VibroSim

Release 1.0

Stephen D. Holland

Jul 27, 2020

CONTENTS:

CHAPTER

ONE

VIBROSIM SIMULATOR

VibroSim Simulator is a set of tools to organize and facilitate simulating Vibrothermography testing using VibroSim.

PLEASE NOTE THAT VIBROSIM MAY NOT HAVE BEEN ADEQUATELY VALIDATED, THE NUMBERS BUILT INTO IT ALMOST CERTAINLY DO NOT APPLY TO YOUR VIBROTHERMOGRAPHY PROCESS. ITS OUTPUT CANNOT BE TRUSTED AND IS NOT SUITABLE FOR ENGINEERING REQUIREMENTS WITHOUT APPLICATION- AND PROCESS-SPECIFIC VALIDATION.

VibroSim Simulator relies directly on the following packages:

- VibroSim_COMSOL
- angled_friction_model
- crackclosuresim2
- VibroSim_WelderModel (if ultrasonic welder-base excitation is to be simulated)
- Limatix

As such, it requires MATLAB, COMSOL, and Python. The COMSOL Structural Mechanics Module and COMSOL LiveLink for MATLAB are also necessary. The Python version must be at least 2.7 (preferably 3.6 or newer) and should include the full IPython, Matplotlib, Numpy, Scipy stack as well as Pandas v0.17.1 or later. To build crackclosuresim2 you will also need the platform compiler for your Python version (see the crackclosuresim2 documentation for more information).

The Git version control system and the GitPython bindings are strongly recommended as the best way to manage or contribute updates.

While the current implementation uses COMSOL for vibration calculation and for heat flow evaluation, because of the modular nature of VibroSim it would be reasonably straightforward to re-implement those steps using other tools. The crack_heatflow package will perform COMSOL-free heat flow evaluation.

1.1 Installation

Make sure you have the commercial prerequisites (MATLAB; COMSOL with Structural Mechanics Module and LiveLink for MATLAB) installed and a scientific Python distribution (usually Anaconda https://www.anaconda.com) installed. See also the Windows Installation notes, below, if applicable.

Like most the other VibroSim components, VibroSim_Simulator is a Python package. Use the usual Python process

```
python setup.py build
```

```
python setup.py install
```

commands to install it. (If running Anaconda on Windows all these commands are from an Anaconda prompt)

After installing VibroSim_Simulator using the above process, repeat the same process on the `angled_friction_model`, `limatix`, and `VibroSim_WelderModel` packages.

Prior to installing the final package (`crackclosuresim2`) you need to make sure you have your "platform compiler" installed. Typically this is GCC on Linux (installed via your OS), XCode on Macintosh, and Visual C++ on Windows.

The exact compiler versions on Windows are listed at https://wiki.python.org/moin/WindowsCompilers The correct Windows compiler for Python 3.5-3.8 is Visual C++ 14.X and can be freely downloaded as "Build Tools for Visual Studio 2019" from https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2019 When running the installer, be sure to install "C++ build tools" and ensure the latest versions of "MSVCv142 - VS 2019 C++ x64/x86 build tools" and "Windows 10 SDK" are checked.

Once you have your platform compiler installed you can perform the usual `setup.py` steps on the crackclosuresim2 package.

The final installation step is to make the MATLAB scripts in the VibroSim_COMSOL package accessible. If a VibroSim_COMSOL .mltbx was included in your archive you can install it using MATLAB's package manager. Otherwise you can set the `MATLABPATH` environment variable to point at the `VibroSim_COMSOL/m_files` subdirectory of the VibroSim_COMSOL archive.

Once all the installation steps are complete, you can test VibroSim by running one of the examples. For example, in a terminal or Anaconda prompt from the examples/ folder of VibroSim_Simulator you could run the vibrosim_demo3 example by typing: `processtrak vibrosim_demo3.prx -a`

See the documentation below for more information on processtrak.

1.2 Windows Installation

Additional steps must be followed for this package to work on Windows. If COMSOL is to be used, the command line executables for COMSOL need to be added to the path. For COMSOL 5.4 these executables were installed to the following directory:

```
C:\Program Files\COMSOL\COMSOL54\Multiphysics\bin\win64
```

This directory needs to be added to the end of the `path` environment variable. Searching "environment variables" in the start menu is a good way to find where to make this edit. Path entries on Windows are usually separated with semicolons (;) but Windows 10 now has a convenient editor you can use to add additional entries without having to worry about separators.

1.3 VibroSim Simulator Workflow

The VibroSim simulator workflow splits the process of performing a VibroSim simulation into a sequence of steps. Each step can be run manually, but as the manual steps can be rather complicated we recommend the use of the ProcessTrak tool from the Limatix package to automate the execution of the steps.

The conceptual steps involved in a VibroSim simulation are:

1. Creation of a geometric model
2. Vibrational analysis
3. Modeling of the vibrothermography excitation
4. Prediction of heating power
5. Modeling of the heat from the crack conducting through the material

1.4 ProcessTrak

ProcessTrak is a commandline tool from the Limatix package that is used to keep track of what has been performed in a multistep process. It is executed by typing `processtrak` at the command line. ProcessTrak is configured by an XML listing of input file steps in a `.prx` file. Usually ProcessTrak is run referencing that `.prx` file followed by additional instructions for what is desired. For example

```
processtrak vibrosim_demo3.prx --status
```

will list out the status of each step in the process for each input file.

ProcessTrak is designed to process input data into output results. The input data is specified in the form of an XML "experiment log" (`.xlg` file). The experiment log specifies or references the various inputs. The first ProcessTrak step is always an implicit `copyinput` step which copies the input `.xlg` to an output "processed experiment log" (`.xlp`) file. The processed experiment log is annotated with Provenance information, log output from the various processing steps, and the result data from each processing step. For example,

```
processtrak vibrosim_demo3.prx -s copyinput
```

will run the implicit `copyinput` step on the input files listed in `vibrosim_demo3.prx`, generating an output `.xlp` file (the input `.xlg` is never touched).

1.5 Git and Limatix-Git

Having confidence in simulation output requires confidence that you executed the code you intended and confidence that you have a repeatable process. We recommend the use of Git and Limatix-Git to perform version management both on the scripts and parameters of the simulation and on the generated output from the simulation. This process will require having Git and GitPython installed. Limatix-Git is included in the Limatix installation.

To start using the Limatix-Git process, entering

```
git init
```

in your simulation directory will create a new Git repository there.

We recommend managing your simulation process with two branches: "master" which contains the scripts and instructions but no output, and "master_processed" which also includes processed output. (These two branches can of course themselves be branched as desired).

The `limatix-git` program exists to help automate the process of committing changed scripts and simulation output to the proper branches. It is based on the assumption that the name of any branch intended to contain processed output ends with `_processed`. It operates on the principle that scripts, input data, etc. should be committed to the master branch, and processed output should be cross-referenced in the `.xlp` files.

To add files to the unprocessed branch, check out that branch, run `limatix-git add -a` to stage files for commit, `git status` to verify only input files have been staged, and `git commit` to perform the commit.

To add files to the processed branch, check out that branch, run `limatix-git add-processed -a` to stage files for commit, `git status` to verify only processed output has been staged, and `git commit` to perform the commit.

1.6 COMSOL-based VibroSim Workflow

The COMSOL-based VibroSim workflow follows roughly the conceptual steps listed above, but the model creation is nominally all done up-front (in reality the first few steps will be iterated to get the model where it needs to be).

The steps involved in a COMSOL-based VibroSim simulation are:

1. Scripting COMSOL to create a geometric and physics model, including mounting, excitation position/couplant, vibration monitoring, and a healed internal boundary representing the crack,
2. Vibration analysis of sample including:

a. Modal analysis

b. Spectrum verification

c. Frequency response calculation

d. Generation of time-domain response.

3. Modeling of the vibrothermography excitation to evaluate response at the crack
4. Prediction of heating power from response at the crack.
5. Modeling of the heat from the crack conducting through the material to the surface.

