VERA Virtual Environment for Reactor Applications

ORNL/SPR-2022/2480

VERAIn Programmer's Manual

May 31, 2022

Erik Walker[1]

[1]Oak Ridge National Laboratory

OAK RIDGE National Laboratory

U.S. DEPARTMENT OF ENERGY

ORNL is managed by UT-Battelle LLC for the US Department of Energy.

ORNL/SPR-2022/2480

VERAIN PROGRAMMER'S MANUAL

Erik Walker[1]

[1]Oak Ridge National Laboratory

Date Published: May 31, 2022

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

VERAIn Programmer's Manual

Revision Log

Revision	Date	Affected Pages	Revision Description
0	5/31/2022	All	New template and editorial updates for VERA 4.3 release. This version supersedes previous document version CASL-U-2019-1884-001.

Document pages that are:

Export Controlled:	None
IP/Proprietary/NDA Controlled:	None
Sensitive Controlled:	None
Unlimited:	All

VERAIn Programmer's Manual

Approvals:

Erik Walker	08/04/2022
Erik Walker, VERAIO Product Software Manager	**Date**
	08/07/2022
Andrew Godfrey, Independent Reviewer	**Date**

EXECUTIVE SUMMARY

This document provides information needed by developers who will be working in the VERAIn source code. It provides a high-level overview of input design, as well as guidelines for adding new inputs.

CONTENTS

1. VERAIN

The VERAIn repository is used to translate the user-defined ASCII input file to a ParameterList XML file to be read by codes within VERA. The ParameterList format is defined in Trilinos Teuchos package at https://docs.trilinos.org/dev/packages/teuchos/doc/html/index.html#TeuchosParameterList_src. The VERAIn parser is a PERL script that translates the individual variables specified in YAML files. These YAML files are located in the `VERAIO/verain/scripts/Templates` directory. To perform the translation, use the parser located in the scripts directory. Run the parser with a command line like the following:

```
VERAIO/verain/scripts/react2xml.pl $CASE.inp $CASE.xml
```

where `$CASE` is the base name of the input file. Parser error messages indicate errors in the input file. Various input parameter examples can be found in the `VERAIO/verain/test` directory.

The react2xml command line options can be viewed by using

```
react2xml.pl --help
```

resulting in:

```
Converts reactor input file (reactor_input_file) into ParameterList
XML file (output_xml_file). ParameterList format is defined in
Trilinos Teuchos package: https://docs.trilinos.org/dev/packages/teuchos/doc/html/
index.html#TeuchosParameterList_src

Program error messages indicate errors in the input file.

Program switches:

--help    This help.

--xml=(on|off)
          Write XML declaration as the first line of the ParameterList
          XML file. Default=on.

--xslt=(on|off)
          Write XSLT processing instruction as the second line of the
          ParameterList XML file. Turns on --xml switch, as well. XSLT
          style file PL9.xsl should be in the same directory as the
          resulting ParameterListXML file in order for transformation
          to work in a browser. Default=on.

--verbose Add processing printouts as the code executes.

--debug   Create debug files for finding the errors in the converter
          program. Does not help much in tracing invalid input.

--schema Write the VERAIn input schematic to STDOUT formatted as JSON.
```

2. VERAIN OUTPUT

Upon completion, several output files may be created, depending on the code options used. Some typical outputs include the following:

- **VERAIn XML file:** file written upon the successful completion of VERAIn
- **VERA HDF output file:** a binary file with results that can be visualized in VERAView or post-processed with user utility codes
- **MPACT output file:** file written upon the successful completion of MPACT (if applicable)
- **MPACT log file:** file written upon the successful completion of MPACT (if applicable)
- **MPACT summary file:** file written upon the successful completion of MPACT (if applicable).
- **standard output file:** a log of all output written to the standard output
- **standard error file:** a log of all output written to the standard error file

In case of errors, the user should examine the standard error file first. If there are any errors in the input processor (VERAIn), then they will be written here. Any error messages from other VERA components will also be written to the standard error file.

