Generating Parametric Analyses

and Using Variables Within Your FDS File

xFDS version 0.3

Brian Cohan PE

PBD Tools
1108 East Main Street, Ste. 906-1944
Richmond, VA 23219 USA
e-mail: briancohan@pbd.tools

ABSTRACT

Fire modeling projects often require studying a variety of configurations, such as different mesh resolutions, smoke exhaust capacity, and more. These analyses often require the modeler to create numerous input files or custom tools to generate an array of scenarios to process. Imagine if you could generate the FDS models and dynamically set values for each scenario from a single FDS file.

This paper introduces xFDS, an open-source command-line tool built to accelerate your fire modeling workflow. xFDS uses a templating system that allows the user to create variables and logic to dictate which lines are included in each scenario. Furthermore, xFDS allows users to run their models locally under various FDS versions without having to install FDS on their machines.

INTRODUCTION

Fire Dynamics Simulator (FDS) is the most common software used for modeling purposes (Wade, Nilsson, Baker, & Olsson, 2021). There are numerous tools available to help with generating FDS models, ranging from spreadsheets and custom scripts to commercial products. While these tools simplify the modeling process, calculations used to determine the FDS inputs are often in a different file, leading to copy-and-paste errors or failure to update certain models correctly. xFDS provides a general purpose framework for developing models in a way that allows the modeler to document their decisions and calculates FDS input values when generating the final models.

STANDARD FDS SYNTAX OVERVIEW

To run an FDS model, the user must supply the program with a single text file that defines all the necessary information for that model (McGrattan, Hostikka, Floyd, McDermott, & Vanella, 2022). Each record is defined using namelist formatted records that start with an ampersand **&** and terminate with a slash **/**. Table 1 lists some of the most commonly used namelist groups.

Table 1 Frequently used Namelist Groups

Namelist	Description	Usage
CTRL	Control Function Parameters	Define control logic to trigger events during the simulation.
DEVC	Device Parameters	Extract useful information from the simulation.
MESH	Mesh Parameters	Define the spatial bounds of the model
OBST	Obstruction	Define the geometry needed for the model.
REAC	Reaction Parameters	Define how combustion works in the model.
SURF	Surface Properties	Define boundary conditions such as materials, burners, and exhaust.
TIME	Simulation Time	Define the temporal bounds of the model.
VENT	Vent Parameters	Define regions where surface conditions are applied.

Each namelist has different parameters that can be defined. For example, the code in Figure 1 below defines a small burner in a room. Line 1 defines the spatial bounds, 6.096 m (20 ft) by 3.658 m (12 ft) and 2.438 m (8 ft) high. Line 2 states that the simulation should run for one minute. Finally, lines 3 through 5 specify a 1000 kW propane fire in a square 1.5 m^2 burner on the floor.

```
1  &MESH XB=  0.000,  6.096,  0.000,  3.658,  0.000,  2.438, IJK=41,24,16/
2  &TIME T_END=60/
3  &REAC FUEL='PROPANE'/
4  &SURF ID='BURNER', COLOR='RED', HRRPUA=666.67/
5  &VENT XB=  0.638,  1.862,  1.138,  2.362,  0.000,  0.000, SURF_ID='BURNER'/
```

Figure 1 SimpleBurner.fds

Challenges in Reading FDS Code

The challenge with the code in Figure 1 is that it is not immediately apparent how big the burner is or how large the fire is. As a reviewer, to verify the inputs are correct, the area (A) must be calculated as seen in equation 1, Next, the area is multiplied by the heat release rate per unit area (HRRPUA, or $\dot{q}''$) to determine the total heat release rate ($\dot{Q}$) as seen in equation 2.

$$A = (xmax - xmin) * (ymax - ymin) = 1.224\ m * 1.224\ m = 1.5\ m^2 \qquad \text{eq. 1}$$

$$\dot{Q} = \dot{q}''A = 666.67\ \frac{kW}{m^2} * 1.5\ m^2 = 1000\ kW \qquad \text{eq. 2}$$

In addition to determining the fire size, the resolution of the mesh is not immediately apparent, nor is it easy to determine if the cells are uniform in each direction. The IJK parameter on line 1 tells the user that the model has 41, 24, and 16 cells in the x, y, and z directions, respectively. To determine the resolution, divide the extent of the mesh by the number of cells in that direction, which results in a cell size of 0.15 m in each direction.