1.7 Troubleshooting

ProcessTrak error: FileNotFoundError in procstepmatlab_execfunc:

- The comsol binaries are not in the system path. Please add them to the command path.

Warning from MATLAB about dataguzzler-lib/matlab or dc_unitsparam:

- These are expected and nothing to worry about

Error from Matlab: Undefined function or variable 'InitializeVibroSimScript'.

- This means VibroSim_COMSOL is not accessible from MATLAB. One way to make it accessible is to install the VibroSim_COMSOL.mltbx as a MATLAB add-on. Another way is to set the MATLABPATH environment variable to the VibroSim_COMSOL m_files subfolder.

Error from processtrak: pkg_resources.DistributionNotFound

- This usually means that the processtrak script was installed by a different version of Python than the version of Python that is executing. Reinstall Limatix using your desired Python version.

1.8 Building the VibroSim_Simulator Documentation

A rendered form of the VibroSim documentation is usually included in distributed VibroSim release archives. It can also be built using Sphinx. Documentation source code can be found in the `docs` folder. Instructions for how to install Sphinx can be found at their website. Once Sphinx is installed an html version of the documentation can be built using the makefile in the `docs` folder:

```
make html
```

Sphinx can also be used to create .pdf documentation using Latex:

```
make latexpdf
```

CHAPTER

TWO

TUTORIAL

Lets look in depth at `vibrosim_demo3` in the examples folder. This example has three files associated with it, `vibrosim_demo3.prx` (processing steps), `vibrosim_demo3.xlg` (parameter storage), and `vibrosim_demo3_comsol.m` (matlab file for creating the COMSOL model).

The `.prx` file contains the processing steps to be performed with the model. It is managed by the software tool called `processtrak`, a part of `Limatix`.

Use the tool in the following way:

`processtrak <args> vibrosim_demo3.prx`

The following are possible arguments.

1. `-s <steps>` : Run only listed steps (multiple OK)
2. `-a` : Run all steps
3. `-i` : Use ipython interactive mode to execute script
4. **`--needed`** [Filter down steps and input files according to what "needs"] to be run – i.e. missing or out-of-order steps, etc. DOES NOT PERFORM PROVENANCE VERIFICATION

Running this command:

`processtrak --status vibrosim_demo3.prx`

produces the following output:

```
Input file: vibrosim_demo3.xlg
---------------------------
           copyinput NOT_EXECUTED NEEDED
         dummyoutput NOT_EXECUTED NEEDED
          buildmodel NOT_EXECUTED NEEDED
            runmodal NOT_EXECUTED NEEDED
  synthetic_spectrum NOT_EXECUTED NEEDED
     entersweepfreqs NOT_EXECUTED NEEDED
 setsweepfrequencies NOT_EXECUTED NEEDED
      sweep_analysis NOT_EXECUTED NEEDED
      enterburstfreq NOT_EXECUTED NEEDED
   setburstfrequency NOT_EXECUTED NEEDED
      burst_analysis NOT_EXECUTED NEEDED
eval_closure_state_from_tip_positions NOT_EXECUTED NEEDED
calc_heating_singlefrequency NOT_EXECUTED NEEDED
   heatflow_analysis NOT_EXECUTED NEEDED
```

None of the steps have been run yet, so they all have the same `NOT_EXECUTED` and `NEEDED` tags.

Now we can run the first step with the following command:

`processtrak vibrosim_demo3.prx -s copyinput`:

```
Processing step copyinput on vibrosim_demo3.xlg->vibrosim_demo3.xlp
```

`processtrak --status vibrosim_demo3.prx`:

```
Input file: vibrosim_demo3.xlg
---------------------------
           copyinput     2020-06-18T13:48:49.707642-05:00
         dummyoutput NOT_EXECUTED NEEDED
          buildmodel NOT_EXECUTED NEEDED
            runmodal NOT_EXECUTED NEEDED
  synthetic_spectrum NOT_EXECUTED NEEDED
     entersweepfreqs NOT_EXECUTED NEEDED
 setsweepfrequencies NOT_EXECUTED NEEDED
      sweep_analysis NOT_EXECUTED NEEDED
      enterburstfreq NOT_EXECUTED NEEDED
   setburstfrequency NOT_EXECUTED NEEDED
      burst_analysis NOT_EXECUTED NEEDED
eval_closure_state_from_tip_positions NOT_EXECUTED NEEDED
calc_heating_singlefrequency NOT_EXECUTED NEEDED
   heatflow_analysis NOT_EXECUTED NEEDED
```

Now that the `copyinput` step has been executed, it is no longer `needed` and the timestamp of the step has been recorded. This data is kept in the `.xlp` file.

The `copyinput` step is always present and represents a near verbatim copy of the unprocessed `.xlg` experiment log to a "processed" `.xlp` experiment log. The only initial difference is the addition of provenance (step execution) information to the `.xlp`. Subsequent steps will operate on the `.xlp` adding processed results and hyperlinks to generated output files. If you make minor changes to the `.xlg` and want to merge those changes into the `.xlp` without wiping all of the generated output you can run the `mergeinput` step instead of `copyinput`.

The steps (except for `copyinput` / `mergeinput`) are specified in the `.prx` file. When you run a step, it uses parameters explicitly specified in the `.prx` step definition as well as implicitly identifying the remaining paramters from the `.xlp` experiment log. The step then returns values which are added to the experiment log. The parameters and returns are documented in the "ProcessTrak Steps" chapter below.

The next step is the `dummyoutput` step. The heat flow portion of the COMSOL model requires a heat distribution input in order to successfully build the model. The `dummyoutput` step creates a heat distribution that is zero to satisfy COMSOL. You can run this step with:

```
processtrak vibrosim_demo3.prx -s dummyoutput
```

The `dummyoutput` step in the `.prx` file is specifed as:

```
<prx:script name="vibrocomsol_createdummyoutput.py"/>
```

The `name=` attribute means to search for this script in the processtrak search path (if you wanted to provide an explicit path you would use `xlink:href=` instead). This particular script is installed by the `VibroSim_Simulator` package and its source is in the `VibroSim_Simulator/pt_steps` subdirectory. As documented below, it takes two parameters from the experiment log: `dc:dest` – the results output location – and `dc:measident` the identifier of the particular simulation run. It then returns (adds to the processed experiment log) `dc:dummy_heatingdata`, which is a hyperlink (following the XLink standard) to the generated dummy heating input file for COMSOL. If you view the `.xlp` file in a text or XML editor after running this step you should be able to find the new `dc:dummy_heatingdata` element. The `dc:dummy_heatingdata` element will be used as a parameter by the `buildmodel` step indicating where the initial heating data is stored.

Steps can be written in Python `.py`, MATLAB `.m`, or MATLAB/COMSOL `_comsol.m`. The next step is `buildmodel`, which is a custom MATLAB/COMSOL script (referenced explicitly by `xlink:href=` instead of being found in the search path). The _comsol portion of the filename is important because it tells ProcessTrak to run the MATLAB script in the COMSOL environment.

NOTE: When running a COMSOL step for the first time it is common for it to ask for a username and password. These are just for communication between the COMSOL server and client on your own computer and they are remembered automatically so they do not really matter, but they might be worth writing down. Do not re-use an important password for the COMSOL server.

