s):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.\sigma.\sigma_j := s\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT} \mid s < \text{SIG\_MIN} \rightarrow \text{EXCEED} \mid s > \text{SIG\_MAX} \rightarrow \text{EXCEED})$

fld_getStrainNode(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.\varepsilon.\varepsilon_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT})$

fld_setStrainNode(i,j,s):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.\varepsilon.\varepsilon_j := s\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT} \mid s < \text{STR\_MIN} \rightarrow \text{EXCEED} \mid s > \text{STR\_MAX} \rightarrow \text{EXCEED})$

fld_getStrainRateNode(i,j):

output: $out := (\forall n \in nodes\ \{n.num = i \rightarrow n.\dot{\varepsilon}.\dot{\varepsilon}_j\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT})$

fld_setStrainRateNode(i,j,s):

transition: $(\forall n \in nodes\ \{n.num = i \rightarrow n.\dot{\varepsilon}.\dot{\varepsilon}_j := s\})$

exception: $exc := (i \notin [1..|nodes|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT} \mid s < \text{STRRATE\_MIN} \rightarrow \text{EXCEED} \mid s > \text{STRRATE\_MAX} \rightarrow \text{EXCEED})$

fld_initElem(nel):

transition: Allocate memory for nel elementT objects in $elements$

$iel := 0$
$\forall e \in elements$
{
  $iel := iel + 1$
  $e.num := iel$
  $\forall i \in [1..\text{NNODEL}]\,\{e.nd[i] := 0\}$
  $e.mtl := 0$
  $\forall j \in \text{TNS\_SUB}$
  {
    $e.\sigma.\sigma_j := 0.0$
    $e.\varepsilon.\varepsilon_j := 0.0$
    $e.\dot{\varepsilon}.\dot{\varepsilon}_j := 0.0$
  }
}

exception: $exc := (nel \notin [1..\text{MAX\_ELEMENTS}] \rightarrow \text{SZE}$
| amount of memory required for nel elementT objects $<$ mem_getAvailMem() $\rightarrow$ ALLOC)

fld_cleanElem():

transition: Deallocate memory for $elements$

exception: none

fld_numElem():

output: $out := |elements|$

exception: none

fld_volElem(i):

transition: ($\forall e \in elements$

$\{$
$e.num = i \rightarrow$
$\forall j \in [1..\text{NNODEL}]$
$\{$
$x_j :=$ fld_getCoord($e.nd[j]$,1)
$y_j :=$ fld_getCoord($e.nd[j]$,2)
$\}$
$\})$

output: $out := \frac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT})$

fld_getConnect(i,j):

output: $out := (\forall e \in elements\ \{e.num = i \rightarrow e.nd[j]\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid j \notin [1..\text{NNODEL}] \rightarrow \text{POSIT})$

fld_setConnect(i,j,c):

transition: $(\forall e \in elements\ \{e.num = i \rightarrow e.nd[j] := c\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid j \notin [1..\text{NNODEL}] \rightarrow \text{POSIT})$

fld_getMaterial(i):

output: $out := (\forall e \in elements\ \{e.num = i \rightarrow e.mtl\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT})$

fld_setMaterial(i,m):

transition: $(\forall e \in elements\ \{e.num = i \rightarrow e.mtl := m\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid m \notin [1..\text{mtl\_numMtl()}] \rightarrow \text{POSIT})$

fld_getStressElem(i,j):

output: $out := (\forall e \in elements\ \{e.num = i \rightarrow e.\sigma.\sigma_j\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT})$

fld_setStressNode(i,j,s):

transition: $(\forall e \in elements\ \{e.num = i \rightarrow e.\sigma.\sigma_j := s\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT} \mid s < \text{SIG\_MIN} \rightarrow \text{EXCEED} \mid s > \text{SIG\_MAX} \rightarrow \text{EXCEED})$

fld_getStrainElem(i,j):

