CAVS REPORT

MSU.CAVS.CMD.2010-R0001

Cyberinfrastructure-based Codes Repository: Description of the Template for Individual Project Folder Organization

Project: DOE-SRCLID
Project Leaders: Mark F. Horstemeyer, Paul T. Wang
Task 2: Cyberinfrastructure
Task 2 Leader: Tomasz Haupt
Task 2 Members: Tomasz Haupt, Ricolindo Carino, and Florina Ciorba

Prepared by: **Florina Ciorba**
Center for Advanced Vehicular Systems
Mississippi State University
Mississippi State, MS 39762

Web site: http://www.cavs.msstate.edu

For public release.

TABLE OF CONTENTS

1. Document Revision History

Date	Version	Description	Author(s)
01/8/10	1.0	Initial Release	Florina Ciorba
03/15/10	2.0	Updated Release	Florina Ciorba

2. Abstract

The purpose of this document is to describe the organization of the project folder and the folder content of any new project that will become a part of the Cyberinfrastructure-based codes repository. The document gives a brief overview of the codes repository as part of the DOE-SRCLID Task 2 - Cyberinfrastructure. In addition, it outlines the steps involved in deploying a new project to the Cyberinfrastructure-based codes repository.

3. Overview of the Cyberinfrastructure-based Codes Repository

As part of the DOE-SRCLID project, one of the tasks of the Cyberinfrastructure team is to develop a repository of computational modules to be shared by all participants of SRCLID tasks. The computational modules correspond to material models simulation codes, pre- and/or post-processing data analysis and visualization codes, etc. Management of changes to computer programs and any other electronic information is traditionally done using a version control system, which is able to track changes and avoid general chaos particularly for large, fast-changing projects, possibly with many authors.

The *repository of material models codes* is developed using Subversion (SVN) [1], an open source, version control system, and hosted at '**/cavs/cmd/data2/svnrepos/cmdcodes/**'. The *Cyberinfrastructure-based codes repository* is a mirror of the SVN *repository of material models codes*. This mirror is hosted onto a machine in the DMZ (or demilitarized zone). Consequently, the content of the *Cyberinfrastructure-based codes repository* is available world-wide on the Internet and can be browsed via the ViewVC interface [2].

4. Individual Project Template

To achieve and maintain consistency in the codes repository, we propose two project templates, one for small or very small software projects, called '**basicSkeleton**', and one for larger software projects, with numerous files, possibly of various types, called '**skeleton**'. Both templates are intended to serve as guidelines for organizing the folder structure of Cyberinfrastructure-based codes repository candidate projects, and suggestions on mapping the project content (documentation, source files, dependence libraries, input/output data, example uses, etc) onto the proposed organization structure. The organization structure and content categorization of the two proposed templates are drawn from common standard practices of software development and software project maintenance.

Each template can prove extremely useful for new materials models users, e.g. new students, new faculty, or external users, and others. The two proposed templates, '**basicSkeleton'** and '**skeleton'**, can also be very useful to more advanced material models users, especially for tracking changes made when implementing new changes or corrections to the material models, as well as when adding new model features. Finally, the goal of proposing these templates is to assist all users in preparing their material models codes for deployment onto the Cyberinfrastructure-based codes repository.

The '**basicSkeleton**' template is recommended for small projects that consist of one or two source files and maybe one header file. The tree-like folder structure of the **'basicSkeleton'** template is given in Figure 1. The recommended content of each folder illustrated in Figure 1 is described in Section 4.1

For more complex and/or larger projects, we propose the folder organization illustrated in Figure 2, called '**skeleton**'. Generally, such projects consist of more than one source file (*.c, *.cpp or *.f), several header files (*.h) and a couple of libraries (*.a, *.so or *.dll). It is very likely that such projects require a Makefile, and that the user may need some scripts for pre-simulation or post-simulation data processing. A good practice is to organize these files into specific directories, as illustrated in Figure 2. A description of the content of each folder and file is given in Section 4.2.

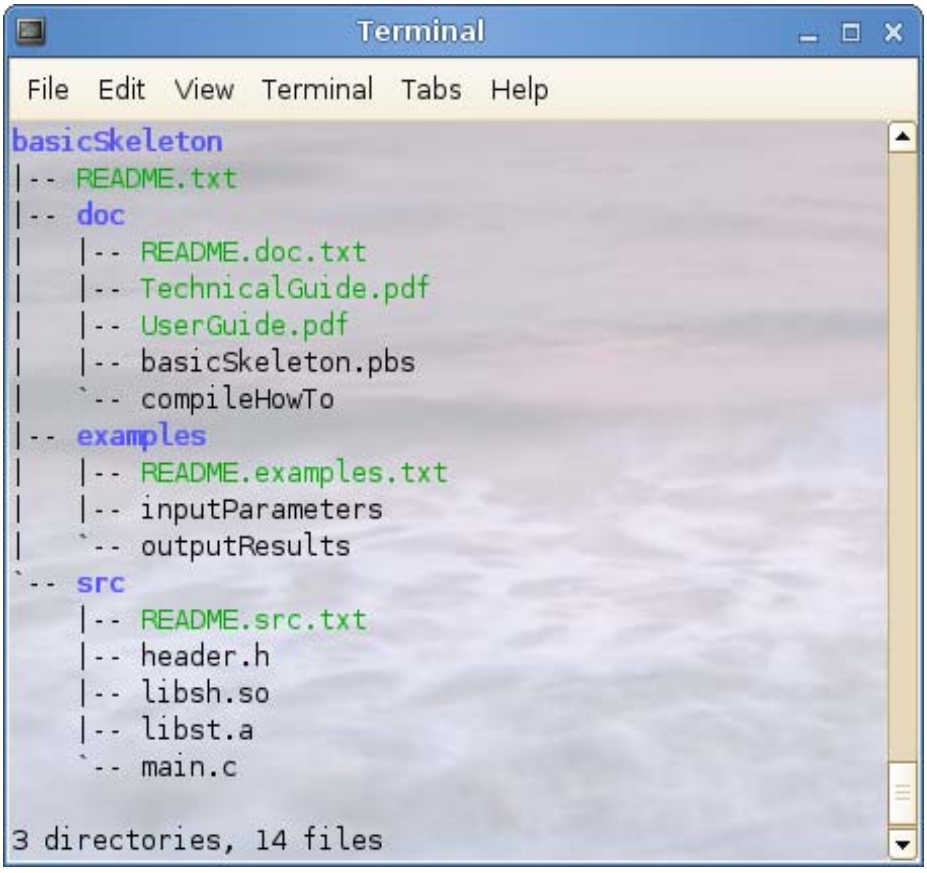

Figure 1 Tree organization for the template folder into 3 directories (folders) and 14 files for a simple software project