If the standard error file does not list any errors, then check the VERA component output files and log files for messages.

3. STYLE GUIDE

3.1 ADDING NEW VARIABLES

This section provides instructions for adding or modifying inputs to be used in the user-provided ASCII input. For most cases, to add a new variable to the input, only the template files found in `VERAIO/verain/scripts/Templates` must be modified. Generally, two files must be modified for every new input variable:

- Directory.yml—describes how to read the input cards
- BLOCK.yml—describes how to convert input cards to parameter names for a particular BLOCK

The different BLOCK options available in an input file are:

`STATE, CORE, ASSEMBLY, CONTROL, INSERT, DETECTOR, EDITS, SHIFT, COBRATF, COUPLING, MPACT, BISON, MAMBA, and RUN`

For example, if a new variable is to be added to the ASSEMBLY block of the user input, then it must be added to the ASSEMBLY section of the Directory.yml file, as well as to the ASSEMBLY.yml file.

If a new variable is added to the Directory.yml file, proper documentation must also be provided. The input description must precede the variable and must be in the following form using LaTeX formatting:

```
#>
#>   {\bf variable\_name} detailed\_input\_name
#>   \VERAInputTable{
#>     name={detailed\_input\_name},
#>     type={variable type: Float, Boolean, Integer... etc.},
#>     need={Optional or Required},
#>     unitsdefault={Default units for this variable},
#>     unitsother={Other acceptable units for this variable},
#>     valuedefault={Default value for this variable, if there is one},
#>     valuesapplicable={Defines acceptable values, or range of values, that this variable
#>        can be defined as},
#>     limitations={A description of the limitations of this variable, if applicable},
#>     description={A detailed description of this variable and what it is used for.},
#>     notes={Any additional information about this variable that was not specified earlier}
#>   }
```

It is strongly advised that a new variable include a `_check:` section that checks to ensure that a given variable is the proper type and within the expected input variable range. An example of a two-variable input card with description is shown in Figure 1. This example input, `b10`, is located in the STATE block. Therefore, the corresponding entries in the STATES.yml file are shown in Figure 2. The names given in the BLOCK.yml file will correspond to the variable name that will be given in the ParameterList XML file.

3.2 TESTING NEW VARIABLES

If a new input card is added, then a test must also be added. To add a test, the user must either modify an existing test or add a new one in the `VERAIO/verain/test` directory. It is preferable to modify an existing test, but when doing so, care must be taken not to impact another test that might use the same input.

A special test input deck called `misc_options.inp` contains several random input variables. The only purpose of this test is to make sure input is written to the XML file correctly. If possible, this test should

```
#>
#>    {\bf b10} b10\_fraction b10\_depletion
#>    \VERAInputTable{
#>      name={b10\_fraction},
#>      type={Float},
#>      need={Optional},
#>      unitsdefault={},
#>      unitsother={Atom fraction of B-10 in boron},
#>      valuedefault={0.199},
#>      valuesapplicable={$>=0$},
#>      limitations={},
#>      description={Boron-10 fraction in coolant},
#>      notes={}
#>    }
#>    \VERAInputTable{
#>      name={b10\_depletion},
#>      type={Boolean},
#>      need={Optional},
#>      unitsdefault={},
#>      unitsother={},
#>      valuedefault={False},
#>      valuesapplicable={True},
#>      limitations={},
#>      description={Flag to enable B-10 depletion in coolant},
#>      notes={Required when using input parameter \texttt{b10} }
#>    }
  b10:
    <<: *dtlist
    _content:
    _check:
      - arraysize()==2
      - is_float(nth(0,()))
      - is_word(nth(1,()))
      - nth(0,())>=0
```

Figure 1. Example b10 inputs and descriptions in Directory.yml

```
b10:
  _pltype: parameter
  _type: double
  _do:
    - copy %STATE/$(_loop)/$b10:0
  _content:

b10_depl:
  _pltype: parameter
  _type: bool
  _do:
    - copy %STATE/$(_loop)/$b10:1
  _content:
```

Figure 2. Example b10 inputs in STATES.yml

be used instead of creating a new test. Once a test has been modified or created, a "gold" xml file must be created and placed in the `VERAIO/verain/test` directory to accompany the test. If a new test is added, then the `CMakeLists.txt` file in the `VERAIO/verain/test` directory must be modified, and another entry must be added to the long list of `ADD_REACTOR_AWARE_INPUT_PARSER_TEST` cases.