output: $out := (\forall e \in elements\ \{e.num = i \rightarrow e.\varepsilon.\varepsilon_j\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT})$

fld_setStrainNode(i,j,s):

transition: $(\forall e \in elements\ \{e.num = i \rightarrow e.\varepsilon.\varepsilon_j := s\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT} \mid s < \text{STR\_MIN} \rightarrow \text{EXCEED} \mid s > \text{STR\_MAX} \rightarrow \text{EXCEED})$

fld_getStrainRateElem(i,j):

output: $out := (\forall e \in elements\ \{e.num = i \rightarrow e.\dot{\varepsilon}.\dot{\varepsilon}_j\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT})$

fld_setStrainRateNode(i,j,s):

transition: $(\forall e \in elements\ \{e.num = i \rightarrow e.\dot{\varepsilon}.\dot{\varepsilon}_j := s\})$

exception: $exc := (i \notin [1..|elements|] \rightarrow \text{POSIT} \mid j \notin \text{TNS\_SUB} \rightarrow \text{POSIT} \mid s < \text{STRRATE\_MIN} \rightarrow \text{EXCEED} \mid s > \text{STRRATE\_MAX} \rightarrow \text{EXCEED})$

Local Functions
N/A

Local Types

coordT := tuple of (x_i : real) $\forall i \in$ DIM_SUB

dispT := tuple of (u_i : real) $\forall i \in$ DIM_SUB

velT := tuple of (v_i : real) $\forall i \in$ DIM_SUB

accT := tuple of (a_i : real) $\forall i \in$ DIM_SUB

fixT := tuple of (fix_i : boolean) $\forall i \in$ DIM_SUB[1]

dofT := tuple of (n_i : integer) $\forall i \in$ DIM_SUB

stressT := tuple of (σ_i : real) $\forall i \in$ TNS_SUB

strainT := tuple of (ε_i : real) $\forall i \in$ TNS_SUB

strainRateT := tuple of ($\dot{\varepsilon}_i$: real) $\forall i \in$ TNS_SUB

connectT := sequence [NNODEL] of integer

nodeT := tuple of (*num* : integer, p : coordT, *fix* : fixT, *dof* : dofT, d : dispT, v : velT, a : accT, *body* : accT, σ : stressT, ε : strainT, $\dot{\varepsilon}$: strainRateT)

elementT := tuple of (*num* : integer, *nd* : connectT, *mtl* : integer, σ : stressT, ε : strainT, $\dot{\varepsilon}$: strainRateT)

Local Constants

VEC_SUB := $\{x | x \in \mathbb{N} \wedge 1 \leq x \leq \text{NDIM}\}$
TNS_SUB := $\{11, 22, 12, 33\}$

Considerations
N/A

[1]The type fixT represents kinematic constraints. At present, it is assumed that kinematic constraints, if present, are zero. This may change in future versions.

2.3.2.2 Boundary Data

Uses

Modules:
Field Data
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Types

surfLoadT := tuple of $(\sigma_{nn}, \sigma_{nt}, \theta : \text{real})$

Exported Functions

Table 2.21: Exported function interfaces for Boundary Data module

Name	Input	Output	Exceptions
bnd_init	integer		ALLOC, SZE
bnd_clean			
bnd_numBoundElem		integer	
bnd_lenBoundElem	integer	real	POSIT
bnd_getConnect	integer, integer	integer	POSIT
bnd_setConnect	integer, integer, integer		POSIT
bnd_getTrac	integer, integer	surfLoadT	POSIT
bnd_setTrac	integer, integer, surfLoadT		EXCEED, POSIT

