NIST Special Publication 1018-5

Fire Dynamics Simulator (Version 5) Technical Reference Guide

Volume 4: Configuration Management Plan

Kevin McGrattan
Simo Hostikka
Jason Floyd
Bryan Klein

In cooperation with:
VTT Technical Research Centre of Finland

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Special Publication 1018-5

Fire Dynamics Simulator (Version 5) Technical Reference Guide

Volume 4: Configuration Management Plan

Kevin McGrattan
Bryan Klein
Fire Research Division
Building and Fire Research Laboratory

Simo Hostikka
VTT Technical Research Centre of Finland
Espoo, Finland

Jason Floyd
Hughes Associates, Inc.
Baltimore, Maryland, USA

November 19, 2008
FDS Version 5.2
SVNRepository Revision : 2588

U.S. Department of Commerce
Carlos M. Gutierrez, Secretary

National Institute of Standards and Technology
James M. Turner, Acting Director

National Institute of Standards and Technology Special Publication 1018-5
Natl. Inst. Stand. Technol. Spec. Publ. 1018-5, ?? pages (October 2007)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 2007

For sale by the Superintendent of Documents, U.S. Government Printing Office
Internet: bookstore.gpo.gov – Phone: (202) 512-1800 – Fax: (202) 512-2250
Mail: Stop SSOP, Washington, DC 20402-0001

Disclaimer

About the Authors

Kevin McGrattan is a mathematician in the Building and Fire Research Laboratory of NIST. He received a bachelors of science degree from the School of Engineering and Applied Science of Columbia University in 1987 and a doctorate at the Courant Institute of New York University in 1991. He joined the NIST staff in 1992 and has since worked on the development of fire models, most notably the Fire Dynamics Simulator.

Simo Hostikka is a Senior Research Scientist at VTT Technical Research Centre of Finland. He received a master of science (technology) degree in 1997 and a doctorate in 2008 from the Department of Engineering Physics and Mathematics of the Helsinki University of Technology. He is the principal developer of the radiation and solid phase sub-models within FDS.

Jason Floyd is a Senior Engineer at Hughes Associates, Inc., in Baltimore, Maryland. He received a bachelors of science degree and a doctorate from the Nuclear Engineering Program of the University of Maryland. After graduating, he won a National Research Council Post-Doctoral Fellowship at the Building and Fire Research Laboratory of NIST, where he developed the combustion algorithm within FDS. He is currently funded by NIST under grant 60NANB5D1205 from the Fire Research Grants Program (15 USC 278f). He is the principal developer of the multi-parameter mixture fraction combustion model and control logic within FDS.

Bryan Klein is an Information Technology Specialist in the Building and Fire Research Laboratory of NIST. Before coming to NIST, Bryan worked for five years with Western Fire Center, Inc., performing a wide range of activities including fire modeling, data acquisition programming, and quantitative fire measurements. His current focus is on FDS development and user support, along with experimental model validation work.

Acknowledgments

The development and maintenance of the Fire Dynamics Simulator has been made possible through a partnership of public and private organizations, both in the United States and abroad. Following is a list of contributors from the various sectors of the fire research, fire protection engineering and fire services communities:

FDS is supported financially via internal funding at both NIST and VTT, Finland. In addition, support is provided by other agencies of the US Federal Government, most notably the Nuclear Regulatory Agency Office of Research. The US NRC Office of Research has funded key validation experiments, the preparation of the FDS manuals, and the development of various sub-models that are of importance in the area of nuclear power plant safety. Special thanks to Mark Salley and Jason Dreisbach for their efforts and support. The Office of Nuclear Material Safety and Safeguards, another branch of the NRC, has supported modeling studies of tunnel fires under the direction of Chris Bajwa and Allen Hansen.

Another source of support for FDS development has been the Microgravity Combustion Program of the National Aeronautics and SpaceAdministration (NASA).