Parameters expected by a MATLAB or MATLAB/COMSOL step are listed in the commented first line of the file, similar to how they would be defined for a MATLAB function. For example:

```
% function ret = vibrosim_demo3_comsol(dc_dest_href,dc_measident_str,dc_dummy_
↪heatingdata_href,dc_amplitude_float,dc_staticload_mount_float,dc_spcmaterial_str,dc_
↪YoungsModulus_float, dc_YieldStrength_float, dc_PoissonsRatio_float, dc_Density_
↪float,dc_spcThermalConductivity_float, dc_spcSpecificHeatCapacity_float,dc_
↪spcrayleighdamping_alpha_float,dc_spcrayleighdamping_beta_float, dc_exc_t0_float,␣
↪dc_exc_t4_float, dc_simulationcameranetd_float,dc_cracksemimajoraxislen_float,dc_
↪cracksemiminoraxislen_float,dc_crack_type_side1_str,dc_crack_type_side2_str)
```

The third parameter dc_dummy_heatingdata_href instructs ProcessTrak to find a entry dc:dummy_heatingdata in the processed experiment log (.xlp), to interpret it as a hypertext reference (xlink:href) and store the value in the MATLAB variable dc_dummy_heatingdata_href. Parameters to Python steps work similarly and are defined by the parameters to the run() function within the step.

Steps can be run in interactive mode with the -i option to processtrak. processtrak vibrosim_demo3.prx -s buildmodel -i

This will cause the step to execute up to any errors or completion and leave an interactive environment. You can then evaluate variables, copy/paste code, etc. In MATLAB you can rerun the script just by typing its name. For COMSOL/MATLAB steps you can also externally run comsol mphclient and use the "Connect to Server" option to interact graphically with the COMSOL model. When you are done, type eval(retcommand) (MATLAB) or press Ctrl-D (Python) to store the step output and move on.

The generated output from a COMSOL/MATLAB step will usually be saved in the _output subdirectory. You can load generated .mph files directly into the COMSOL GUI. In some cases temporary output .mph files are left under the system temporary directory (usually /tmp or c:\temp) with only reprocessed output stored in the _output subdirectory.

Steps can be run out of order, as long as the .xlp has everything that is needed for the step. If needed inputs are not present, the step will fail. Obviously running steps out of order can cause inconsistencies in the final results if you are not careful.

processtrak vibrosim_demo3.prx -s entersweepfreqs

processtrak --status vibrosim_demo3.prx:

```
Input file: vibrosim_demo3.xlg
---------------------------
           copyinput     2020-06-18T13:48:49.707642-05:00
         dummyoutput NOT_EXECUTED NEEDED
          buildmodel NOT_EXECUTED NEEDED
            runmodal NOT_EXECUTED NEEDED
  synthetic_spectrum NOT_EXECUTED NEEDED
 entersweepfreqs      2020-06-18T13:53:03.368990-05:00  FAILURE NEEDED
 setsweepfrequencies NOT_EXECUTED NEEDED
      sweep_analysis NOT_EXECUTED NEEDED
      enterburstfreq NOT_EXECUTED NEEDED
   setburstfrequency NOT_EXECUTED NEEDED
      burst_analysis NOT_EXECUTED NEEDED
```

(continues on next page)

(continued from previous page)

```
eval_closure_state_from_tip_positions NOT_EXECUTED NEEDED
calc_heating_singlefrequency NOT_EXECUTED NEEDED
   heatflow_analysis NOT_EXECUTED NEEDED
```

All steps will be run from scratch if the following command is run:

```
processtrak vibrosim_demo3.prx -a
```

All steps with the `NEEDED` flag will be run with the following command:

```
processtrak vibrosim_demo3.prx -a --needed
```

Once all of the steps have been run, you can see the final output (hyperlinked in vibrosim_demo3.xlp) by loading the heatflow COMSOL model `vibrosim_demo3_output/meas1_heating.mph`, opening the `Results` tree and looking at the `vibro_heating_plot`. You can drag the model around, select different times to view the heating, etc. There is also a snapshot of this plot that should be saved in `vibrosim_demo3_output/meas1_heating.png`.

In general VibroSim saves output after each step. In many cases these are plots, .csv's, etc. All output is written in standard or text based formats where possible (sometimes compressed by default e.g. with bzip2) In addition, temporary COMSOL models are sometimes written to the system temporary directory (profile/AppData/Local/Temp on Windows) – these can occasionally be useful when troubleshooting. See the reference manual for more detailed information on individual step outputs.

CHAPTER

THREE

SUMMARY OF VIBROSIM_DEMO3 FILES

3.1 vibrosim_demo3.prx

Lists the set of steps to be run and lists the experiment logs (.xlg) on which those steps should be run.

3.2 vibrosim_demo3.xlg

A .xlg contains the unprocessed experiment log. processtrak is primarily a tool for processing data collected in an experiment, after all. It contains all the parameters necessary to start the simulation. The first step in a simulation is to copy the .xlg into a processed experiment log .xlp file. This new .xlp file contains all the parameters in the .xlg, all parameters and results of processtrak steps, and tracking information about when each step was run and if it completed properly.

.xlg files are xml based, meaning they are hierarchical in nature. It is a single experiment tag with multiple measurement tags. Parameters that are consistent for a number of simulations can be stored under the experiment tag, thus making them global. These parameters can be overwritten in the measurement tags, allowing the user to run multiple simulations with slightly varying input parameters. For example:

```
<dc:experiment xmlns:dc="http://limatix.org/datacollect" xmlns:xlink="http://www.w3.
↪org/1999/xlink" xmlns:dcv="http://limatix.org/dcvalue" xmlns:prx="http://limatix.
↪org/processtrak/processinginstructions">
    <dc:measident>meas1</dc:measident> <!-- measident is used as a filename prefix␣
↪for the various output files generated -->
    <dc:measurement>
        <dc:measident>meas1_direct_singlefreq</dc:measident>
        <dc:heatcalctype>singlefrequency</dc:heatcalctype>
        ...
    </dc:measurement>
    <dc:measurement>
        <dc:measident>meas1_via_weldercalc</dc:measident>
        <dc:heatcalctype>welder</dc:heatcalctype>
        ...
    </dc:measurement>
</dc:experiment>
```

3.3 `vibrosim_demo3_comsol.m`

This file contains all instructions necessary to build the COMSOL model for use in VibroSim_Simulator. There are examples of this in the examples folder. In depth information about how these files work can be found in the documentation of the sister software package `VibroSim_COMSOL`.

CHAPTER

FOUR

LISTING OF EXAMPLES

4.1 vibrosim_demo3.prx

This basic demonstration uses as an example a simple surface crack in a bar-shaped test specimen. It is programmed for broadband (sweep and tuned toneburst) excitation.

4.2 vibrosim_demo3_crosscheck.prx

This example, otherwise similar to `vibrosim_demo3.prx`, demonstrates cross checking the `calc_heating_singlefrequency` step against the combined `calc_singlefrequency_motion` and `calc_heating_welder` steps that together have the same effect. To perform the cross-checking, it demonstrates the use of sub-measurements and multiple `<prx:elementmatch>` criteria to operate different steps on different sub-measurements.

4.3 vibrosim_demo4.prx

This basic demonstration illustrates testing a simple edge crack in a bar. It is programmed for broadband (sweep and toneburst) excitation.

4.4 cantilever_example.prx

This basic demonstration tests a cantilever with one fixed end and a surface crack, excited with an ultrasonic welder.

4.5 cantilever_example_viscousdamping.prx

This example extends `cantilever_example.prx` with a more physically meaningful but more complicated damping model, including prediction of radiative vibration losses into the cantilever mount.

4.6 complex_model_demo.prx

This example demonstrates simulating a cracked gear. It demonstrates the use of the COMSOL CAD import module to load in a CAD cross-section profile to COMSOL that is revolved, mirrored, etc. and then meshed. The example uses ultrasonic welder excitation.

CHAPTER

FIVE

PROCESSTRAK STEPS

The following are ProcessTrak steps that come with various components of VibroSim. All parameters listed in these steps must be explicitly passed from the `.prx` file or found in the processed experiment log (`.xlp`) file. Parameters in the `.xlp` file can either come from the unprocessed experiment log (`.xlg`), or have been placed in the `.xlp` file as a return from one of the prior steps.

`vibrosim_synthetic_spectrum``()`

This ProcessTrak step will take the output of the modal analysis and calculate a spectrum. Each mode in the modal analysis is a peak in the spectrum, with the amplitude and bandwidth determined by the complex valued eigenfrequency.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters **`dc:modalfreqs`** – Frequencies of the eigenmodes.