Semantics

State Variables

boundElements : set of boundElementT
ielb : integer

State Invariants

$|boundElements| \leq$ MAX_BOUNDELS

Assumptions

1. The function bnd_init() will always be called before other functions in this module.

Access Routine Semantics

bnd_init(*nelb*):

transition: Allocate memory for *nelb* boundElementT objects in *boundElements*

$ielb := 0$
$\forall b \in boundElements$
{
  $ielb := ielb + 1$
  $b.num := ielb$
  $\forall i \in [1..\text{NNODELB}] \{b.nd[i] := 0\}$
  $\forall i \in [1..\text{NNODELB}] \{b.loads[i] := \langle\rangle\}$
}

exception: $exc := (nelb \notin [1..\text{MAX\_BOUNDELS}] \rightarrow$ SZE
| amount of memory required for *nelb* boundElementT objects < mem_getAvailMem() $\rightarrow$ ALLOC)

bnd_clean():

transition: Deallocate memory for *boundElements*

exception: none

bnd_numBoundElem():

output: $out := |boundElements|$

exception: none

bnd_lenBoundElem(i):[2]

transition: ($\forall b \in boundElements$

{
$b.num = i \rightarrow$
$\forall j \in$ [1..NNODELB]
{
$x_j :=$ fld_getCoord($b.nd[j]$,1)
$y_j :=$ fld_getCoord($b.nd[j]$,2)
}
})

output: $out := \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

exception: $exc := (i \notin [1..|boundElements|] \rightarrow$ POSIT)

bnd_getConnect(i,j):

output: $out := (\forall b \in boundElements\ \{b.num = i \rightarrow b.nd[j]\})$

exception: $exc := (i \notin [1..|boundElements|] \rightarrow$ POSIT | $j \notin$ [1..NNODELB] $\rightarrow$ POSIT)

bnd_setConnect(i,j,c):

transition: $(\forall b \in boundElements\ \{b.num = i \rightarrow b.nd[j] := c\})$

exception: $exc := (i \notin [1..|boundElements|] \rightarrow$ POSIT | $j \notin$ [1..NNODELB] $\rightarrow$ POSIT)

bnd_getTrac(i,j):

output: $out := (\forall b \in boundElements\ \{b.num = i \rightarrow b.loads[j]\})$

exception: $exc := (i \notin [1..|boundElements|] \rightarrow$ POSIT | $j \notin$ [1..NNODELB] $\rightarrow$ POSIT)

bnd_setTrac(i,j,t):

transition: $(\forall b \in boundElements\ \{b.num = i \rightarrow b.loads[j] := t\})$

exception: $exc := (i \notin [1..|boundElements|] \rightarrow$ POSIT | $j \notin$ [1..NNODELB] $\rightarrow$ POSIT | $t.\sigma_{nn} <$ SIG_MIN $\rightarrow$ EXCEED | $t.\sigma_{nn} >$ SIG_MAX $\rightarrow$ EXCEED | $t.\sigma_{nt} <$ SIG_MIN $\rightarrow$ EXCEED | $t.\sigma_{nt} >$ SIG_MAX $\rightarrow$ EXCEED)

[2]It is assumed for the time being that two-noded linear boundary elements will be used. If this changes, so will the length formula.

Local Functions
N/A

Local Types

boundConnectT := sequence [NNODELB] of integer
tracT := sequence [NNODELB] of surfLoadT
boundElementT := tuple of (*num* : integer, *nd* : boundConnectT, *loads* : tracT)

Local Constants
N/A

Considerations
N/A

2.3.2.3 Material Property Data

Uses

Modules:
Floating Point Operations
Integer Operations
Log Message Control
Log Messages
System Constants

Syntax

Exported Constants
N/A

Exported Types
N/A

Exported Functions

Table 2.22: Exported function interfaces for Material Property Data module

Name	Input	Output	Exceptions
mtl_init mtl_clean	integer		ALLOC, SZE
mtl_numMtl		integer	
mtl_getEmod	integer	real	POSIT, TYP
mtl_setEmod	integer, real		EXCEED, POSIT, TYP
mtl_getPois	integer	real	POSIT, TYP
mtl_setPois	integer, real		EXCEED, POSIT, TYP
mtl_getDens	integer	real	POSIT
mtl_setDens	integer, real		EXCEED, POSIT