While some users put comments in the FDS input files, there is no guarantee that the comments accurately reflect the records. Additionally, users often leverage tools to generate their fire models. These tools range from spreadsheets and programming scripts to dedicated graphical user interfaces. While these tools may contain additional documentation of the decisions made during model development, they may not be available to a reviewer. Furthermore, spreadsheets and programming scripts might not be reusable on different projects depending on how they were developed. xFDS allows these decisions to be documented in the master file that generates scenarios.

USING TEMPLATES TO GENERATE FDS FILES

Templates are files that contain placeholders where values can be substituted at compilation time. For example, the code in Figure 2 defines the heat release rate per unit area (HRRPUA). It requires two variables to be defined: `hrr` and `area`. This syntax allows the user to define the desired heat release rate and the burner area to ensure that the HRRPUA is calculated correctly without requiring external tools. The variables can be defined in the template or a configuration file that describes one or more scenarios to generate. Both these techniques, as well as how to render the templates, are described in the sections below.

```
1  &SURF ID='FIRE', HRRPUA={{ hrr / area }}/
```

Figure 2Defining HRRPUA with Variables

WRITING A FDS TEMPLATE

Defining and using variables

xFDS uses the Jinja templating system (Pallets Project, 2022) for generating models. Variables are defined using curly braces and percent signs along with the `set` keyword as follows: `{% set <variable> = <value> %}`. A set of double curly braces `{{ <variable> }}` is used to indicate where text should be replaced. Lines 1 and 2 in Figure 3 define the `hrr` and `area` variables for calculating HRRPUA. This method clarifies to the reader that the heat release rate should be 1000 kW over a 1.5 m^2 area. On line 3, the double curly braces tell xFDS to use the `hrr` and `area` variables to calculate the value for HRRPUA. This code would generate the FDS record on Line 4 of Figure 1. Variables can also be defined in a configuration file, as described later in this report.

```
1  {% set hrr = 1000 %}
2  {% set area = 1.5 %}
3  &SURF ID='FIRE', COLOR='RED', HRRPUA={{ hrr / area }}/
```

Figure 3 Setting Variables to Calculate HRRPUA

Reusing code by Defining Macros

The Jinja templating system allows users to define macros that produce a set of FDS records. The syntax for defining a macro is `{% macro <macro_name>(parameters) %}<macro body>{% endmacro %}` as seen in Lines 1-4 of Figure 4. This macro takes x, y, and z locations followed by a unique name for that location. The visibility and temperature devices are generated every time the macro is called.

The tenability macro is called on lines 7-8 with different x coordinates and names. The output for these lines is seen in Figure 5. Note how the elevation of the devices is specified on line 6 of Figure 4 as 6 feet above the floor. This line uses a filter called convert, which takes the value 6 and says the value was specified in feet and xFDS needs to be converted to meters.

```
{% macro tenability(x, y, z, name) %}
&DEVC ID='VIS_{{ name }}', XYZ={{ (x, y, z)|xyz }}, QUANTITY='VISIBILITY'/
&DEVC ID='TMP_{{ name }}', XYZ={{ (x, y, z)|xyz }}, QUANTITY='TEMPERATURE'/
{% endmacro %}

{% set z = 6|convert('ft', 'm') %}
{{ tenability(0, 0, z, "Point_1") }}
{{ tenability(1, 0, z, "Point_2") }}
```