Return dc:modal_synthetic_spectrum Synthetic spectrum figure.

`vibrocomsol_runmodal_comsol``()`

This ProcessTrak step will open COMSOL, run the modal analysis study, and save the results.

The purpose of the modal analysis is to identify candidate resonant modes of the specimen. In vibrothermography testing you want to excite enough resonant modes that you will be pretty much guaranteed that any point in the specimen will have large strains in at least one of the modes. The modal analysis can be used to find these resonances. In experiments, this would be performed using a broadband frequency sweep across the entire range of possible excitation frequencies.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **`dc:dest`** – Results output folder.
- **`dc:measident`** – Measurement identifier.
- **`dc:model_comsol`** – Save file for the COMSOL model.

Return dc:modalcalc_comsol Save file for the COMSOL model with modal results.

Return dc:modalfreqs Frequencies of the eigenmodes.

`vibrocomsol_createdummyoutput``()`

This ProcessTrak step will create a dummy heating data file. This needs to exist to create the COMSOL model. The ProcessTrak step `vibrocomsol_calc_heating_welder` will populate this file with crack heating data. The ProcessTrak step *`vibrocomsol_heatflow_analysis_comsol`* will use the data to analyze heat flows.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:dest** – Results output folder.
- **dc:measident** – Measurement identifier.

Return dc:dummy_heatingdata Output file for heating data. Heating data is the heat power of the crack as a function of time and position on crack.

vibrocomsol_entersweepfreqs()

This ProcessTrak step will show the results of the modal analysis and ask the user to specify the range of frequencies to be run by a later ProcessTrak step (*vibrocomsol_sweep_analysis_comsol*). If the returns of this ProcessTrak step are already defined in the experiment log (.xlp) then those predefined values are given as defaults for the user to accept. This would occur if this step has been run previously or if they are defined in the initial experiment log (.xlg).

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:modalfreqs** – Frequencies of the eigenmodes.
- **dc:modalcalc_comsol** – Save file for the COMSOL model with modal results.

Return dc:sweep_start_frequency Starting frequency for a sweep analysis.

Return dc:sweep_step_frequency Frequency step for a sweep analysis.

Return dc:sweep_end_frequency Ending frequency for a sweep analysis.

vibrocomsol_setsweepfrequencies_comsol()

This ProcessTrak step will open COMSOL and set the parameters needed for the sweep analysis.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:dest** – Results output folder.
- **dc:measident** – Measurement identifier.
- **dc:model_comsol** – Save file for the COMSOL model.
- **dc:sweep_start_frequency** – Starting frequency for a sweep analysis.
- **dc:sweep_step_frequency** – Frequency step for a sweep analysis.
- **dc:sweep_end_frequency** – Ending frequency for a sweep analysis.

Return dc:model_comsol_withsweepfrequencies Save file for the COMSOL model with sweep study results.

vibrocomsol_sweep_analysis_comsol()

This ProcessTrak step will open COMSOL, run the sweep analysis study, and save the results.

The purpose of the frequency sweep is to do a more precise analysis than the modal analysis of specific candidate excitations over a range of frequencies. This is analogous to the narrowband sweeps that are performed in the vibrothermography process to identify the exact optimal excitation frequency for a particular resonant mode.

Identify the dominant frequency from the crack strain magnitude and vibrometer plots, and make note of this frequency for use in the tone-burst analysis.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:dest** – Results output folder.
- **dc:measident** – Measurement identifier.

- **dc:model_comsol_withsweepfrequencies** – Save file for the COMSOL model with sweep study results.

Return dc:sweep_spectrum Sweep spectrum image.

vibrocomsol_enterburstfreq()

This ProcessTrak step will show the results of the modal analysis and the sweep analysis, and then ask the user to specify the single frequency to be run by a later ProcessTrak step (*vibrocomsol_burst_analysis_comsol*). If the returns of this ProcessTrak step are already defined in the experiment log (.xlp) then those values are given as defaults for the user to accept. This would occur if this step has been run previously or if they are defined in the initial experiment log (.xlg).

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:modalfreqs** – Frequencies of the eigenmodes.
- **dc:modalcalc_comsol** – Save file for the COMSOL model with modal results.
- **dc:sweep_spectrum** – Sweep spectrum image.

Return dc:excitation_frequency The frequency used to excite the specimen.

vibrocomsol_setburstfrequency_comsol()

This ProcessTrak step will open COMSOL and set the burst frequency to be used in the *vibrocomsol_burst_analysis_comsol* step.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:dest** – Results output folder.
- **dc:measident** – Measurement identifier.
- **dc:model_comsol** – Save file for the COMSOL model.
- **dc:excitation_frequency** – The frequency used to excite the specimen.

Return dc:model_comsol_withburstfrequency Save file for the COMSOL model with burst study parameters.

vibrocomsol_burst_analysis_comsol()

This ProcessTrak step will open COMSOL, run the burst analysis study, and save the results. It will calculate the stress at the crack face, a critical component of the heating calculation. The burst frequency must be chosen from the results of the sweep analysis and injected into the model using the *vibrocomsol_enterburstfreq* and *vibrocomsol_setburstfrequency_comsol* processtrak steps.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:dest** – Results output folder.
- **dc:measident** – Measurement identifier.
- **dc:model_comsol_withburstfrequency** – Save file for the COMSOL model with burst study parameters.

Return dc:burstcalc_comsol Save file for the COMSOL model with burst study results.

Return dc:harmonicburst_normalstress Stress in the model at the crack center, normal to the crack face. (Mode I)

Return dc:harmonicburst_shearstressmajor Shear stress in the model at the crack center, in the crack semi-major direction. (Mode II)

Return dc:harmonicburst_shearstressminor Shear stress in the model at the crack center, in the crack semi-minor direction. (Mode III)

`vibrosim_calc_heating_singlefrequency()`

This ProcessTrak step will calculate the heat generated by the crack when the sample is subjected to the burst excitation.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **`dc:dest`** – Results output folder.
- **`dc:measident`** – Measurement identifier.
- **`dc:friction_coefficient`** – Friction coefficient of the crack surface.
- **`dc:msqrtR`** – Crack asperity density.
- **`dc:staticload`** – Static bending opening load on crack.
- **`dc:tortuosity`** – Crack tortuosity, standard deviation of the crack trajectory.
- **`dc:numdraws`** – Crack tortuosity is a statistical distribution, this parameter defines how many draws to take at each position along the crack for calculating the crack heating.
- **`dc:YoungsModulus`** – Youngs modulus of the material.
- **`dc:PoissonsRatio`** – Poissons Ratio of the material.
- **`dc:YieldStrength`** – Yield strength of the material.
- **`dc:Density`** – Density of the material.
- **`dc:crack_model_normal`** – Which crack closure model to use for normal loading. `ModeI_throughcrack_CODformula` or `Tada_ModeI_CircularCrack_along_midline`.
- **`dc:crack_model_shear`** – Which stick/slip model to use for shear loading. `Fabrikant_ModeII_CircularCrack_along_midline`, `ModeII_throughcrack_CSDformula`, or `ModeIII_throughcrack_CSDformula`.
- **`dc:crack_model_shear_factor`** – Sensitivity factor for shear vs normal heating.
- **`dc:exc_t0`** – Start of excitation envelope ramp-up.
- **`dc:exc_t1`** – End of excitation envelope ramp-up.
- **`dc:exc_t2`** – Start of excitation envelope ramp-down.
- **`dc:exc_t3`** – End of excitation envelope ramp down.
- **`dc:exc_t4`** – End of excitation vibration calculation.
- **`dc:excitation_frequency`** – The frequency used to excite the specimen.
- **`dc:harmonicburst_normalstress`** – Stress in the model at the crack center, normal to the crack face. (Mode I)
- **`dc:harmonicburst_shearstressmajor`** – Shear stress in the model at the crack center, in the crack semi-major direction. (Mode II)
- **`dc:harmonicburst_shearstressminor`** – Shear stress in the model at the crack center, in the crack semi-minor direction. (Mode III)

- **dc:crack_type_side1** – Crack type of side1, can be `halfthrough` or `quarterpenny`.
- **dc:crack_type_side2** – Crack type of side2, can be `halfthrough` or `quarterpenny`.
- **dc:crack_shearstress_axis** – `major` (mode II) or `minor` (mode III) axis, defines the axis used to calculate shear heating.
- **dc:thickness** – Thickness of the material at the crack, used only for `halfthrough` cracks.
- **dc:closurestate_side1** – Closure state, side 1.
- **dc:closurestate_side2** – Closure state, side 2.
- **dc:a_side1** – Semimajor axis length of side 1.
- **dc:a_side2** – Semimajor axis length of side 2.