Originally, the basic hydrodynamic solver was designed by Ronald Rehm and Howard Baum with programming help from Darcy Barnett, Dan Lozier and Hai Tang of the Computing and Applied Mathematics Laboratory (CAML) at NIST, and Dan Corley of the Building and Fire Research Laboratory (BFRL). Jim Sims of CAML developed the original visualization software. The direct pressure solver was written by Roland Sweet of the National Center for Atmospheric Research (NCAR), Boulder, Colorado. Kuldeep Prasad added the multiple-mesh data structures, paving the way for parallel processing. Charles Bouldin devised the basic framework of the parallel version of the code.

At NIST, Glenn Forney developed the visualization tool Smokeview that not only made the public release possible, but it also serves as the principal diagnostic tool for the continuing development of FDS.

William Grosshandler and Tom Cleary, both currently at NIST, developed an enhancement to the smoke detector activation algorithm, originally conceived by Gunnar Heskestad of Factory Mutual. Steve Olenick of Combustion Science and Engineering (CSE) implemented the smoke detector model into FDS.

William Grosshandler is also the developer of RadCal, a library of subroutines that have been incorporated in FDS to provide the radiative properties of gases and smoke.

Prof. Nick Dembsey of Worcester Polytechnic Institute (WPI) has provided valuable feedback about the pyrolysis model used within FDS.

Professor Fred Mowrer of the University of Maryland provided a simple of model of gas phase extinction to FDS.

Chris Lautenburger of the University of California, Berkeley, and Jose Torero and Guillermo Rein of the University of Edinburg. have provided valuable insight in the development of the solid phase model.

Contents

Chapter 1

Introduction

This plan is applicable to Fire Dynamics Simulator (FDS). This document follows guidelines set forth in IEEE STD 828-1998, Standard for Software Configuration Management Plans.

1.1 Purpose

The purpose of this document is to identify and describe the overall policies and methods for CM to be used for FDS. This plan will be updated as the necessity arises.

This CM Plan (CMP) will establish and provide the basis for a uniform and concise CM practice for FDS The primary intention of this CMP is to provide information on the CM policies and methods to be adopted and implemented by the FDS development team.

The primary objective of the FDS CMP is twofold. First is to establish the process for tracking and controlling FDS requirements and enhancements. Second is to establish a change management status and reporting system for officially released FDS versions, documentation, and associated input data files.

1.2 Scope

This plan covers the Modeling and Simulation (M&S) configuration management activities associated with FDS.

1.3 Fire Dynamics Simulator (FDS) Model Description

FDS is a Computational Fluid Dynamics (CFD) model of fire-driven fluid flow. The model solves numerically a form of the Navier-Stokes equations appropriate for low-speed, thermally-driven flow with an emphasis on smoke and heat transport from fires. The partial derivatives of the conservation equations of mass, momentum and energy are approximated as finite differences, and the solution is updated in time on a three-dimensional, rectilinear grid. Thermal radiation is computed using a finite volume technique on the same grid as the flow solver. Lagrangian particles are used to simulate smoke movement, sprinkler discharge, and fuel sprays.

Chapter 2

CM Management

Configuration management is defined as part of the project management system required for the technical and administrative direction of the project. This section describes the relationship of configuration management to the project management structure as it relates to controlling the configuration of FDS.

2.1 FDS CM Manager Responsibilities

The CM Manager for FDS is Kevin McGrattan, Fire Modeling Group Leader of the Building and Fire Research Laboratory at the National Insitute of Standards and Technology located in Gaithersburg, MD. The responsibilities of the CM Manager include:

1. Maintaining configuration control over the officially released software and data for FDS.
2. Final approval of changes to the officially released software and data for FDS.
3. Developing and maintaining the FDS Configuration Management Plan (CMP).
4. Maintaining all officially released software and data for the FDS configuration baselines and documenting the baseline contents.
5. Authorize individuals to become members of the software development team (SDT).
6. Authorize establish of software baselines and identification of Computer Software Configuration Items (CSCIs)
7. Recording new FDS change proposals and tracking them to completion

The CM Manager may designate a member of the SDT to be responsible for one or more CM Manager responsibilties.