Figure 4 Defining Macros in a Template

```
&DEVC ID='VIS_Point_1', XYZ=  0.000,  0.000,  1.829, QUANTITY='VISIBILITY'/
&DEVC ID='TMP_Point_1', XYZ=  0.000,  0.000,  1.829, QUANTITY='TEMPERATURE'/
&DEVC ID='VIS_Point_2', XYZ=  1.000,  0.000,  1.829, QUANTITY='VISIBILITY'/
&DEVC ID='TMP_Point_2', XYZ=  1.000,  0.000,  1.829, QUANTITY='TEMPERATURE'/
```

Figure 5 Output from Calling a Macro

Using Loops to Generate Items

Another way to quickly generate many records is to use a loop. The syntax for iterating through a loop is `{% for <variable> in <iterable> %}<records>{% endfor %}` where the iterable is a list of values or a function that generates values. Loops can also be nested. Figure 6 demonstrates creating slices for two different quantities at three different elevations. As seen in Figure 7, there are 6 different slices defined corresponding to each elevation and quantity.

```
{% for z in [3, 6, 9] %}
{% for qty in ['TEMPERATURE', 'VISIBILITY'] %}
&SCLF PBZ={{ z }}, QUANTITY='{{ qty }}'/
{% endfor %}
{% endfor %}
```

Figure 6 Generating Slices with Loops

```
&SCLF PBZ=3, QUANTITY='TEMPERATURE'/
&SCLF PBZ=3, QUANTITY='VISIBILITY'/
&SCLF PBZ=6, QUANTITY='TEMPERATURE'/
&SCLF PBZ=6, QUANTITY='VISIBILITY'/
&SCLF PBZ=9, QUANTITY='TEMPERATURE'/
&SCLF PBZ=9, QUANTITY='VISIBILITY'/
```

Figure 7 Slices Generated by Loops

Conditional Rendering

The last major advantage of using a template is enabling and disabling certain records based on variables. The syntax for an if statement is `{% if <condition1> %} <records_if_condition1_is_true> {% elif <condition2> } %<records_if_ condition2_is_true> {% else %}<default records>{% endif %}`. The `elif` block is optional, but the template can also support multiple `elif` blocks.

Figure 8 shows how different fires can be selected depending on the value of the `room` variable. For example, if the configuration specifies three scenarios, one for each room, this one section in the template ensures the correct design fire is used for each model.

```
1  {% if room == 'Room 1' %}
2  &VENT SURF_ID='FIRE', XB=.../
3  {% elif room == 'Room 2' %}
4  &VENT SURF_ID='FIRE', XB=.../
5  {% else %}
6  &VENT SURF_ID='FIRE', XB=.../
7  {% endif %}
```

Figure 8 Using If Statements to Render Correct Fire Location

GENERATING MULTIPLE MODELS FROM A TEMPLATE

xFDS is a command line utility that leverages the template system. A configuration file must be specified to generate models from a template. The configuration file needs to be called `pbd.yml`, written in YAML syntax (The YAML Project, 2021), and at the root of the fire modeling project along with the modeling template. The `xfds render` command identifies the configuration file and renders all specified models within the project. The configuration file is required even if there is only a single model. Figure 9 shows the minimum amount of information needed to render the template. The `name` parameter tells xFDS how to name all the output files, while the files parameter tells xFDS all the files to use for the model. In this example, the configuration tells xFDS that there is a template file called `simple_atrium.fds`, and the basename of the output file should also be `simple_atrium`. xFDS adds the file extension based on the input file. The reason for this is made apparent in the Running FDS Files on a section below.

```
1  xfds:
2    render:
3      - name: simple_atrium
4        files:
5          - simple_atrium.fds
```