Return dc:heatpower Heat power vs crack location figure.

Return dc:heatingdata A .tsv file where the four columns are time (s), radius (m), side 1 heating (W/m^2), and side 2 heating (W/m^2).

Return dc:heatingtotalpower Total heating power of the crack.

vibrocomsol_heatflow_analysis_comsol()

This Processtrak step will take the crack heating power data and project it along the crack in the COMSOL model. A heatflow study is performed to analyze the flow of heat in the specimen during the excitation time. The `vibro_heating_image` return is a snapshot of the heating data at the very end of the excitation.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:dest** – Results output folder.
- **dc:measident** – Measurement identifier.
- **dc:model_comsol** – Save file for the COMSOL model.
- **dc:heatingdata** – A .tsv file where the four columns are time (s), radius (m), side 1 heating (W/m^2), and side 2 heating (W/m^2).
- **dc:exc_t3** – End of excitation envelope ramp down.

Return dc:vibro_heating_image Snapshot of the heating specimen.

Return dc:heatflow_comsol Save file for the COMSOL model with heatflow study results.

vibrosim_calc_heating_welder()

This processtrak step will call the angled friction model to determine the heating in each motion cycle as a function of position within the crack.

This ProcessTrak step is included in the `VibroSim_Simulator` software package.

Parameters

- **dc:dest** – Results output folder.
- **dc:measident** – Measurement identifier.
- **dc:friction_coefficient** – Friction coefficient of the crack surface.
- **dc:msqrtR** – Crack asperity density.
- **dc:staticload** – Static bending opening load on crack.

- **dc:tortuosity** – Crack tortuosity, standard deviation of the crack trajectory.
- **dc:numdraws** – Crack tortuosity is a statistical distribution, this parameter defines how many draws to take at each position along the crack for calculating the crack heating.
- **dc:YoungsModulus** – Youngs modulus of the material.
- **dc:PoissonsRatio** – Poissons Ratio of the material.
- **dc:YieldStrength** – Yield strength of the material.
- **dc:Density** – Density of the material.
- **dc:crack_model_normal** – Which crack closure model to use for normal loading. `ModeI_throughcrack_CODformula` or `Tada_ModeI_CircularCrack_along_midline`.
- **dc:crack_model_shear** – Which stick/slip model to use for shear loading. `Fabrikant_ModeII_CircularCrack_along_midline`, `ModeII_throughcrack_CSDformula`, or `ModeIII_throughcrack_CSDformula`.
- **dc:crack_model_shear_factor** – Sensitivity factor for shear vs normal heating.
- **dc:exc_t0** – Start of excitation envelope ramp-up.
- **dc:exc_t4** – End of excitation vibration calculation.
- **dc:motion** – Table of motion of the tip position, contact force, crack stress, laser sense point, etc., resulting from the welder tip and specimen interaction. Multicolumn csv.
- **dc:crack_type_side1** – Crack type of side1, can be `halfthrough` or `quarterpenny`.
- **dc:crack_type_side2** – Crack type of side2, can be `halfthrough` or `quarterpenny`.
- **dc:crack_shearstress_axis** – `major` (mode II) or `minor` (mode III) axis, defines the axis used to calculate shear heating.
- **dc:thickness** – Thickness of the material at the crack, used only for `halfthrough` cracks.
- **dc:closurestate_side1** – Closure state, side 1.
- **dc:closurestate_side2** – Closure state, side 2.
- **dc:a_side1** – Semimajor axis length of side 1.
- **dc:a_side2** – Semimajor axis length of side 2.

Return dc:heatingdata A .tsv file where the four columns are time (s), radius (m), side 1 heating (W/m^2), and side 2 heating (W/m^2).

Return dc:heatingtotalpower Total heating power of the crack.

vibrosim_eval_closure_state_from_tip_positions()

This ProcessTrak step is used in vibrosim to evaluate crack closure state from crack tip positions given in an XML element. It is provided by the `crackclosuresim2` package.

The crack closure state is given as four arrays interpreted as text within XML elements of the experiment log, e.g:

```
<dc:reff_side1 dcv:units="m" dcv:arraystorageorder="C">
  <dcv:arrayshape>9</dcv:arrayshape>
  <dcv:arraydata>
```

(continues on next page)

(continued from previous page)

```
    .5e-3 .7e-3 .9e-3 1.05e-3 1.2e-3 1.33e-3 1.45e-3 1.56e-3 1.66e-3
  </dcv:arraydata>
</dc:reff_side1>
<dc:seff_side1 dcv:units="Pa" dcv:arraystorageorder="C">
  <dcv:arrayshape>9</dcv:arrayshape>
  <dcv:arraydata>
    0.0 50e6 100e6 150e6 200e6 250e6 300e6 350e6 400e6
  </dcv:arraydata>
</dc:seff_side1>

<dc:reff_side2 dcv:units="m" dcv:arraystorageorder="C">
  <dcv:arrayshape>9</dcv:arrayshape>
  <dcv:arraydata>
    .5e-3 .7e-3 .9e-3 1.05e-3 1.2e-3 1.33e-3 1.45e-3 1.56e-3 1.66e-3
  </dcv:arraydata>
</dc:reff_side2>
<dc:seff_side2 dcv:units="Pa" dcv:arraystorageorder="C">
  <dcv:arrayshape>9</dcv:arrayshape>
  <dcv:arraydata>
    0.0 50e6 100e6 150e6 200e6 250e6 300e6 350e6 400e6
  </dcv:arraydata>
</dc:seff_side2>
```

The `reff` (effective tip radius) values are given in meters and the `seff` (corresponding normal stress) values are given in Pascals. The radius values should be listed in increasing order. The last radius value on each side (side1 - left or side2 - right) should correspond to the length of that side of the crack.

Parameters

- **`dc:dest`** – Results output folder.
- **`dc:measident`** – Measurement identifier.
- **`dc:YoungsModulus`** – Youngs modulus of the material.
- **`dc:PoissonsRatio`** – Poissons Ratio of the material.
- **`dc:YieldStrength`** – Yield strength of the material.
- **`dc:reff_side1`** – Effective tip radius array of crack side 1.
- **`dc:seff_side1`** – Normal stress corresponding to tip radius array, side 1.
- **`dc:reff_side2`** – Effective tip radius of crack side 2.
- **`dc:seff_side2`** – Normal stress corresponding to tip radius array, side 2.
- **`dc:crack_model_normal`** – Which crack closure model to use for normal loading. `ModeI_throughcrack_CODformula` or `Tada_ModeI_CircularCrack_along_midline`.
- **`dc:crack_model_shear`** – Which stick/slip model to use for shear loading. `Fabrikant_ModeII_CircularCrack_along_midline`, `ModeII_throughcrack_CSDformula`, or `ModeIII_throughcrack_CSDformula`.

Return dc:closureplot_side1 Plot of the closure state, side 1.

Return dc:closureplot_side2 Plot of the closure state, side 2.

Return dc:closurestate_side1 Closure state, side 1.