2.2 FDS Software Development Team Member Responsibilities

FDS Development Team Members are those individuals allowed to make changes to the official FDS CSCI repository. Their specific responsibilities include:

1. Represent interests of project management and all groups who may be affected by changes to the software baselines.

2. Assign, review, and provide for disposition of action items.

3. Determine or review the availability of resources required to complete the proposed change or modification, assess the impact of the proposed change upon the system, examine cost considerations, and determine the impact of the change on development schedules.

Chapter 3

CM Process

The IRM Suite Configuration Management (CM) process consists of four primary functions: configuration identification, configuration control, configuration status accounting, and configuration audits. The responsibilities of each CM function are listed in the paragraphs below, as they relate to controlling the configuration of the IRM Suite.

3.1 Configuration Identification

Configuration identification consists of placing significant FDS related files under configuration control. These products are called Computer Software Configuration Items (CSCIs). CSCIs are defined as a collection of software, data, or documentation required for an end use function and designated by the Project Engineer for separate configuration management. The CSCIs include:

1. Compiled FDS executables
2. FDS source code files
3. FDS documentation
4. Input files for software testing, verification testing, and validation testing
5. Experimental data files used for validation testing
6. Scripts and post-processing utilities used for software testing
7. FDS web pages and wikis

3.1.1 Maintaining CSCIs

All CSCIs are maintained using the online service GoogleCode. GoogleCode is a free service offered by Google to support configuration management and software development for open source applications. GoogleCode uses Subversion (SVN) for revision control. Under this system a centralized repository containing all CSCIs resides on a GoogleCode server. Versions are indentified by a version number which represents the version of the entire repository rather than of a specific file (i.e. anytime a change is made to the repository all files are incremented in version number). A record of version number when a specific file was last changed is maintained.

As an open source program, any individual can obtain a copy of the repository or retrieve specific versions repository. Only CM Manager approved members of the SDT; however, can make committ changes to the repository.

The current location of the FDS repository is `http://fds-smv.googlecode.com/svn/trunk/`

3.1.2 Identifying Specific CSCIs in Compiled Executables

At the start of an FDS simulation FDS writes header information to the Smokeview output file, FDS output file, and the FDS log file. This header information contains the version of FDS used to perform that simulation. While each release is tagged with a specific version number (e.g. 5.0.1), there may be many commits of source code, documentation, or other files to the SVN repository before a new version is released with an incremented version number. Thus, if a developer or a user who performs their own compilation between baseline releases discovers an error, the version number written to the output files may not be sufficient to identify the specific set of source code files used. Rather one would need to know the SVN revision number of the most recently committed source file.

This is accomplished by using source file tagging. In each source file a series of character parameter strings are defined. For example the strings in main.f90 are:

```
CHARACTER(255), PARAMETER :: mainid='$Id: FDS_5_Configuration_Management_Plan.tex 2588 2008-11-03 13
CHARACTER(255), PARAMETER :: mainrev='$Revision: 2588 $'
CHARACTER(255), PARAMETER :: maindate='$Date: 2008-11-03 08:54:38 -0500 (Mon, 03 Nov 2008) $'
```

The string contains the name of the source file, the version of the file (this reflects only the source file version and not the overall release version number), the date and time of the file version, and the person who checked the file in to the archive. Upon compiling, these strings will be stored in the executable file. The user is then capable of searching the executable file, for example, for strings beginning with `$Id:`. This will result in a list of all source files compiled and their version. Within the SVN archive any specific version of a source file can be extracted and differences between versions can be determined.

Within the source code a series of subroutines were created, one per source file, which parses the `$Id:` in each file and extracts from it the SVN revision number. Each of these subroutines is called at the start of an FDS run and the largest (and hence most recent) revision number is determined. This number is written along with the FDS version number to the output header information. A user can now identify specifically the source code used for a particular compilation of FDS when reporting an error.