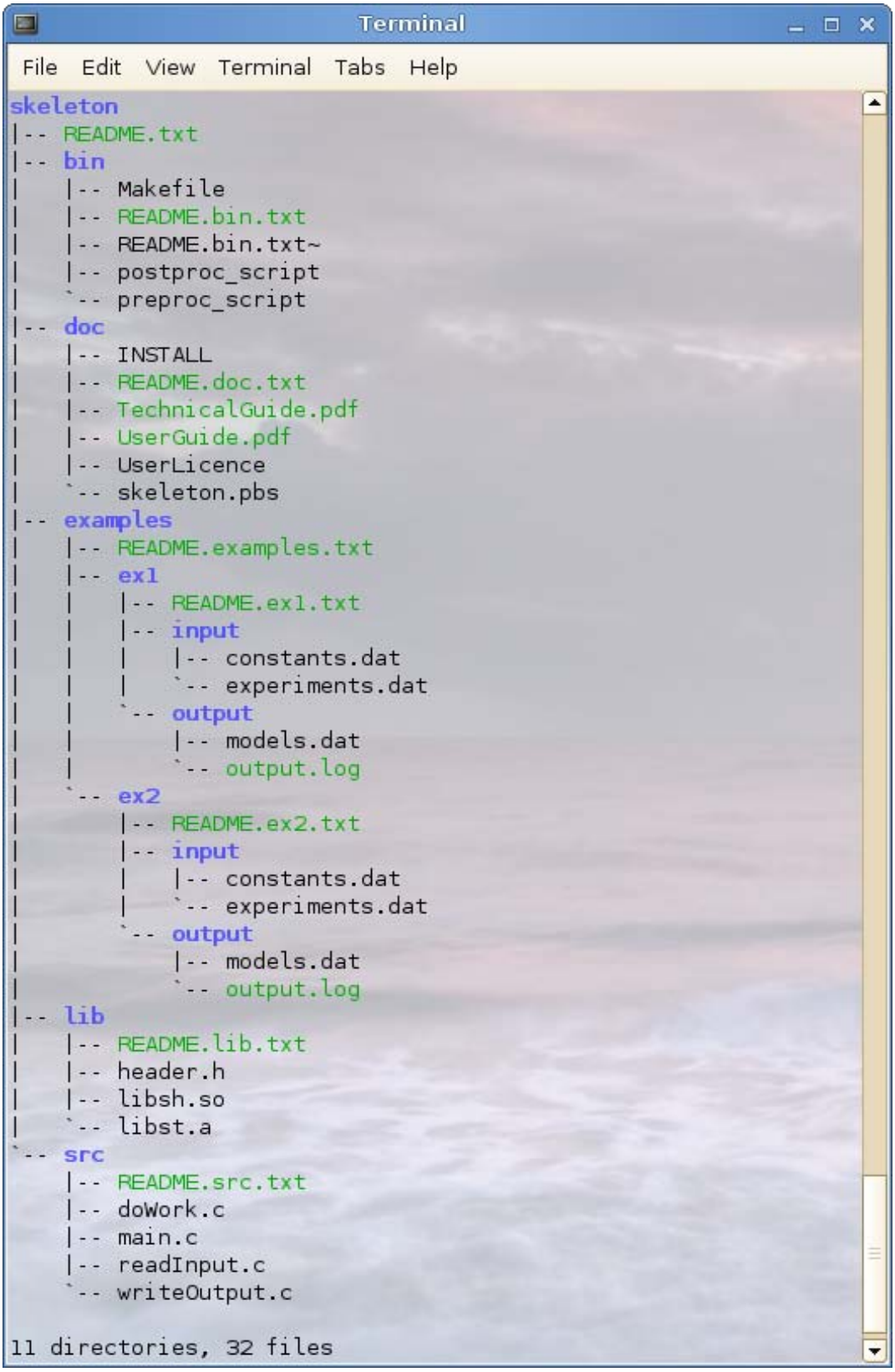

Figure 2 Tree organization for the template folder into 11 directories (folders) and 32 files for a larger or more complex software project

4.1. Description of folder organization

The organization and content of the sample small and large project templates is purely illustrative and not mandatory. Depending on the software project, the folder organization may be slightly different from the one illustrated in Figure 1 or in Figure 2. This report aims to provide a set of useful guidelines to help the user prepare their software projects for deployment in the Cyberinfrastructure-based repository of codes, with the least amount of effort and for the best possible outcome.

Illustratively, in Table 1 we highlight the recommended and optional files for both proposed templates. The files or folders marked as 'recommended' are intended to give the users an idea of the files or folders that are important for a project, and which should preferably not be omitted. For a better understanding of the file types marked as recommended we use different colors as shown in Table 1.

Small project folder organization	Large project folder organization
basicSkeleton * |-- README.txt * |-- doc * | |-- README.doc.txt | |-- TechnicalGuide.pdf * | |-- UserGuide.pdf * | |-- basicSkeleton.pbs | `-- compileHowTo * |-- examples * | |-- README.examples.txt * | |-- inputParameters * | `-- outputResults * `-- src * |-- README.src.txt |-- header.h |-- libst.a |-- libsh.so `-- main.c *	skeleton * |-- README.txt * |-- bin * | |-- Makefile * | |-- README.bin.txt | |-- postproc_script | |-- preproc_script | `-- skeleton.pbs |-- doc * | |-- INSTALL | |-- README.doc.txt | |-- TechnicalGuide.pdf * | |-- UserGuide.pdf * | `-- UserLicence |-- examples * | |-- README.examples.txt * | |-- ex1 * | | |-- README.ex1.txt | | |-- input * | | | |-- constants.dat * | | | `-- experiments.dat | | `-- output * | | |-- models.dat | | `-- output.log * | `-- ex2 | |-- README.ex2.txt | |-- input | | |-- constants.dat

	| | `-- experiments.dat |  `-- output | |-- models.dat | `-- output.log |-- lib |  |-- README.lib.txt |  |-- libst.a |  |-- libsh.so |  `-- Header.h `-- src *   |-- README.src.txt   |-- doWork.c   |-- main.c *   |-- readInput.c   `-- writeOutput.c
3 folders, 14 files	**11 folders, 32 files**