Return dc:closurestate_side2 Closure state, side 2.

Return dc:a_side1 Semimajor axis length of side 1.

Return dc:a_side2 Semimajor axis length of side 2.

`vibrosim_plot_welder_motion``()`

This ProcessTrak step will plot the welder motion.

Provided by the `VibroSim_WelderModel` package.

Parameters

- **`dc:dest`** – Results output folder.
- **`dc:measident`** – Measurement identifier.
- **`dc:motion`** – Table of motion of the tip position, contact force, crack stress, laser sense point, etc., resulting from the welder tip and specimen interaction. Multicolumn csv.
- **`dc:exc_t0`** – Start of excitation envelope ramp-up.

Return plots Welder motion plots.

`vibrosim_simulate_welder``()`

The processtrak step takes the desired weld profile and the pneumatic force and dynamic behavior of the welder and specimen and generates a motion table. It generates this table through a time based integration simulation of the repeated impacts of the welder.

This step can be accelerated through the use of OpenCL. To use the GPU the device information needs to be passed to the processtrak step in the `.prx` file. Look in the `cantilever_example.prx` file for an example.

Provided by the `VibroSim_WelderModel` package.

Parameters

- **`dc:dest`** – Results output folder.
- **`dc:measident`** – Measurement identifier.
- **`dc:dynamicmodel`** – Time-domain specimen stress and motion at transducer contact, laser, and crack locations
- **`dc:exc_t0`** – Start of excitation envelope ramp-up.
- **`dc:exc_t1`** – End of excitation envelope ramp-up.
- **`dc:exc_t2`** – Start of excitation envelope ramp-down.
- **`dc:exc_t3`** – End of excitation envelope ramp down.
- **`dc:exc_t4`** – End of excitation vibration calculation.
- **`dc:mass_of_welder_and_slider`** – Assumed mass of welder assembly.
- **`dc:pneumatic_force`** – Pneumatic force behind the welder.
- **`dc:welder_elec_ampl`** – Electrical excitation amplitude going into the welder. Not a calibrated value.
- **`dc:YoungsModulus`** – Youngs modulus of the material.
- **`dc:PoissonsRatio`** – Poissons Ratio of the material.
- **`dc:welder_spring_constant`** – The springiness of the welder mounts.
- **`dc:R_contact`** – Welder tip assumed Hertzian contact radius.
- **`dc:welder_elec_freq`** – Frequency of the electrical welder excitation.
- **`dc:contact_model_timestep`** – Timestep used in the contact model.

- **dc:gpu_device_priority_list** – Prioritized list of gpus to use in place of cpu.
- **dc:gpu_precision** – `single` or `double`.

Return dc:motion Table of motion of the tip position, contact force, crack stress, laser sense point, etc., resulting from the welder tip and specimen interaction. Multicolumn csv.

vibrosim_process_multisweep()

This processtrak step will process the freq_band_analysis output to create a time-domain waveform. This time domain waveform, generated from the multiple sweeps of relevant frequencies to the system, represents the impulse response of the system.

Provided by the `VibroSim_WelderModel` package.

Parameters

- **dc:dest** – Results output folder.
- **dc:measident** – Measurement identifier.
- **dc:seg1_xducercontactprobe_displ** – Transducer contact probe displacement.
- **dc:seg1_xducercontactprobe_vel** – Transducer contact probe velocity.
- **dc:seg1_laser_displ** – Displacement at laser vibrometer spot.
- **dc:seg1_laser_vel** – Velocity at laser vibrometer spot.
- **dc:seg1_crackcenterstress** – Crack center stress.
- **dc:seg2_xducercontactprobe_displ** – Transducer contact probe displacement.
- **dc:seg2_xducercontactprobe_vel** – Transducer contact probe velocity.
- **dc:seg2_laser_displ** – Displacement at laser vibrometer spot.
- **dc:seg2_laser_vel** – Velocity at laser vibrometer spot.
- **dc:seg2_crackcenterstress** – Crack center stress.
- **dc:seg3_xducercontactprobe_displ** – Transducer contact probe displacement.
- **dc:seg3_xducercontactprobe_vel** – Transducer contact probe velocity.
- **dc:seg3_laser_displ** – Displacement at laser vibrometer spot.
- **dc:seg3_laser_vel** – Velocity at laser vibrometer spot.
- **dc:seg3_crackcenterstress** – Crack center stress.
- **dc:seg4_xducercontactprobe_displ** – Transducer contact probe displacement.
- **dc:seg4_xducercontactprobe_vel** – Transducer contact probe velocity.
- **dc:seg4_laser_displ** – Displacement at laser vibrometer spot.
- **dc:seg4_laser_vel** – Velocity at laser vibrometer spot.
- **dc:seg4_crackcenterstress** – Crack center stress.
- **dc:endcrop** – The amount of time in seconds to crop off the generated time domain waveforms to remove the anticausal portion of the signal.

Return dc:dynamicmodel Time-domain specimen stress and motion at transducer contact, laser, and crack locations

vibrocomsol_multisweep_seg_analysis_comsol()

This processtrak step will run one of the multiple sweep analyses prescribed by *vibrosim_optimize_freqbands*.

Provided by the `VibroSim_Simulator` package.

Parameters

- **`dc:dest`** – Results output folder.
- **`dc:measident`** – Measurement identifier.
- **`dc:model_comsol_withsegboundaries`** – Save file for the COMSOL model with segment boundaries.
- **`segnum_int`** – Segment number.

Return dc:segX_xducercontactprobe_displ Transducer contact probe displacement.

Return dc:segX_xducercontactprobe_vel Transducer contact probe velocity.

Return dc:segX_laser_displ Displacement at laser vibrometer spot.

Return dc:segX_laser_vel Velocity at laser vibrometer spot.

Return dc:segX_crackcenterstress Crack center stress.

`vibrocomsol_set_freqbands_comsol``()`

This ProcessTrak step opens a COMSOL file and sets the frequency bands for the multisweep study.

Provided by the `VibroSim_Simulator` package.

Parameters

- **`dc:dest`** – Results output folder.
- **`dc:measident`** – Measurement identifier.
- **`dc:model_comsol`** – Save file for the COMSOL model.
- **`dc:freqband_seg1_start`** – Starting frequency of a frequency band.
- **`dc:freqband_seg1_step`** – Step frequency of a frequency band.
- **`dc:freqband_seg1_end`** – End frequency of a frequency band.
- **`dc:freqband_seg2_start`** – Starting frequency of a frequency band.
- **`dc:freqband_seg2_step`** – Step frequency of a frequency band.
- **`dc:freqband_seg2_end`** – End frequency of a frequency band.
- **`dc:freqband_seg3_start`** – Starting frequency of a frequency band.
- **`dc:freqband_seg3_step`** – Step frequency of a frequency band.
- **`dc:freqband_seg3_end`** – End frequency of a frequency band.
- **`dc:freqband_seg4_start`** – Starting frequency of a frequency band.
- **`dc:freqband_seg4_step`** – Step frequency of a frequency band.
- **`dc:freqband_seg4_end`** – End frequency of a frequency band.

Return dc:model_comsol_withsegboundaries Save file for the COMSOL model with segment boundaries.

`vibrosim_optimize_freqbands``()`

This ProcessTrak step optimizes the frequency bands for the ProcessTrak step *`vibrocomsol_multisweep_seg_analysis_comsol`*.

Run this on output of modal analysis to interpret the modal decay coefficients and plan a three or four segment frequency domain calculation that will be invertable to a time-domain response.

This step prepares the model for a frequency sweep taken in multiple parts with varying time steps. The frequency bands are chosen to include the modes present in the modal analysis, and the frequency step is chosen to avoid aliasing. A large frequency step can be used for segments with modes that are expected to damp quickly, and small frequency steps can be used for segments with modes that damp slowly. Thereby removing the need for a small frequency step across the whole spectrum.