3.2 Configuration Control

Configuration control is the exercising of established procedures to classify, approve or disapprove, release, implement, and confirm changes to officially released models and databases. The CM Manager is the ultimate is the gatekeeper for all changes required to an established officially released FDS Suite baseline. These change requests could include:

1. Modifications of defined requirements

2. Corrections for deficiecies

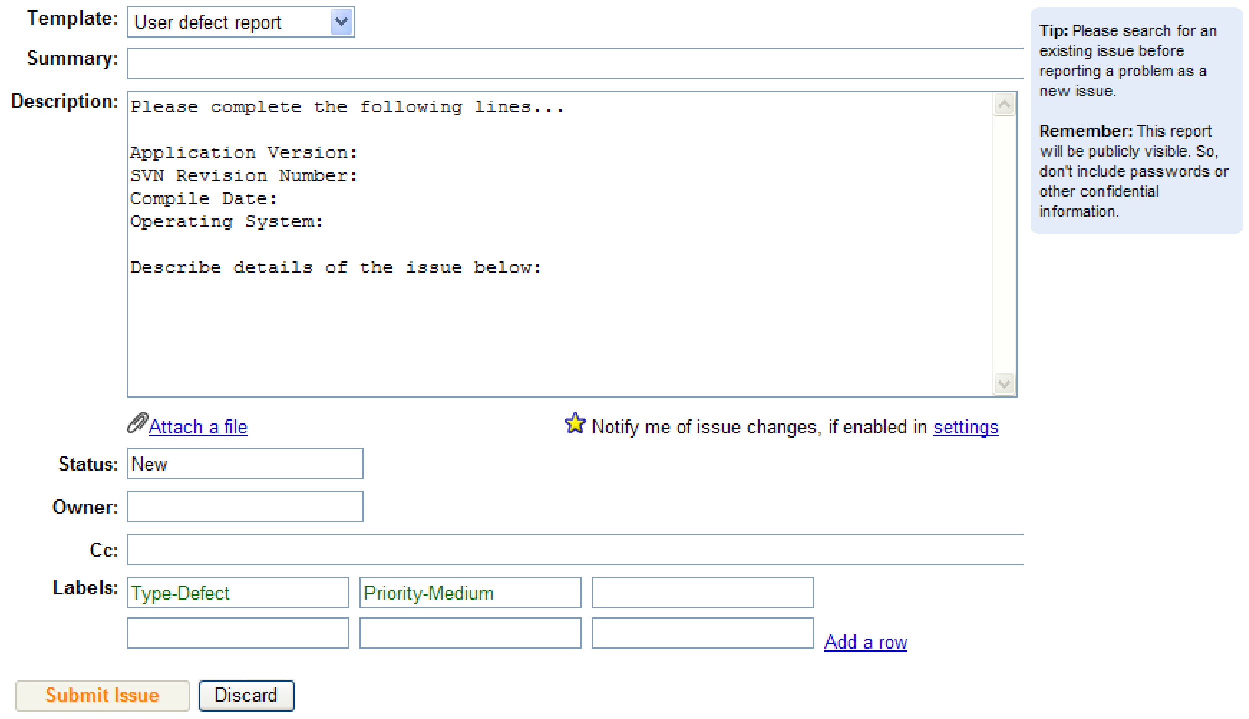
3. Changes to existing capabilities

4. Model enhancements

3.3 Software Changes

3.3.1 Creating a Change Request

Change requests are submitted using the FDS Issue Tracker. The Issue Tracker is an online service that is part of GoogleCode. The current location of the Issue Tracker is `http://code.google.com/p/fds-smv/issues/list`. A change request is initiated by opening a new issue. The issue report contains the baseline identification (version number, complie date, and SVN revision number), operating system, and a description of the defect or ehancement request. Input files or other supporting documentation can be attached.



Template: User defect report

Summary:

Description:

```
Please complete the following lines...

Application Version:
SVN Revision Number:
Compile Date:
Operating System:

Describe details of the issue below:
```

Tip: Please search for an existing issue before reporting a problem as a new issue.

Remember: This report will be publicly visible. So, don't include passwords or other confidential information.