Table 1 Tree organization of recommended and optional files and folders for small and large software projects: recommended files/folders are marked with a **red asterisk** '*', recommended text files are highlighted in **green color**, recommended folders – in **blue color** and optional files are highlighted in **black color**

4.2. Description of folder content

In this section we give a description of the content of each folder and file type. The rationale behind the proposed **small** project organization and content distribution is that every small project should preferably contain: **README.txt** file describing briefly the purpose and content of the project; **doc** folder with more detailed documentation on the theory behind the material model, technical details regarding the implementation of the material model in a computer program, as well as instructions on how to compile and execute the project locally, or remotely using a queuing system (e.g. PBS); **examples** (at least one) to help a new user understand how to give the required input; and most importantly a **src** folder, containing the source files of the material model implementation. A detailed description of each small project folder and file is given in Table 2.

Small project file/folder name	Content Description
basicSkeleton *	top level project folder
|-- README.txt *	description of the files and folders of the project; meant to be read before taking any significant action with the items in the directory
|-- doc *	documentation folder
| |-- README.doc.txt	information about the files contained in this folder

| |-- **TechnicalGuide.pdf** *	theory behind the material model (e.g. technical report, journal article, conference paper, etc.)
| |-- **UserGuide.pdf** *	information on the material model implementation in a computer program (e.g. numerical methods used, precision, etc.)
| |-- **basicSkeleton.pbs**	instructions for remote execution (e.g. using PBS queues)
| `-- **compileHowTo** *	Instructions on compiler options for creating the executable version of the project
|-- **examples** *	Illustrating examples folder; if the examples and related data are too large to be included together in the codes repository, please describe the location of illustrative examples for this project
| |-- **README.examples.txt** *	description of the example, their purpose etc.
| |-- **inputParameters** *	input parameters used in this example
| `-- **outputResults** *	output results produced with this example
`-- **src** *	source code folder (*.c, *cpp, *.f, *.f90); the files given in this template are illustrative of C source files
|-- **README.src.txt**	details on the source files and their inter-relations
|-- **header.h**	declarations of classes, subroutines, variables, and other identifiers
|-- **libst.a**	software static library required by the model
|-- **libsh.so**	software shared library required by the model
`-- **main.c** *	implementation of main (which calls all other routines) – the routine called by the compiler

Table 2 Description of folder content, including the files and subfolders in the proposed 'basicSkeleton' software project template

Similarly to smaller software projects, the rationale behind the proposed **larger** software project organization and content distribution is that every larger project should preferably contain: **README.txt** file describing briefly the purpose and content of the project; **bin** folder, containing instructions on how to compile and test the project; **doc** folder with more detailed documentation on the theory behind the material model, technical details regarding the implementation of the material model in a computer program, as well as instructions on how to execute the project locally, or remotely using a queuing system (e.g. PBS); **examples** (at least one) aiding the new user in understanding the type of input and output required and produced, respectively, by the project code; **lib** folder containing any software libraries that may be required by the main project code; and most

importantly the **src** folder, containing the source files of the material model implementation. A detailed description of each folder and file is given in Table 3.

Large project file/folder name	Content Description
skeleton *	top level project folder
|-- README.txt *	description of the files and folders of the project; meant to be read before taking any significant action with the items in the directory
|-- bin *	"how-to" folder
| |-- Makefile *	compiler options for creating the executable version of the project
| |-- README.bin.txt	information about the files contained in this folder
| |-- postproc_script	output data manipulation scripts
| |-- preproc_script	input data manipulation scripts
| `-- skeleton.pbs	instructions for remote execution (e.g. using PBS queues)
|-- doc *	documentation folder
| |-- INSTALL	instructions on parameters configuration , compiler requirements, etc.
| |-- README.doc.txt	information about the files contained in this folder
| |-- TechnicalGuide.pdf *	theory behind the material model (e.g. technical report, journal article, conference paper, etc.)
| |-- UserGuide.pdf *	information on the material model implementation in a computer program (e.g. numerical methods used, precision, etc.)
| `-- UserLicence	license governing the usage or redistribution of the project (all licenses, including free ones, must be explicitly specified in this document)
|-- examples *	Illustrating examples folder; if the examples and related data are too large to be included together in the codes repository, please describe the location of illustrative examples for this project
| |-- README.examples.txt *	description of the example, their purpose etc.
| |-- ex1 *	illustrating example 1
| | |-- README.ex1.txt	details on the characteristics of example 1
| | |-- input *	input parameters used in this example
| | | |-- constants.dat *	input constants used by the model
| | | `-- experiments.dat	input from experiments used by the model
| | `-- output *	output results produced with this example
| | |-- models.dat	output data (constants, etc) produced by the model for example 1

| | `-- output.log *	execution log of example 1
| `-- ex2	illustrating example 2
| |-- README.ex2.txt	details on the characteristics of example 2
| |-- input	input parameters used in this example
| | |-- constants.dat	input constants used by the model
| | `-- experiments.dat	input from experiments used by the model
| `-- output	output results produced with this example
| |-- models.dat	output data (constants, etc) produced by the model for example 2
| `-- output.log	execution log of example 2
|-- lib	libraries folder
| |-- README.lib.txt	instruction on creating/obtaining the libraries
| |-- libst.a	software static library required by the model
| |-- libsh.so	software shared library required by the model
| `-- Header.h	declarations of classes, subroutines, variables, and other identifiers
`-- src *	source code folder; the files given in this template are illustrative of C source files - this folder is by no means limited to C-language source files
|-- README.src.txt	details on the source files and their inter-relations
|-- doWork.c	implementation material model routine (called by the main routine)
|-- main.c *	implementation of main (which calls all other routines) – the routine called by the compiler
|-- readInput.c	implementation of the routine responsible for reading the input data to be used during execution
`-- writeOutput.c	implementation of the routine responsible for storing the data and the output produced during execution

Table 3 Description of folder content, including the files and subfolders in the proposed 'skeleton' template

Candidate projects representing or using software developed by CAVS should include the Software Application License Agreement that applies to the particular software project.

As a common practice both for small and large project, it is recommended that every '**README**' text file include the information described in Table 2 in the form of a document header. This would prevent the user from unnecessary folder searches by providing proper

information right where it is need it. Such appropriate information proximity is highly desirable for making the project be self-contained, self-explanatory, and self-consistent.