Parameters

- **`dc:modalfreqs`** – Frequencies of the eigenmodes.
- **`dc:temporal_decay_divisor`** – The factor by which the time domain impulse response should decay within the calculation period. Residual magnitudes past the calculation period implicitly wrap back and overlap with the impulse response, acting as interference.
- **`dc:spectral_decay_divisor`** – The factor by which resonances outside a segment under construction must decay to by a segment boundary in order to be ignored when constructing the segment.

Return dc:freqband_seg1_start Starting frequency of a frequency band.

Return dc:freqband_seg1_step Step frequency of a frequency band.

Return dc:freqband_seg1_end End frequency of a frequency band.

Return dc:freqband_seg2_start Starting frequency of a frequency band.

Return dc:freqband_seg2_step Step frequency of a frequency band.

Return dc:freqband_seg2_end End frequency of a frequency band.

Return dc:freqband_seg3_start Starting frequency of a frequency band.

Return dc:freqband_seg3_step Step frequency of a frequency band.

Return dc:freqband_seg3_end End frequency of a frequency band.

Return dc:freqband_seg4_start Starting frequency of a frequency band.

Return dc:freqband_seg4_step Step frequency of a frequency band.

Return dc:freqband_seg4_end End frequency of a frequency band.

CHAPTER

SIX

EXPERIMENT LOG PARAMETERS

This chapter lists the experiment log parameters most likely to be needed in your `.xlg` file. These are used as inputs to the various steps.

6.1 dest

Where to save the generated output.

```
<dc:dest xlink:type="simple" xlink:href="vibrosim_demo3_output/"/>
```

6.2 measident

Used as a filename prefix for the various output files generated. It should be redefined in submeasurements if they are being used.:

```
<dc:measident>meas1</dc:measident>
```

6.3 amplitude

`amplitude` defines the displacement amplitude of the transducer. A calibration must be done to relate the voltage in the electronics to the displacement of the transducer as a function of frequency. A constant 10 micron/Volt calibration file is included in VibroSim COMSOL.:

```
<dc:amplitude dcv:units="Volts">3</dc:amplitude>
```

6.4 friction_coefficient

Friction coefficient of the crack surface. Used to calculate the heat generated by the crack. These numbers may not match a direct friction measurement because they include non-idealized behavior, but they are interpreted as friction coefficients:

```
<dc:friction_coefficient>.0661</dc:friction_coefficient>
```

Friction coefficient values for Ti 6-4 and Inconel 718 were evaluated empirically in late 2019/early 2020 in testing at Iowa State Unversity to be .066 and 0.146 respectively. Be warned that these values are not necessarily applicable to your material system.

6.5 msqrtR

Crack heating in mode I motion can be attributed to angled friction, where angled surfaces rub against each other while the crack as a whole opens and closes cleanly. The `msqrtR` is product of the asperity density and the typical asperity radius.

```
<dc:msqrtR dcv:units="m^-1.5">3.14e6</dc:msqrtR>
```

msqrtR values for Ti 6-4 and Inconel 718 were evaluated empirically in late 2019/early 2020 in testing at Iowa State Unversity to be 3.14*10^6 m^(-3/2) and 6.77*10^6 m^(-3/2) respectively. Be warned that these values are not necessarily applicable to your material system.

6.6 crack_model_shear_factor

Sensitivity factor for shear vs normal heating. This was included in the model just in case empirical observation warranted it. Based on our observations, heating due to normal vibration is on the same order as heating due to the same magnitude of shear vibration, so using 1.0 here is reasonable.

```
<dc:crack_model_shear_factor>1.0</dc:crack_model_shear_factor>
```

6.7 staticload_mount

Combined static load on mounts.

```
<dc:staticload_mount dcv:units="N">0</dc:staticload_mount>
```

6.8 staticload

Static bending opening load on crack. Used in determining closure state

```
<dc:staticload dcv:units="Pascals">0</dc:staticload>
```

6.9 xducerforce

Force of with which the transducer is pressed into the specimen. Used in model creation when the couplant is attached to the model. Only used in the `solidmech_static` COMSOL study or by the welder model:

```
<dc:xducerforce dcv:units="Newtons">0</dc:xducerforce>
```

6.10 tortuosity

The growth mechanisms of a fatigue crack usually result in a rough crack surface. The roughness is further exaggerated for intergranular crack growth wherein the crack growth path can deviate from the growth plane and follow the grain boundaries of several grains until the macroscopic applied load (typically mode-I loading) coerces the local crack trajectory to follow the symmetric growth plane. This process results in a fatigue crack, with planar macroscopic trajectory, but with a very rough and tortuous local trajectory.

Crack tortuosity is quantified as the standard deviation of a distribution of angles of node points on the crack with taken with respect to a line connecting the end points of the crack.

```
<dc:tortuosity dcv:units="degrees">30.5</dc:tortuosity>
```

6.11 numdraws

Crack tortuosity is a statistical distribution, this parameter defines how many draws to take at each position along the crack for calculating the crack heating.

```
<dc:numdraws>100</dc:numdraws>
```

6.12 spcmaterial

String defining the specimen material. This is used in building the COMSOL model to determine the names of the material parameters for the specimen.:

```
<dc:spcmaterial>Ti 6-4</dc:spcmaterial>
```

6.13 YoungsModulus

Youngs modulus of the specimen material.

```
<dc:YoungsModulus dcv:units="Pascals">117.9e9</dc:YoungsModulus>
```

6.14 YieldStrength

Yield strength of the specimen material.

```
<dc:YieldStrength dcv:units="Pascals">944.58e6</dc:YieldStrength>
```

6.15 PoissonsRatio

Poissons ratio of the specimen material.

```
<dc:PoissonsRatio>0.342</dc:PoissonsRatio>
```

6.16 Density

Density of the specimen material.

```
<dc:Density dcv:units="kg/m^3">4430</dc:Density>
```

6.17 spcThermalConductivity

Thermal conductivity of the specimen material.

```
<dc:spcThermalConductivity dcv:units="W/m/K">6.7</dc:spcThermalConductivity>
```

6.18 spcSpecificHeatCapacity

Specific heat capacity of the specimen material.

```
<dc:spcSpecificHeatCapacity dcv:units="J/kg/K">526.3</dc:spcSpecificHeatCapacity>
```

6.19 simulationcameranetd

Magnitude of simulated camera noise: noise equivalent temperature difference (NETD).

```
<dc:simulationcameranetd dcv:units="K">.022</dc:simulationcameranetd>
```

6.20 spcrayleighdamping_alpha

Rayleigh damping coefficient alpha. Rayleigh damping has components that are proportional to the mass and to the stiffness matrices of the specimen. Alpha is the coefficient of the mass matrix in the equation. Conceptually, increasing alpha will increase the damping in the low frequencies.

```
<dc:spcrayleighdamping_alpha dcv:units="s^-1">2</dc:spcrayleighdamping_alpha>
```

6.21 spcrayleighdamping_beta

Rayleigh damping coefficient alpha. Rayleigh damping has components that are proportional to the mass and to the stiffness matrices of the specimen. Beta is the coefficient of the stiffness matrix in the equation. Conceptually, increasing beta will increase the damping in the high frequencies.

```
<dc:spcrayleighdamping_beta dcv:units="s">5e-9</dc:spcrayleighdamping_beta>
```

6.22 mass_of_welder_and_slider

Mass of the ultrasonic welder assembly, for the ultrasonic welder model.:

```
<dc:mass_of_welder_and_slider dcv:units="kg">2.0</dc:mass_of_welder_and_slider>
```

6.23 pneumatic_force

Force holding the ultrasonic welder assembly to the specimen, as used in the welder model. Contrast with `dc:xducerforce`.

```
<dc:pneumatic_force dcv:units="N">100</dc:pneumatic_force>
```

6.24 welder_elec_ampl

Electrical amplitude driving the ultrasonic welder, arbitrary units. Please note that the driving characteristics of the ultrasonic welder model are expected to change in a future version of VibroSim.

```
<dc:welder_elec_ampl>1e8</dc:welder_elec_ampl>
```