Figure 9 Minimal xFDS Configuration File

Specifying Variables and Parameters

Variables used in the template file can be specified under the `variables` keyword, as seen in lines 6-8 in Figure 10. Any variables defined in this section are available in the template file but are not used to generate additional scenarios. In contrast, variables specified under the `parameters` keyword tell xFDS what scenarios to generate. For example, on lines 9-11 in Figure 10, two parameters are specified: `cfm` representing the total exhaust capacity for an atrium, and `mua_perc`, which specifies the percentage of the total exhaust to be supplied by makeup air. xFDS produces 12 output files, as seen in Table 2, to cover the three exhaust rates and four makeup air percentages. Note how the parameters are also used in the `name` field to tell xFDS how to name each model.

```
1  xfds:
2    render:
3      - name: simple_atrium_{{ cfm }}_{{ mua_perc }}
4        files:
5          - simple_atrium.fds
6        variables:
7          hrr: 1000 # kW
8          area: 1.5 # m
9        parameters:
10         cfm: [100000, 125000, 150000]  # cfm
11         mua_perc: [0, 85, 90, 95] # %
```

Figure 10 xFDS Configuration File Specifying Scenarios

Table 2 List of Models Generated by xFDS

Model Name	Exhaust Capacity	Makeup Air Percentage
simple_atrium_100000_0.fds	100,000 cfm	0 %
simple_atrium_100000_85.fds	100,000 cfm	85 %
simple_atrium_100000_90.fds	100,000 cfm	90 %
simple_atrium_100000_95.fds	100,000 cfm	95 %
simple_atrium_125000_0.fds	125,000 cfm	0 %
simple_atrium_125000_85.fds	125,000 cfm	85 %
simple_atrium_125000_90.fds	125,000 cfm	90 %
simple_atrium_125000_95.fds	125,000 cfm	95 %
simple_atrium_150000_0.fds	150,000 cfm	0 %
simple_atrium_150000_85.fds	150,000 cfm	85 %
simple_atrium_150000_90.fds	150,000 cfm	90 %
simple_atrium_150000_95.fds	150,000 cfm	95 %

Adding Information to Scenarios

Sometimes it is advantageous to add information to specific scenarios without having to add a new parameter to the models. For example, when the makeup air percentage is 0%, that might imply that mechanical makeup air should not be supplied. Instead, doors and windows should be used to provide the makeup air.

Lines 12-15 of Figure 11 tell xFDS to include `open_doors = true` when `mua_perc = 0`. For each rule specified under the include keyword, xFDS compares the values shared between the `include` rule and the parameters (lines 10 and 11) to determine if the rule should be applied. Table 3 shows how xFDS adds the additional information specified to the scenarios without mechanical makeup air. These include rules can also overwrite values defined in the variables section.

```yaml
xfds:
  render:
    - name: simple_atrium_{{cfm}}_{{mua_perc}}
      files:
        - simple_atrium.fds
      variables:
        hrr: 1000 # kW
        area: 1.5 # m
      parameters:
        cfm: [100000, 125000, 150000]  # cfm
        mua_perc: [0, 85, 90, 95] # %
        include:
            # Open doors if makeup air percentage is 0
          - mua_perc: 0
            open_doors: true
```

Figure 11 Adding Information to Specific Scenarios

Table 3 List of Models Generated by xFDS with Extra Information Added

Model Name	Exhaust Capacity	Makeup Air Percentage	Open Doors
simple_atrium_100000_0.fds	100,000 cfm	0 %	True
simple_atrium_100000_85.fds	100,000 cfm	85 %	-
simple_atrium_100000_90.fds	100,000 cfm	90 %	-
simple_atrium_100000_95.fds	100,000 cfm	95 %	-
simple_atrium_125000_0.fds	125,000 cfm	0 %	True
simple_atrium_125000_85.fds	125,000 cfm	85 %	-
simple_atrium_125000_90.fds	125,000 cfm	90 %	-
simple_atrium_125000_95.fds	125,000 cfm	95 %	-
simple_atrium_150000_0.fds	150,000 cfm	0 %	True
simple_atrium_150000_85.fds	150,000 cfm	85 %	-
simple_atrium_150000_90.fds	150,000 cfm	90 %	-
simple_atrium_150000_95.fds	150,000 cfm	95 %	-