4. VERARUN

VERARun is a set of Python scripts that allows for the easy execution of problems in VERA. The scripts are located in the VERAIO repository in the `VERAIO/verarun` directory.

To execute a problem using VERARun, a user must type the following:

```
verarun $CASE
```

Additional command line options can be found by typing `verarun` with no arguments to return the following options:

```
usage: verarun [--devs] [-x] [--schema] [-e email_addr] [-h] [-c config_file]
               [-N job_name] [-l] [-n nprocs] [-O] [--ppn cpus_per_node]
               [-m mem_per_process] [-p project] [-q queue] [-s subdir]
               [-d vera_install_dir] [-v vera_version] [--verbose] [-W]
               [-w job_id] [-t walltime] [--chain] [--debug] [--hostname host]
               [-r {overwrite,readwrite}]
               [--vera-installs-dir vera_installs_dir]
               [input_path [input_path ...]]

Creates and optionally submits machine-specific VERA jobs.

positional arguments:
  input_path            path to VERA input (.inp) or XML (.xml) files

optional arguments:
  --devs, --allow-devs  override VERA_PROD_VERSIONS and allow development VERA
                        versions, implies -l
  -x, --dry-run         dry run only, create but don't execute the PBS script
  --schema              schema from react2xml.pl
  -e email_addr, --email-addrs email_addr
                        comma-delimited list of email addresses to notify of
                        job completion, defaults to ${USER}@$(hostname)
  -h, --help            print detailed help
  -c config_file, --host-config-file config_file
                        override host configuration file, supercedes
                        --hostname and --vera-installs-dir
  -N job_name, --job-name job_name
                        name for the PBS job
  -l, --list-vera-versions
                        list available VERA versions
  -n nprocs, --np nprocs, --nprocs nprocs, --num-procs nprocs
                        total number of processors need for the MPACT run
                        (mpiexec -np param), defaults to value computed from
                        input
  -O, --output-job-name
                        print the job id to stdout
  --ppn cpus_per_node, --pnode cpus_per_node
```

```
                        specify processors per node, by default this is
                        calculated
  -m mem_per_process, --pmem mem_per_process, --proc-memory mem_per_process
                        specify memory required per processor in GB, defaults
                        to undefined
  -p project, --project project
                        optional project or account to specify for the job,
                        overriding any default, where a value of "none" omits
                        a project
  -q queue, --partition queue, --queue queue
                        Torque queue or Slurm partition
  -s subdir, --subdir subdir
                        create subdir, a value of "." specifies automatically
                        generated subdir name
  -d vera_install_dir, --vera-dir vera_install_dir
                        path to VERA installation, superceding --vera-
                        installs-dir, --vera-version, and the host
                        configuration
  -v vera_version, --vera-version vera_version
                        name of VERA version to use
  --verbose             turn on verbose messaging
  -W                    wait on job last submitted via verarun, overrides -w
  -w job_id, --wait-job-id job_id
                        ID of job which must complete before starting this job
  -t walltime, --wall-time walltime
                        wallclock execution time in floating point hours,
                        defaults to 24.0

advanced arguments:
  --chain, --chain-jobs
                        each job depends on its predecessor
  --debug               debug mode
  --hostname host       force the hostname
  -r {overwrite,readwrite}, --restart {overwrite,readwrite}
                        optional restart mode
  --vera-installs-dir vera_installs_dir
                        path to vera_installs directory containing VERA
                        versions

Version 1.12
```

5. ACKNOWLEDGEMENTS

This research was supported by the US Department of Energy and the Nuclear Energy Advanced Modeling and Simulation Program.