6.25 welder_spring_constant

The welder spring constant represents the springiness of the welder mounts, usually coming from the rubber pneumatic seals.

```
<dc:welder_spring_constant dcv:units="N/m">5000</dc:welder_spring_constant>
```

6.26 R_contact

R_contact represents the effective contact radius, from a Hertzian contact perspective, that gives rise to compliance between the

<dc:R_contact dcv:units="m">25.4e-3</dc:R_contact>

6.27 couplantx

This parameter to some _comsol.m build scripts provides the x coordinate of the welder or transducer contact.

```
<dc:couplantx dcv:units="m">.245</dc:couplantx>
```

6.28 couplanty

This parameter to some _comsol.m build scripts provides the y coordinate of the welder or transducer contact.

```
<dc:couplanty dcv:units="m">.025</dc:couplanty>
```

6.29 couplantz

This parameter to some _comsol.m build scripts provides the z coordinate of the welder or transducer contact.

```
<dc:couplantz dcv:units="m">0</dc:couplantz>
```

6.30 couplantangle

This parameter to some _comsol.m build scripts provides the orientation coordinate of the transducer contact if the transducer has a square tip. Otherwise (and for ultrasonic welder contact in general) it should be specified as NaN.

```
<dc:couplantangle dcv:units="degrees">NaN</dc:couplantangle> <!-- NaN means use round␣
↪couplant (i.e. excitation zone) -->
```

6.31 crack_model_normal

Which crack closure model to use for normal loading. `ModeI_throughcrack_CODformula` or `Tada_ModeI_CircularCrack_along_midline`.

```
<dc:crack_model_normal>Tada_ModeI_CircularCrack_along_midline</dc:crack_model_normal>
```

6.32 crack_model_shear

Which stick/slip model to use for shear loading. `Fabrikant_ModeII_CircularCrack_along_midline`, `ModeII_throughcrack_CSDformula`, or `ModeIII_throughcrack_CSDformula`.

```
<dc:crack_model_shear>Fabrikant_ModeII_CircularCrack_along_midline</dc:crack_model_
↪shear>
```

6.33 crack_shearstress_axis

Axis on which to calculate the shear stress. This can be either `major` (representing stress exhibited in mode II crack displacment) or `minor` (representing stress exhibited in mode III crack displacement).

```
<dc:crack_shearstress_axis>major</dc:crack_shearstress_axis>
```

6.34 crack_type_side1

Crack type of crack side number one (negative side of the crack major axis). This can be either `halfthrough`, representing a through thickness crack, or `quarterpenny`, representing a crack that is elliptical in shape going into the surface of the specimen.

```
<dc:crack_type_side1>quarterpenny</dc:crack_type_side1>
```

6.35 crack_type_side2

Crack type of crack side number two (positive side of the crack major axis). This can be either `halfthrough`, representing a through thickness crack, or `quarterpenny`, representing a crack that is elliptical in shape going into the surface of the specimen.

```
<dc:crack_type_side2>quarterpenny</dc:crack_type_side2>
```

6.36 thickness

Thickness of the material at the through crack. The geometry in COMSOL is not so well integrated that this will not be populated automatically. This should be an average thickness if thickness is not constant across the crack.

```
<dc:thickness dcv:units="meters">1e-3</dc:thickness>
```

6.37 reff_side1

`reff_side1`, `seff_side1`, `reff_side2`, `seff_side2` are one way to specify the closure state of the crack: by radius of closure point at different external stress levels. These are interpreted in the context of the selected crack model_normal and need to be preprocessed by the `eval_closure_state_from_tip_positions` step into a `.csv` table with closure stress and initial opening displacement along the crack.

```
<dc:reff_side1 dcv:units="m" dcv:arraystorageorder="C"><dcv:arrayshape>9</
↪dcv:arrayshape><dcv:arraydata>.5e-3 .7e-3 .9e-3 1.05e-3 1.2e-3 1.33e-3 1.45e-3 1.
↪56e-3 1.66e-3</dcv:arraydata></dc:reff_side1>
```

6.38 seff_side1

`reff_side1`, `seff_side1`, `reff_side2`, `seff_side2` are one way to specify the closure state of the crack: by radius of closure point at different external stress levels. These are interpreted in the context of the selected crack model_normal and need to be preprocessed by the `eval_closure_state_from_tip_positions` step into a `.csv` table with closure stress and initial opening displacement along the crack.

```
<dc:seff_side1 dcv:units="Pa" dcv:arraystorageorder="C"><dcv:arrayshape>9</
↪dcv:arrayshape><dcv:arraydata>0.0 50e6 100e6 150e6 200e6 250e6 300e6 350e6 400e6</
↪dcv:arraydata></dc:seff_side1>
```

6.39 reff_side2

`reff_side1`, `seff_side1`, `reff_side2`, `seff_side2` are one way to specify the closure state of the crack: by radius of closure point at different external stress levels. These are interpreted in the context of the selected crack model_normal and need to be preprocessed by the `eval_closure_state_from_tip_positions` step into a `.csv` table with closure stress and initial opening displacement along the crack.:

```
<dc:reff_side2 dcv:units="m" dcv:arraystorageorder="C"><dcv:arrayshape>9</
↪dcv:arrayshape><dcv:arraydata> .5e-3 .7e-3 .9e-3 1.05e-3 1.2e-3 1.33e-3 1.45e-3 1.
↪56e-3 1.66e-3</dcv:arraydata></dc:reff_side2>
```

6.40 seff_side2

`reff_side1`, `seff_side1`, `reff_side2`, `seff_side2` are one way to specify the closure state of the crack: by radius of closure point at different external stress levels. These are interpreted in the context of the selected crack model_normal and need to be preprocessed by the `eval_closure_state_from_tip_positions` step into a `.csv` table with closure stress and initial opening displacement along the crack.

```
<dc:seff_side2 dcv:units="Pa" dcv:arraystorageorder="C"><dcv:arrayshape>9</
↪dcv:arrayshape><dcv:arraydata> 0.0 50e6 100e6 150e6 200e6 250e6 300e6 350e6 400e6</
↪dcv:arraydata></dc:seff_side2>
```

6.41 closurestate_side1

`closurestate_side1` and `closurestate_side2` are hypertext references to `.csv` files representing the closure state of e

<dc:closurestate_side1 xlink:type=”simple” xlink:href=”crackclosure.csv”/>

6.42 closurestate_side2

`closurestate_side1` and `closurestate_side2` are hypertext references to `.csv` files representing the closure state of e
<dc:closurestate_side2 xlink:type="simple" xlink:href="crackclosure.csv"/>

6.43 cracksemimajoraxislen

Half-crack length along the surface (major) axis for a surface crack, or full crack length along the surface (major) axis for an ed
<dc:cracksemimajoraxislen dcv:units="m">1.66e-3</dc:cracksemimajoraxislen>

6.44 cracksemiminoraxislen

Crack length along the depth (minor) axis for a surface crack, in meters (`thickness` is used instead for edge cracks``). It deter
<dc:cracksemiminoraxislen dcv:units="m">1.66e-3</dc:cracksemiminoraxislen>

6.45 exc_t0

Start of envelope ramp-up.

```
<dc:exc_t0 dcv:units="s">0.0</dc:exc_t0>
```

6.46 exc_t1

End of enelope ramp-up.

```
<dc:exc_t1 dcv:units="s">0.02</dc:exc_t1>
```

6.47 exc_t2

Start of envelope ramp-down

```
<dc:exc_t2 dcv:units="s">0.98</dc:exc_t2>
```

6.48 exc_t3

End of envelope ramp-down.

```
<dc:exc_t3 dcv:units="s">1.00</dc:exc_t3>
```

6.49 exc_t4

End of vibration calculation.

```
<dc:exc_t4 dcv:units="s">1.00</dc:exc_t4>
```

INDEX