Project Title	[Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date	MM-DD-YYYY [indicate the date of last modification]*required
Author(s)	[list all people who wrote the code, added new features, etc] *required
Email Address	[recommended]
Web Page	[optional]
Description	[one sentence description of the project] *required
Organization and content of current folder	[describe the organization of files or directories in the current folder] *required

Table 4 Recommended document header with descriptive information about the content of the current folder

5. Deployment of a new project to the Cyberinfrastructure-based codes repository

This section describes the procedure for deploying a new project to the Cyberinfrastructure-based codes repository (CI-B CR). The procedure for deployment of new candidate projects described here and illustrated in Figure 3, is the first step towards a rapid and efficient population of the Cyberinfrastructure-based codes repository. This procedure is not meant to be absolute and we believe any improvements will become obvious as soon as it is put into practice.

5.1. Request for approval to include a new project into the codes repository

In order to obtain the approval for deployment of '**projectA**', a user, say '***user1***', who may be the actual developer or just the main user of '**projectA**', contacts the leader of the appropriate Task, say 'TaskB', to which '**projectA**' is related or in which '**projectA**' is used. The task leader contacts the DOE-SRCLID program leaders, i.e., Dr. M. Horstemeyer and Dr. P. Wang. As soon as the program leaders, together with the task leader, agree that '**projectA**' should be included in the Cyberinfrastructure-based codes repository, they notify the user as

well as the administrators of the repository, namely Dr. F. Ciorba, Mr. G. Henley, Dr. R. Carino and Dr. T. Haupt, regarding the approval for including '**projectA**' in the repository.

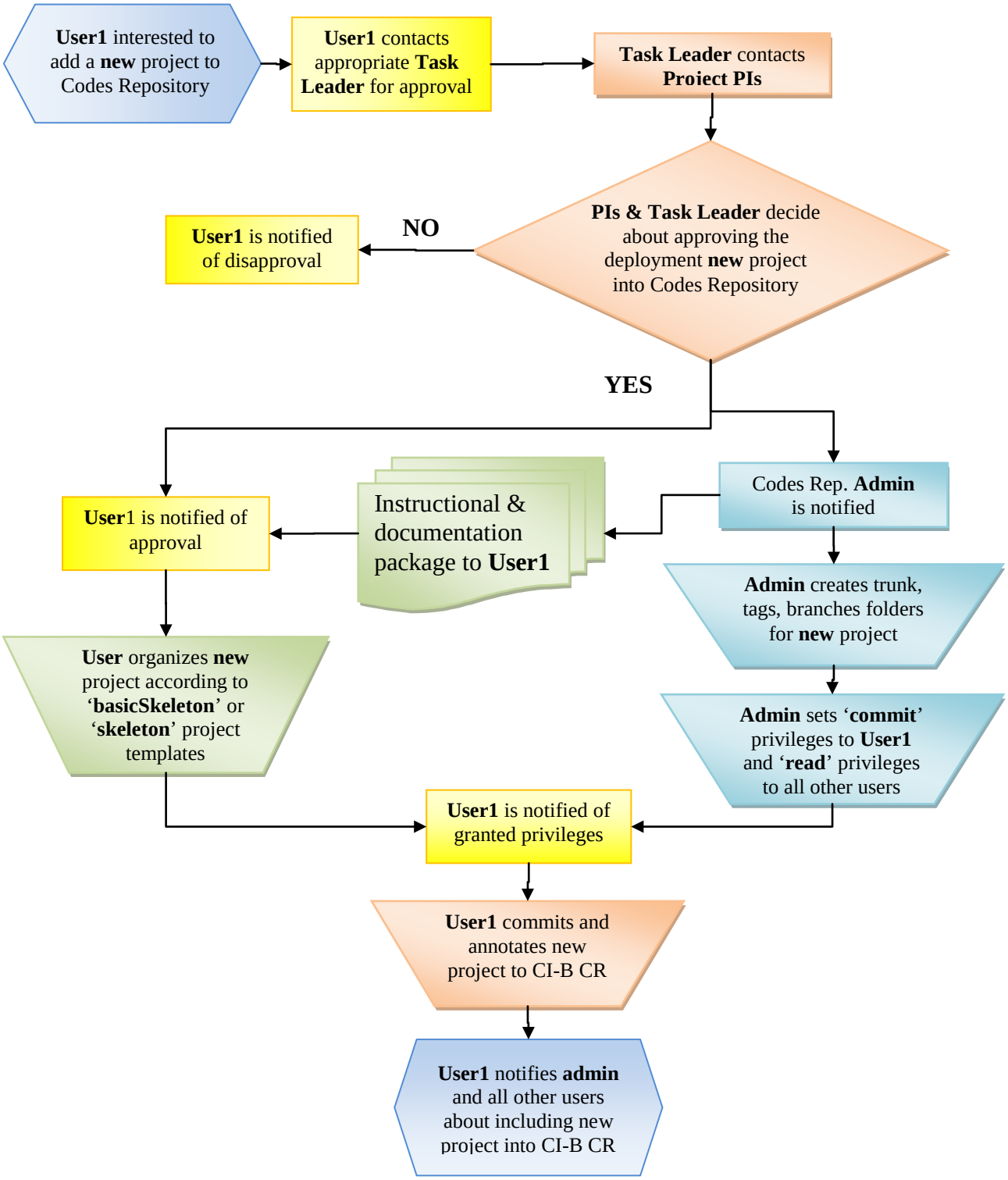

Figure 3 Flowchart describing the procedure for deployment of a new software project to the CI-B CR

5.2. Provision of Instructional Package to the User

In the second step, the administrator(s) of the codes repository provides the user with a complete package of instructions and documentation consisting of: the current document, a zip or a tar archive of the 'skeleton' template, detailed instructions on using SVN Linux and Windows clients. This package will cover all aspects related to the SVN codes repository and the related procedures and policies.

5.3. Organization of the candidate project based on the recommended template

In the third step the user organizes the candidate 'projectA' along the lines of the recommended '**skeleton**' or '**basicSkeleton**' project template described in Section 4. This step is *crucial* to ensuring thorough documentation, good sustainability, and wide usability of the candidate project, particularly in the following cases:

- new features are being added by multiple authors (developers)
- the project is in use by many users, with different degrees of expertise in the field
- external users are interested in using the project or
- original developer is no longer involved in the project, etc.