Semantics

State Variables

materials : set of materialT
imtl : integer

State Invariants

$|materials| \leq$ MAX_MATERIALS

Assumptions

1. The function mtl_init() will always be called before other functions.

Access Routine Semantics

mtl_init(*nmtl*):

transition: Allocate memory for *nmtl* materialT objects in *materials*

$imtl := 0$
$\forall m \in materials$
{
  $imtl := imtl + 1$
  $m.num := imtl$
  $m.type := linear\_elastic$
  $m.E := 0.0$
  $m.\nu := 0.0$
}

exception: $exc := (nmtl \notin [1..\text{MAX\_MATERIALS}] \rightarrow \text{SZE}$
| amount of memory required for *nmtl* materialT objects < mem_getAvailMem() → ALLOC)

mtl_clean():

transition: Deallocate memory for *materials*

exception: none

mtl_numMatl():

output: $out := |materials|$

exception: none

mtl_getEmod(i):

output: $out := (\forall m \in materials\ \{m.num = i \rightarrow m.E\})$

exception: $exc := (i \notin [1..|materials|] \rightarrow \text{POSIT} \mid m.type \neq linear\_elastic \rightarrow \text{TYP})$

mtl_setEmod(i,E):

transition: $(\forall m \in materials\ \{m.num = i \rightarrow m.E := E\})$

exception: $exc := (i \notin [1..|materials|] \rightarrow \text{POSIT} \mid m.type \neq linear\_elastic \rightarrow \text{TYP} \mid E < \text{E\_MIN} \rightarrow \text{EXCEED} \mid E > \text{E\_MAX} \rightarrow \text{EXCEED})$

mtl_getPois(i):

output: $out := (\forall m \in materials\ \{m.num = i \rightarrow m.\nu\})$

exception: $exc := (i \notin [1..|materials|] \rightarrow \text{POSIT} \mid m.type \neq linear\_elastic \rightarrow \text{TYP})$

mtl_setPois(i,ν):

transition: $(\forall m \in materials\ \{m.num = i \rightarrow m.\nu := \nu\})$

exception: $exc := (i \notin [1..|materials|] \rightarrow \text{POSIT} \mid m.type \neq linear\_elastic \rightarrow \text{TYP} \mid \nu < \text{NU\_MIN} \rightarrow \text{EXCEED} \mid \nu > \text{NU\_MAX} \rightarrow \text{EXCEED})$

mtl_getDens(i):

output: $out := (\forall m \in materials\ \{m.num = i \rightarrow m.\rho\})$

exception: $exc := (i \notin [1..|materials|] \rightarrow \text{POSIT}\ )$

mtl_setDens(i,ρ):

transition: $(\forall m \in materials\ \{m.num = i \rightarrow m.\rho := \rho\})$

exception: $exc := (i \notin [1..|materials|] \rightarrow \text{POSIT} \mid \rho < \text{RHO\_MIN} \rightarrow \text{EXCEED} \mid \rho > \text{RHO\_MAX} \rightarrow \text{EXCEED})$

Local Functions
N/A

Local Types
materialTypeT := {*linear_elastic*}
materialT := tuple of (*num* : integer, *type* : materialTypeT, E : real, ν : real, ρ : real)

Local Constants
N/A

Considerations
N/A

2.3.3 PDE Solver

See documentation in DynSWS-MID-PDE-1.0 [3].

References

[1] B. Karchewski, "Module guide for two and three dimensional dynamic model of soil-water-structure interaction," Course Project - CES 741, McMaster University, Mar. 2012.

[2] ——, "Software requirements specification for two and three dimensional dynamic model of soil-water-structure interaction," Course Project - CES 741, McMaster University, Feb. 2012.

[3] ——, "Module internal design for partial differential equation solver module for two and three dimensional dynamic model of soil-water-structure interaction," Course Project - CES 741, McMaster University, Mar. 2012.