Excluding Scenarios

It may not be desirable to generate every possible combination of the parameters specified. For example, the ductwork in the atrium might not be able to support more than 85% of the exhaust capacity when the maximum exhaust is specified. Therefore, lines 16-21 in Figure 12 tell xFDS to exclude scenarios with 150,000 cfm of exhaust and either 90% or 95% of makeup air. As such, Table 4 shows that these scenarios are no longer generated.

```yaml
xfds:
  render:
    - name: simple_atrium_{{cfm}}_{{mua_perc}}
      files:
        - simple_atrium.fds
      variables:
        hrr: 1000 # kW
        area: 1.5 # m
      parameters:
        cfm: [100000, 125000, 150000]  # cfm
        mua_perc: [0, 85, 90, 95] # %
        include:
            # Open doors if makeup air percentage is 0
          - mua_perc: 0
            open_doors: true
        exclude:
            # Not enough duct area to supply > 90% of cfm
          - cfm: 150000
            mua_perc: 95
          - cfm: 150000
            mua_perc: 90
```

Figure 12 Excluding Specific Scenarios

Table 4 List of Models Generated by xFDS with Extra Information Added and Scenarios Excluded

Model Name	Exhaust Capacity	Makeup Air Percentage	Open Doors
simple_atrium_100000_0.fds	100,000 cfm	0 %	True
simple_atrium_100000_85.fds	100,000 cfm	85 %	-
simple_atrium_100000_90.fds	100,000 cfm	90 %	-
simple_atrium_100000_95.fds	100,000 cfm	95 %	-
simple_atrium_125000_0.fds	125,000 cfm	0 %	True
simple_atrium_125000_85.fds	125,000 cfm	85 %	-
simple_atrium_125000_90.fds	125,000 cfm	90 %	-
simple_atrium_125000_95.fds	125,000 cfm	95 %	-
simple_atrium_150000_0.fds	150,000 cfm	0 %	True
simple_atrium_150000_85.fds	150,000 cfm	85 %	-

RUNNING MODELS WITH XFDS

In addition to generating models, xFDS can help with running fire models. For models that are small enough to run on a workstation or server where xFDS is installed, the `xfds run` command can help run the model under any recent version of FDS. For larger files that need to run on a cluster, xFDS can assist with generating Portable Batch System (PBS) files.

Running FDS Files Locally

FDS users may have FDS installed on their computers. However, depending on their familiarity with the software, they may not know what version is installed on their machine or if it matches the FDS

version where they run their production models, which can lead to unexpected errors. Meanwhile, other users may need access to different versions of FDS depending on each project's needs.

Docker is a software package for containerizing applications (Docker, 2022). A container is a standard unit of software that packages up code and its dependencies to run a program in an isolated environment. There are Docker images available for versions of FDS going back to FDS version 5.5.3 (Weiße, 2022). The `xfds run` command takes an FDS input file and runs it in an isolated container.

To run a model, pass the directory containing the model as an argument to the xFDS `run` command, as shown in Figure 13. By default, xFDS uses the latest version of FDS available. However, if a specific version of FDS is desired, the version can be specified, as seen in Figure 14, which forces xFDS to use FDS version 6.7.9. The xFDS documentation has more information on options for running FDS files. (PBD Tools, 2022)

```
$ xfds run /path/to/directory/with/fds_file
```

Figure 13 Running a Model with xFDS Run

```
$ xfds run -v 6.7.9 /path/to/directory/with/fds_file
```

Figure 14 Running a Model with a Specific Version of FDS

Running FDS Files on a Cluster

A compute cluster is often required for larger models, and these compute clusters often use PBS files to submit jobs. The xFDS config file can process multiple template files for a single configuration by listing them under the `files` option for the job, as seen on Lines 4-6 in Figure 15. Both files are processed as templates and placed in the output directory for the simulation. The base name of each file (file name without the file extension) has the same name specified by the `name` parameter on Line 3.

```
1 xfds:
2   render:
3     - name: atrium_{{cfm}}_{{mua_perc}}
4       files:
5         - simple_atrium.fds
6         - simple_atrium.pbs
7       variables:
8         proc: 20
9         ...
```