5.3.1 Public access to repository content

The *SVN repository of material model codes* is hosted at '**/cavs/cmd/data2/svnrepos/cmdcodes/**' and mirrorred onto a machine in the DMZ. Machines in DMZ provide external services that are exposed to larger networks, such as the Internet. Consequently, the content of the *SVN repository of material model codes* is made available on the Internet. The *Cyberinfrastructure-based codes repository* is public a mirror of the *SVN repository of material model codes*, providing access to repository content via the ViewVC browser interface. Every user must bear this in mind when deciding upon the actual content to include in the *SVN repository of material model codes*.

5.3.2 Intellectual property and licensing

It is strongly recommended that content intended to be included in the CI-B CR, such as

material model implementation, software package, libaries, etc., which falls under proprietarship, intellectual property, licencing and export control rules or regulations, **not** be deployed to the CI-B CR. Therefore, it is recommended that the users guarantee **non violation** of any rules or regulations pertaining and applicable to the content in question.

5.4. Creation of new project directory in the repository and customization of permissions

This step may take place at the same time with the third step. Specifically, during the time the user is organizing the candidate project appropriately, the codes repository administrator creates the appropriate project directory in the repository for the candidate project, in which the **trunk**, **tags**, and **branches** folders are also created. Next, the codes repository administrator grants the author (i.e., **user1**) the privileges for '**committing**' (i.e., uploading) to that project directory, and notifies the author that the codes repository is ready to host the candidate project.

5.5. New project commitment to the codes repository

Once the user has been notified about the creation of a directory in the codes repository to host the candidate project and enforcement of proper access permissions, the user is able to deploy '**projectA**' to the codes repository.

The first deployment of any new candidate project will constitute the main '**trunk**' of the particular project, and hence will appear under the **projectA/trunk** folder. Depending on the evolution of the project, subsequent commits can be done to the '**tags**', appearing under **projectA/tags**, or '**branches**', under **projectA/branches** folders.

A detailed explanation of **trunks**, **branches**, and **tags** is given in the technical report accompanying the current document [3]. Herein we include an example of a history tree with **trunk**, **tags** and **branches** in Figure 3. In addition, we give a very brief description of **trunk**, **tags** and **branches** below:

projectA/trunk – contains the *main code base*; it is always stable

projectA/tags – contains *code releases*, each release being a set of code (or a code snapshot) deployed under a certain tag, e.g., **tag "1.0"** would appear as **projectA/tags/1.0**.

projectA/branches – contains *various new features of certain code releases*, e.g. the code from the **trunk**, that has been tagged as release "**1.0**", having a new feature is considered **branch "1.1"**, and **appears** under **projectA/branches/1.1**.

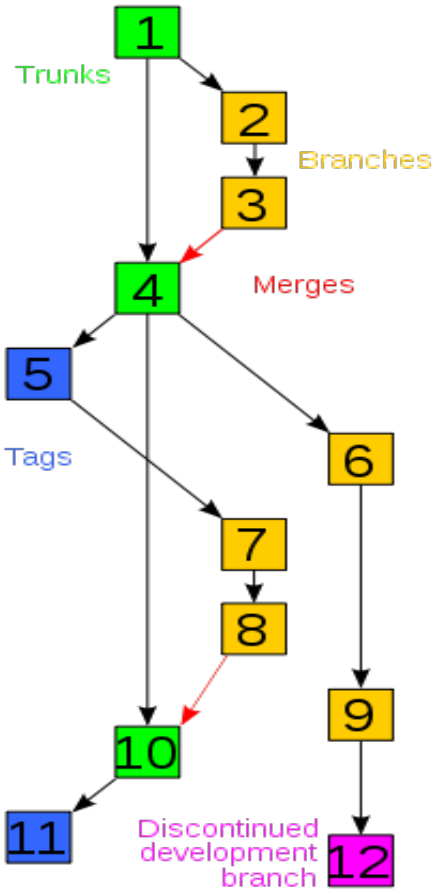

Figure 4 Example of a history tree of a version controlled project, showing branching, merging and tagging (image obtained under license Cc-by-sa-3.0-migrated, GFDL from http://en.wikipedia.org/wiki/File: Revision_controlled_project_visualization.svg).

5.6. Annotation of the project in the codes repository

The main motivation behind developing a repository of material models codes is increasing the usability of the existing material models implemented as computer programs and currently in use by CAVS researchers. One of the immediate benefits of such a repository is increasing the "life span" of the codes hosted by the repository. This benefit is crucial to the rapid dissemination of any new features or models, while providing a very short learning and adoption curve for new and/or unfamiliar users.

Following the initial commitment of projectA to the repository under projectA/trunk, the author is required to give a tag to the particular project. The 'initial tag' is meant to serve as an indicator of existing or future codes releases for projectA. Hence, the first tag could be '1.0', and any other user interested in using projectA for the first time, would be encouraged to download projectA/tags/1.0 onto a local machine.

It may often be the case that the person requesting the inclusion of a project in the codes repository is not the developer of the code, but a user. A pertinent example is LAMMPS, developed at SANDIA and used worldwide for a wide range of simulations and scopes. For this example, the person requesting to include LAMMPS in the codes repository would most likely be the main LAMMPS user at CAVS, say 'user2'. Following the procedure described herein, LAMMPS would be included committed to the codes repository under lammps/trunk, then tagged as release '1.0' under lammps/tags/1.0, and annotated to indicate the version of LAMMPS used by 'user2' and the date this version of LAMMPS was released by its developers, e.g., on 9 Jan 2009.

Assuming 'user2' employs LAMMPS for several simulations in which different libraries are used and different phenomena are studied, 'user2' should create a different tag for each of the various uses of LAMMPS. For instance, tag '1.1' would indicate that release tagged 1.0 of LAMMPS is used to study one phenomenon, while tag '1.2' implies that the same release 1.0 is used to study a different phenomenon. These annotations would appear under lammps/tags/1.1 and /lammps/tags/1.2, respectively. The differences regarding the use of lammps/tags/1.1 vs lammps/tags/1.0 should be clearly stated in at least one of the following documents: lammps/tags/1.1/README.txt, lammps/tags/1.1/bin/README.bin.txt, lammps/tags/1.1/doc/README.doc.txt. This holds for any new different way of using LAMMPS, release 1.0.