Attach a file

Notify me of issue changes, if enabled in settings

Status: New

Owner:

Cc:

Labels: Type-Defect, Priority-Medium

Add a row

Figure 3.1: **Screenshot of issue tracker reporting form**

If the issue is opened by a user, it will be given a status of 'New' until it is reviewed by a developer. If the issue is opened by a developer, the developer can immeadiate assign a status and an owner.

3.3.2 Processing a Change Request

A change request may be evaluated by any member of the development team.

If the request duplicates an existing request, then the issue status is changed to `Duplicate` and the requestor is sent the issue number of the existing request.

If the request is for an enhancement to FDS, then then the issue type is changed to `Type-Ehancement` and the component type is declared. The request is evaluated for suitability. If it is a valid enhancement request, then the status is changed to `Accepted`. If the request is not to be addressed under the next baseline, then the status will be changed to `OnHold`. If the request is denied, then the status is changed to `WontFix`. An accepted request will be assigned to a developer.

If the request is identifying a defect in FDS, then the issue type is changed to `Type-Defect` and the component type is declared. The defect is evaluated for validity. If insufficient information has been provided in the request, then the status is changed to `MoreInfo` and a description of the additional information required is sent to the requestor. If the defect is due to user error or is the intended function of FDS, then the status is changed to `Invalid`. If the defect is valid, then the request reviewer will change the status to `Accepted` and assign the request to a developer.

Once a change request has be addressed by a developer and changes submitted to the repositoty, the request status is changed to `Fixed`. Once either the requestor or another developer has verified that the changes address the original request, then the status is changed to `Verified`.

3.3.3 Committing Changes

Once a developer has addressed a change request, the changed CSCIs are committed to the SVN repository. A description of the changes will be added to the SVN change log. This description should first identify the primary component being changed (for example: FDS Source or FDS Documentation). This component identification will be followed by a brief summary of the changes made. The issue identifier will be included as part of that brief description.

3.3.4 Change Verification

Once a change has been committed and the issue tracker updated to reflect that the issue has been `Fixed`, the changes will be verified. Verification can be done by either the requester of the change or by another developer. Once the changes have been verified to solve the problem reported in the issue, the issue status will be changed to `Verified`. At this point the issue is closed.

3.4 Establishing Baselines

The decision to establish a new baseline is made by the CM Manager. The new baseline is named, tested, and the released based on the procedures that follow.

3.4.1 Naming Baselines

New FDS baselines are identified using a specific naming convention. Baselines are identified as app.#.#.#_os. Where app is the application name (fds), #.#.# is a version number, and _os is the operating system that the baseline executable was compiled for. The version number consists of three integers where the first number is the *major* release, the second is the *minor* release, and the third is the *maintenance* release. Major releases occur every few years, and as the name implies dramatically changed functionality of the model. Minor releases occur every few months, and may cause minor changes in functionality. Release notes can help you decide whether the changes should effect the type of applications that you typically do. Maintenance releases are either just bug fixes or the addition of minor enhancements (such as a new output quantity), and should not affect code functionality.

3.4.2 Baseline Software Testing

Each proposed baseline release will undergo software testing. Three suites of test cases exists: a functional test suite, a verification test suite, and a validation test suite. Testing will depend upon the type of baseline release: maintenance, minor, or major.

Each maintenance release will be tested with the functional test suite. It will be verified that the proposed baseline successfully executes all of the functional test cases.

Each minor release will be tested with the functional test suite and the verification test suite. It will be verified that the proposed baseline successfully executes all of the functional and verification test cases. The verification manual will be updated with the results.

Each major release will be tested with all three test suites. It will be verified that the proposed baseline successfully executes all of the functional and verification test cases. The verification manual will be updated with the results. The results of the validation cases will be evaluated to ensure that the predictive performance of the proposed baseline as either remained equivalent or improved. The validation manual will be updated.

3.4.3 Release of a New Baseline

Following successful completion of the required baseline testing, a baseline can be released. Prior to release, the version identification information within the FDS source code will be updated to reflect the new baseline. FDS documentation will be updated to reflect the new baseline. The baseline will be compiled and new executable files or installation packages will be placed on the FDS download site. Prior baselines will be deprecated. The current FDS dowload site is `http://code.google.com/p/fds-smv/downloads/list`.