Figure 15 Specifying a PBS file

The PBS file uses the same templating system described above for the FDS files. For example, line 8 in Figure 15 specifies that the model requires 20 processors to perform the calculation. The number of nodes required can be calculated within the template directly using the floor division (`//`) and modulo (`%`) operators. The floor division divides the number of cores required by the number available on a single node and rounds down to the nearest integer. At the same time, the modulo operator determines how many cores are required once other nodes are filled. These calculations are shown in Table 5, while Lines 1 and 2 of Figure 16 show how these calculations can be implemented. xFDS also makes the name parameter (Line 3 of Figure 15) available to the templates, as seen on Lines 3 and 23 of Figure 16. The output of the relevant lines can be seen in Figure 17.

Table 5 Calculation of Nodes Needed for a Model

Cores Needed	Cores Per Node	Floor Division (Cores // Cores Per Node)	Modulo (Cores % Cores Per Node)
20	12	1	8
20	16	1	4

```
1  #PBS -l nodes={{ proc // 12 }}:node1:ppn=12:1:node1:ppn={{ proc % 12 }}
2  #PBS -l nodes+={{ proc // 16 }}:node2:ppn=16:1:node2:ppn={{ proc % 16 }}
3  #PBS -N {{ name }}
4  #PBS -S /bin/bash
5  #PBS -m abe
6  #PBS -M xfds@pbd.tools
7  #PBS -j oe
8
9  # load required modules
10 module load fds/6.7.9
11
12 # change to the working directory
13 cd $PBS_O_WORKDIR
14
15 # set MPI variables
16 export I_MPI_PIN=1
17 export I_MPI_PIN_MODE=pm
18 export OMP_NUM_THREADS=1
19 export MPI_PPN=$(($PBS_NUM_PPN / $OMP_NUM_THREADS))
20 export MPI_NP=$(($PBS_NP / $OMP_NUM_THREADS))
21
22 # run fds in parallel
23 mpiexec -np $MPI_NP fds_mpi {{ name }}.fds
```

Figure 16 Sample PBS Template File

```
1  #PBS -l nodes=1:node1:ppn=12:1:node1:ppn=8
2  #PBS -l nodes+=1:node2:ppn=16:1:node2:ppn=4
3  #PBS -N atrium_100000_0
...
23 mpiexec -np $MPI_NP fds_mpi atrium_100000_0.fds
```

Figure 17 Rendered PBS File

CONCLUSION

It takes time for experienced FDS users to understand what the FDS code does, even for simple test cases like the example shown in Figure 1. For example, the mesh resolution and peak heat release rate can be determined by interpreting the FDS code, but that takes time and external tools, leading to copy-and-paste errors. The xFDS templating system makes it easy to understand these values and performs calculations for the user, which reduces errors and accelerates the fire modeling process.

REFERENCES

Docker. (2022, August 19). *Use Containers to Build, Share, and Run your Applications*. Retrieved from Docker: https://www.docker.com/resources/what-container/

McGrattan, K., Hostikka, S., Floyd, J., McDermott, R., & Vanella, M. (2022). *Fire Dynamics Simulator User Guide.* Gaithersburg, Maryland: National Institute of Standards and Technology. doi:http://dx.doi.org/10.6028/NIST.SP.1019

Pallets Project. (2022, April 28). *Template Designer Documentation*. Retrieved from Jinja version 3.1.2: https://jinja.palletsprojects.com/en/3.1.x/templates/

PBD Tools. (2022, August 21). Retrieved from xFDS Documentation: https://xfds.pbd.tools/

The YAML Project. (2021, November 15). *YAML 1.2*. Retrieved from https://yaml.org/

Wade, C., Nilsson, D., Baker, G., & Olsson, P. (2021). *Fire Engineering Practitioner Tools: Survey and Analysis of Needs.* Gaithersburg, MD: SFPE.

Weiße, R. (2022, July 25). *FDS Dockerfiles*. Retrieved from Github: https://github.com/openbcl/fds-dockerfiles