In the case that 'user2' decides to use a more recent version of LAMMPS, e.g. the one released on 7 Jul 2009, there are two options:

- if 'user2' considers that the previous version released on 9 Jan 2009 should be preserved separately from the latest version, he would have to create and annotate appropriately a corresponding branch '1.0', under lammps/branches/1.0
- if 'user2' considers that the previous version released on 9 Jan 2009 is not of use and could be replaced with the latest version, there would be no need for a new branch

Regardless of the choice above, the latest version released on 7 Jul 2009 needs to be tagged as '2.0' under lammps/tags/2.0 with the appropriate annotation. Every new or different way of using the LAMMPS tagged 2.0 should be tagged as 2.1, 2.2 and so on, while each new tag contains the proper explanations and clarifications in at least one of the following documents:

lammps/tags/2.1/README.txt,

lammps/tags/2.1/bin/README.bin.txt

lammps/tags/2.1/doc/README.doc.txt, and so on.

Multiple branches of any project, each with features different than the features in other branches can be merged together, such that the trunk contains the features of all branches, see Figure 3.

References

1. [SVN, 2009] http://subversion.tigris.org/

2. [ViewVC, 2009] http://www.viewvc.org/

3. [CCG TR-02, 2009] TR-2009-02.v1.0-GH, “Introduction to Subversion”, Greg Henley.

APPENDIX

A. Template files included in 'basicSkeleton'

A.1 README.txt

Package: **basicSkeleton**	Location: **top level folder**	File: **README.txt**

Project Title : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date : MM-DD-YYYY [indicate the date of last modification]*required
Author(s) : [list all people who wrote the code, added new features, etc] *required
Email Address : [recommended]
Web Page : [optional]
Description : [one sentence description of the project] *required
Organization and
content of current
folder : [describe the organization of files or directories in the current folder] *required

This is the README for the Project <basicSkeleton>. README is a normal plain text file containing information about other files in the directory and is meant to be read before taking any significant action with the items in the directory.

```
basicSkeleton *
|-- README.txt *
|-- doc *
|   |-- README.doc.txt
|   |-- TechnicalGuide.pdf *
|   |-- UserGuide.pdf *
|   |-- basicSkeleton.pbs
|   `-- compileHowTo *
|-- examples *
|   |-- README.examples.txt *
|   |-- inputParameters *
|   `-- outputResults *
`-- src *
    |-- README.src.txt
    |-- header.h
    |-- libst.a
```

```
   |-- libsh.so
   `-- main.c *
```

A.2 README.doc.txt

Package: **basicSkeleton**	Location: **doc**	File: **README.doc.txt**

```
Project Title  : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date   : MM-DD-YYYY [indicate the date of last modification]*required
Author(s)      : [list all people who wrote the code, added new features, etc] *required
Email Address  : [recommended]
Web Page       : [optional]
Description    : [one sentence description of the project] *required
Organization and
content of current
folder         : [describe the organization of files or directories in the current folder] *required

basicSkeleton *
|-- doc *
|   |-- README.doc.txt
|   |-- TechnicalGuide.pdf *
|   |-- UserGuide.pdf *
|   |-- basicSkeleton.pbs
|   `-- compileHowTo *
```

A.3 CompileHowTo

Package: **basicSkeleton**	Location: **doc**	File: **compileHowTo**

```
########################
#Project Title  : [Give the name of your software project, e.g. DMGfit or UMAT] *required
#Release Date   : MM-DD-YYYY [indicate the date of last modification]*required
#Author(s)      : [list all people who wrote the code, added new features, etc] *required
#Email Address  : [recommended]
#Web Page       : [optional]
#Description    : [one sentence description of the project] *required
#Note           :  Ensure that the C, C++ or Fortran compiler you use is in your path, by
#                    issuing the corresponding swsetup command in the terminal windows from
#                    which you will run your project.
#Recommended    : Please run the commands described below from the src folder of your
project.
########################

1. Compile src/main.c to an object file (without linking):

icc -O2 -check bounds main.c -c main.o
```

2. Link src/main.o with the proper libraries to create an executable binary:

icc -o main main.o -lst (or -lsh)

3. Run the project
./main

A.4 README.doc.txt

Package: **basicSkeleton**	Location: **doc**	File: **basicSkeleton.pbs**

```
#!/bin/sh
#PBS -N skeleton
#PBS -q q16p96h@raptor
#PBS -l nodes=2:ppn=4
#PBS -l walltime=60:00:00
#PBS -M username@cavs.msstate.edu
#PBS -m abe
#PBS -j oe
#PBS -r n
#PBS -V
# Set the stack size to unlimited
ulimit -s unlimited
# Set the core size to zero
ulimit -c 0
# List all resource limits
ulimit -a
echo "I ran on:"
# Print the nodes on which the project will run
cat $PBS_NODEFILE
# Change to your execution directory to your project directory.
cd /cavs/cmd/data1/users/<jdoe>/your-run-dirSandbox/bin/
# For projects that write often to files during execution, copy the all files required for
execution to /data/lustre/
# (the project executable, the pbs script, the input files), submit the project from there, while
at the end of the
# execution copy all files back to your project directory.
# Change to your execution directory
cd /data/lustre/
# Copy all necessary files to the execution directory
cp /cavs/cmd/data1/users/<jdoe>/you-run-dirSandbox/bin/<all files required during
execution> .
# Use mpirun to run with 2 nodes for 60 hours for example
mpirun -np 4 ./your-mpi-program > output.log
# Move all files from the execution directory back to your project directory, e.g.
```

```
/cavs/cmd/data1/users/<jdoe>/your-run-dirSandbox/out/
mv *.* /cavs/cmd/data1/users/<jdoe>/your-run-dirSandbox/out/
#
echo "All Done!"
```

A.5 README.examples.txt

Package: **basicSkeleton**	Location: **examples**	File: **README.examples.txt**

```
Project Title  : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date   : MM-DD-YYYY [indicate the date of last modification]*required
Author(s)      : [list all people who wrote the code, added new features, etc] *required
Email Address  : [recommended]
Web Page       : [optional]
Description    : [one sentence description of the project] *required
Organization and
content of current
folder         : [describe the organization of files or directories in the current folder] *required

basicSkeleton *
|-- examples *
|   |-- README.examples.txt *
|   |-- inputParameters *
|   `-- outputResults *
```