Chapter 4

Coding Standard

4.1 Indentation

Each new level of source code will be indented by three spaces. Indentation will occur following a `DO`, `IF ELSE`, `ELSEIF`, `SELECT CASE`, or `CASE` statement. Comment lines are not indented.

4.2 Source File Organization

Each source file in FDS shall contain subroutines and functions specific to narrow subsegment or physical submodel of the software. Currently the FDS source consists of the following files:

4.3 Explicit Variable Typing

All variables in FDS will be explicitly typed. To aid in this, all source code files will contain the expression `IMPLICIT NONE`. All `REAL` numbers in FDS will have their kind indicated. For example:

```
REAL(EB) :: x
x = 0._EB
```

and not:

```
REAL(EB) :: x
x = 0.
```

4.4 Loop and Branch Naming

Any `DO` loop, `IF` block, or `SELECT CASE` block that continues for more than 20 lines will be named. For example:

```
LOOPNAME: DO
   more than 20 lines of code
END DO LOOPNAME
```

and not:

```
DO
   more than 20 lines of code
END DO
```

Table 4.1: FDS Source Code Files

File Name	Filetype	Description
cons.f90	Fortran	Definition of physical constants, parameters, global variables
ctrl.f90	Fortran	Control functions
devc.f90	Fortran	Definition of device, control, and property variables
divg.f90	Fortran	Divergence computation
dump.f90	Fortran	Output processing
evac.f90	Fortran	Evacuation model
fire.f90	Fortran	Combustion model
func.f90	Fortran	Miscellaneous functions
ieva.f90	Fortran	Defintion of evacuation variables and evacuation functions
init.f90	Fortran	Intialization
irad.f90	Fortran	Radiation transport variables, RADCAL, and Mie solver
isob.c	C	Functions for generating iso surfaces
main.f90	Fortran	Main timestepping routine
main_mpi.f90	Fortran	Main timestepping routine for parallel version of FDS
mesh.f90	Fortran	Defintion of mesh variables
part.f90	Fortran	Droplet and particle mass and energy transport
pois.f90	Fortran	CRAYFISHPAK Poisson solver
prec.f90	Fortran	Kind defintions
pres.f90	Fortran	Pressure solver
radi.f90	Fortran	Radiation transport solver and initialization
read.f90	Fortran	Input processsing
smvv.f90	Fortran	Interface routines for Smokeview
turb.f90	Fortran	Module for testing turbulence models
type.f90	Fortran	Variable type defintions
vege.f90	Fortran	WUI vegetation model
velo.f90	Fortran	Velocity solver
wall.f90	Fortran	Wall heat transfer and boundary conditions

4.5 Named Constants

Named integer constants shall be used rather than integers or text strings for controlling code execution. This is done to enhance readability and improve execution speed by avoiding string comparisons. For example:

```
INTEGER :: OPTIONS
INTEGER, PARAMETER :: OPTION1=1, OPTION2=2

SELECT CASE (OPTIONS)
   CASE(OPTION1)
      ...
   CASE(OPTION2)
      ...
END SELECT
```

and neither:

```
INTEGER :: OPTIONS

SELECT CASE (OPTIONS)
   CASE(1)
      ...
   CASE(2)
      ...
END SELECT
```

nor:

```
CHARACTER(10):: OPTIONS

SELECT CASE (OPTIONS)
   CASE('OPTION1')
      ...
   CASE('OPTION2')
      ...
END SELECT
```

4.6 Redundant Code

Where a block of code must be used repeatedly in different subroutines a `FUNCTION` or `SUBROUTINE` shall be created. For example, there are a number of places where FDS must compute the average molecular weight of a gas. Rather than having that block of code repeated in each location, a single function was created to perform this function. This accomplishes two goals. By using a function or subroutine, a meaningful name can be used to represent the function being performed which improves source code readibilty. If the form of the function must be changed do to a bug fix or an enhancement, then having only one location to fix reduces that chance of the error that would occur if one of multiple locations was missed.