A.6 README.src.txt

Package: **basicSkeleton**	Location: **src**	File: **README.src.txt**

```
Project Title  : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date   : MM-DD-YYYY [indicate the date of last modification]*required
Author(s)      : [list all people who wrote the code, added new features, etc] *required
Email Address  : [recommended]
Web Page       : [optional]
Description    : [one sentence description of the project] *required
Organization and
content of current
folder         : [describe the organization of files or directories in the current folder] *required

basicSkeleton *
`-- src *
    |-- README.src.txt
    |-- header.h
    |-- libst.a
    |-- libsh.so
    `-- main.c *
```

B. Template files included in 'skeleton'

B.1 README.txt

Package: **skeleton**	Location: **top level folder**	File: **README.txt**

Project Title : *required
Release Date : MMM-DD-YYYY [indicate the date of last modification]
Author(s) : [list all people who wrote the code, added new features, etc] *required
Email Address : [recommended]
Web Page : [optional]
Description : [one sentence description of the project] *required
Project structure: [Describe the organization of files or directories pertaining to the project. For a sample description, see below.]

This is the README for the Project <Skeleton>. README is a normal plain text file containing information about other files in the directory and is meant to be read before taking any significant action with the items in the directory.

```
skeleton                    *(top level project folder)
|-- README.txt               *(this file; information about the files and folders of the
                              project; to be read before taking any significant action
                              with the items in the directory)
|-- bin                     *("how-to" folder)
|   |-- Makefile             (compiler options for creating the executable version of the
                              project)
|   |-- README.bin.txt       (information about the files contained in this folder)
|   |-- postproc_script      (output data manipulation scripts)
|   |-- preproc_script       (input data manipulation scripts)
|   `-- skeleton.pbs         (sample script for remote execution, e.g on raptor or
                              maverick)
|-- doc                     *(documentation folder) specific instructions for a user to
                              install or use the project, user license, as well as what the
                              project does)
|   |-- INSTALL              *(instructions on parameters configuration , compiler
                              requirements, etc.)
|   |-- README.doc.txt       (information about the files contained in this folder)
|   |-- TechnicalGuide.pdf   *(theory behind the material model in the form of a technical
                              report, journal article, conference paper, etc.)
|   |-- UserGuide.pdf        *(information on the material model implementation in a
                              computer program (e.g. numerical methods used, precision,
                              etc.)
|   `-- UserLicence          *(type of license governing the usage or redistribution of the
                              project; all licenses - including free ones - must be explicitly
                              specified in this document)
|-- examples                *(illustrative project examples folder, each with input files,
```

```
                                  output files, sample runs, etc; if the examples and related
                                  data is too large to be included together in the codes
                                  repository, please describe the location of illustrative
                                  examples for this project)
|   |-- README.examples.txt      *(description of the examples, their purpose etc.)
|   |-- ex1                      *(project example 1)
|   |   |-- README.ex1.txt       *(details on the characteristics of example #1)
|   |   |-- input                *(input files needed by the project for example 1, e.g.
                                  constants, experiments, control variables, etc)
|   |   |   |-- constants.dat    *(input constants used by the model)
|   |   |   `-- experiments.dat  *(input from experiments used by the model)
|   |   `-- output               *(output files produced by the project for example 1, e.g. the
                                     results of the simulation such as atoms position, structures,
                                     log files, etc)
|   |       |-- models.dat        *(output data - constants, etc - produced by the model for
                                     ex1)
|   |       `-- output.log        *(ex1 execution log)
|   `-- ex2
|       |-- README.ex2.txt
|       |-- input
|       |   |-- constants.dat
|       |   `-- experiments.dat
|       `-- output
|           |-- models.dat
|           `-- output.log
|-- lib                           (folder with the software libraries needed for the project, e.g.
                                     meam. poems for LAMMPS)
|   |-- README.lib.txt            (instruction on creating/obtaining the libraries)
|   |-- lib.a                     (software static library required by the model)
|   |-- lib.so                    (software shared library required by the model)
|   `-- header.h                  (declarations of classes, subroutines, variables, and other
                                     identifiers)
`-- src                           *(folder with source files, e.g. '.h', '.c', '.f', '.f90' files; the files
                                     given in this template are illustrative of C source files – this
                                     folder is by no means limited to C-language source files)
    |-- README.src.txt            *(details on the source files and their inter-relations)
    |-- doWork.c                  (implementation material model routine - called by the main
                                     routine)
    |-- main.c                    (implementation of the routine "main", which calls all other
                                     routines; main is the routine called by the compiler)
    |-- readInput.c               (implementation of the routine responsible for reading the
                                     input data to be used during execution)
    `-- writeOutput.c             (implementation of the routine responsible for storing the
                                     data and the output produced during execution)
```

B.2 Makefile

Package: **skeleton**	Location: **bin**	File: **Makefile**

```
#######################
#Project Title  : [Give the name of your software project, e.g. DMGfit or UMAT] *required
#Release Date   : MM-DD-YYYY [indicate the date of last modification]*required
#Author(s)      : [list all people who wrote the code, added new features, etc] *required
#Email Address  : [recommended]
#Web Page       : [optional]
#Description    : [one sentence description of the project] *required
#######################
# Sample Makefile to build skeleton program
# Ensure that the C, C++ or Fortran compiler you use is in your path, by issuing the corresponding swsetup command in the terminal windows from which you will run your project.
# --- macros using Intel Compiler Suite
#### User configurable options #####
CC=icc
CXX=icpc
FC=ifort
MPIFC=mpif90 #or mpif77
### End User configurable options ###

#Preprocessor compiler flags, e.g. $(OPTFLAGS) $(INCLUDE_DIR) -DMPI_$(ARCH), INCLUDE_DIR is specified when include file is not in the current or system includes directory
CFLAGS= -O3 -I <include_dir_1>/include -I <include_dir_2>/include -I ../src/lib/
#Linker flags, e.g. $(INCLUDE_DIR) $(OPTFLAGS)
FFLAGS=
#System libraries are usually in /usr/lib or /lib
#User defied libraries, e.g. $(LIB_PATH) $(LIB_LIST), use -L<lib_path_1> when library <lib_path_1>/<lib_name_1> isn not in the current or system libraries directories
LIBS = -L<lib_path_1> -l<lib_name_1> -L<lib_path_2> -l<lib_name_2> -L../src/lib/ -lSkeleton

OBJECTS= skeleton.o readInput.o doWork.o writeOutput.o
# --- targets
default: skeleton

all:    skeleton

skeleton:   $(OBJECTS)
      $(CC)  -o skeleton  $(OBJECTS) $(LIBS)

skeleton.o: ../src/header.h main.c readInput.c doWork.c writeOutput.c
```

```
	$(CC) $(CFLAGS) -c main.c

readInput.o: ../src/header.h readInput.c
	$(CC) $(CFLAGS) -c readInput.c

doWork.o: ../src/header.h doWork.c
	$(CC) $(CFLAGS) -c doWork.c

writeOutput.o: ../src/header.h writeOutput.c
	$(CC) $(CFLAGS) -c writeOutput.c

# --- remove binary and executable files
clean:
	rm -f skeleton $(OBJECTS)
```

B.3 README.bin.txt

Package: **skeleton**	Location: **bin**	File: **README.bin.txt**

```
Project Title  : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date   : MM-DD-YYYY [indicate the date of last modification]*required
Author(s)      : [list all people who wrote the code, added new features, etc] *required
Email Address  : [recommended]
Web Page       : [optional]
Description    : [one sentence description of the project] *required
Organization and
content of current
folder         : [describe the organization of files or directories in the current folder] *required

skeleton *
|-- bin *
|   |-- Makefile *
|   |-- README.bin.txt
|   |-- postproc_script
|   `-- preproc_script
```

B.4 INSTALL

Package: **skeleton**	Location: **doc**	File: **INSTALL**

```
Project Title  : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date   : MM-DD-YYYY [indicate the date of last modification]*required
Author(s)      : [list all people who wrote the code, added new features, etc] *required
Email Address  : [recommended]
Web Page       : [optional]
Description    : [one sentence description of the project] *required
```

Organization and
content of current
folder : [describe the organization of files or directories in the current folder] *required

Notes on installing the software packages required by the project. This could include a specific compiler of a particular version number, math libraries such as BLAS, ATLAS, LAPACK, message passing libraries, etc.

B.5 README.doc.txt

Package: **skeleton**	Location: **doc**	File: **README.doc.txt**

Project Title : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date : MM-DD-YYYY [indicate the date of last modification]*required
Author(s) : [list all people who wrote the code, added new features, etc] *required
Email Address : [recommended]
Web Page : [optional]
Description : [one sentence description of the project] *required
Organization and
content of current
folder : [describe the organization of files or directories in the current folder] *required

```
skeleton *
|-- doc *
|   |-- INSTALL
|   |-- README.doc.txt
|   |-- TechnicalGuide.pdf *
|   |-- UserGuide.pdf *
|   |-- UserLicence
|   |-- skeleton.pbs
```

B.6 skeleton.pbs

Package: **skeleton**	Location: **doc**	File: **skeleton.pbs**

```
#!/bin/sh
#PBS -N skeleton
#PBS -q q16p96h@raptor
#PBS -l nodes=2:ppn=4
#PBS -l walltime=60:00:00
#PBS -M username@cavs.msstate.edu
#PBS -m abe
#PBS -j oe
#PBS -r n
#PBS -V
# Set the stack size to unlimited
```

```
ulimit -s unlimited
# Set the core size to zero
ulimit -c 0
# List all resource limits
ulimit -a
echo "I ran on:"
# Print the nodes on which the project will run
cat $PBS_NODEFILE
# Change to your execution directory to your project directory.
cd /cavs/cmd/data1/users/<jdoe>/your-run-dirSandbox/bin/
# For projects that write often to files during execution, copy the all files required for
execution to /data/lustre/
# (the project executable, the pbs script, the input files), submit the project from there, while
at the end of the
# execution copy all files back to your project directory.
# Change to your execution directory
cd /data/lustre/
# Copy all necessary files to the execution directory
cp /cavs/cmd/data1/users/<jdoe>/you-run-dirSandbox/bin/<all files required during
execution> .
# Use mpirun to run with 2 nodes for 60 hours for example
mpirun -np 4 ./your-mpi-program > output.log
# Move all files from the execution directory back to your project directory, e.g.
/cavs/cmd/data1/users/<jdoe>/your-run-dirSandbox/out/
mv *.* /cavs/cmd/data1/users/<jdoe>/your-run-dirSandbox/out/
#
echo "All Done!"
```

B.7 README.examples.txt

Package: **skeleton**	Location: **examples**	File: **README.examples.txt**

```
Project Title  : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date   : MM-DD-YYYY [indicate the date of last modification]*required
Author(s)      : [list all people who wrote the code, added new features, etc] *required
Email Address  : [recommended]
Web Page       : [optional]
Description    : [one sentence description of the project] *required
Organization and
content of current
folder         : [describe the organization of files or directories in the current folder] *required

skeleton *
|-- README.examples.txt *
|-- ex1 *
|   |-- README.ex1.txt
```

```
|   |-- input *
|   |   |-- constants.dat *
|   |   `-- experiments.dat
|   `-- output *
|       |-- models.dat
|       `-- output.log *
`-- ex2
    |-- README.ex2.txt
    |-- input
    |   |-- constants.dat
    |   `-- experiments.dat
    `-- output
        |-- models.dat
        `-- output.log
```

B.8 README.examples.txt

Package: **skeleton**	Location: **examples**	File: **README.examples.txt**

Project Title : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date : MM-DD-YYYY [indicate the date of last modification]*required
Author(s) : [list all people who wrote the code, added new features, etc] *required
Email Address : [recommended]
Web Page : [optional]
Description : [one sentence description of the project] *required
Organization and
content of current
folder : [describe the organization of files or directories in the current folder] *required

```
skeleton *
|-- lib
| |-- README.lib.txt
| |-- header.h
| |-- libsh.so
  `-- libst.a
```

B.9 README.src.txt

Package: **skeleton**	Location: **src**	File: **README.src.txt**

Project Title : [Give the name of your software project, e.g. DMGfit or UMAT] *required
Release Date : MM-DD-YYYY [indicate the date of last modification]*required
Author(s) : [list all people who wrote the code, added new features, etc] *required
Email Address : [recommended]
Web Page : [optional]
Description : [one sentence description of the project] *required

Organization and
content of current
folder : [describe the organization of files or directories in the current folder] *required

```
skeleton *
|-- src *
|   |-- README.src.txt
|   |-- doWork.c
|   |-- main.c *
|   |-- readInput.c
    `-- writeOutput.c
```