|  |  |
| --- | --- |
| Issue | 2.0 |
| Topic | Tealeaves quick guide |
| Description | How to run Tealeaves tool and analyze its output |

History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Remark | Status |
| 1.0 | 04.03.2021 | Tool-Hotline Tealeaves | Initial Version | - |
| 2.0 | 05.01.2022 | Phan Phuoc Dien (SX/BSV21-EA) | Added more command syntax | - |

Contents

[1. Overview 3](#_Toc92287571)

[1.1. Tool Installation 3](#_Toc92287572)

[1.2. Tealeaves UseCases 3](#_Toc92287573)

[1.3. Tool Usage 5](#_Toc92287574)

[1.4. Summary 6](#_Toc92287575)

[2. Command syntax: 7](#_Toc92287576)

[3. Output files explaination 9](#_Toc92287577)

[4. Headless mode / swb\_shell Invocation: 11](#_Toc92287578)

[5. Known Restrictions: 11](#_Toc92287579)

[6. Help: 12](#_Toc92287580)

[7. Others: 12](#_Toc92287581)

# Overview

Tealeaves is a tool used for in identifying possible Software (SW) Build Abort by parsing and analyzing PVER source. It also solves the error validations before SW Build triggers.

Its core functionalities include:

* Support for MDGB / DGSB
* Detect and filter inherited errors (e.g. missing CompuMethod or Systemconstant, which may lead to other errors)
* User can change or add Systemconstants by editing "\_tealeaves/syscons\_defines.txt" in CLI version.
* Parsed-resources caching
* Solve missing interfaces and processes
* Provide framework for automated validation of internal "teaLeaves test cases"
* Source generator used by teaLeaves virtually (e.g. generates DELib files and many others)

## Tool Installation

Tealeaves is available in Toolbase Environment Explorer under DGS environment

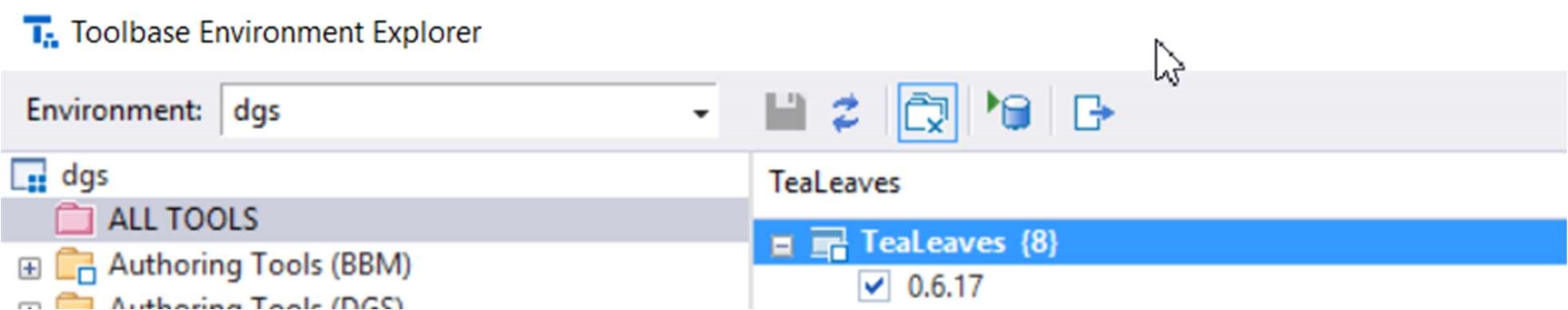


Figure Install Tealeaves

Select latest version and run Toolbase Update as shown in above image. Once installed, the tool is available in C:\toolbase\tealeaves\<version>. It can be invoked in headless mode.

## Tealeaves UseCases

Tealeaves detects build errors which occur during SW build listed below (called Checker Mode):

* Missing Systemconstant
* Missing required processes
* Multiple definition of DSM elements
* Conditions/Values with incorrect syntax
* CoreProc: Missing EEP Subblocks in Masterlayout Configuration
* Double definition of Elements
* Missing open Interface

Tealeaves solves SW Build errors which occur during SW build listed below (called Solver Mode):

* Missing Systemconstant:
  + If the Systemconstant definition is missing but it is defined in iDB file -> after solving mode it will be added into **‘gconf\_sy\_pavast\_solved.xml’** (this only applicable for ADJUSTABLE category, more detail go to *Category types* section)
  + If the Systemconstant definition is missing and it is **NOT** defined in iDB file -> after solving mode it will be added into **‘swadp\_pavast\_solved.xml’**

Example steps to solve missing Systemconstant error:

1. Add your Systemconstant in this file:

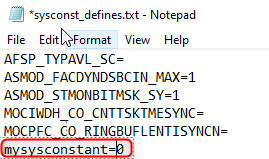


Figure Sysconst\_defines.txt

1. Run “tealeaves --solve”
2. After solving, the Systemconstant will be added to either swadp\_pavast\_solved or gconf\_sy\_pavast\_solved:

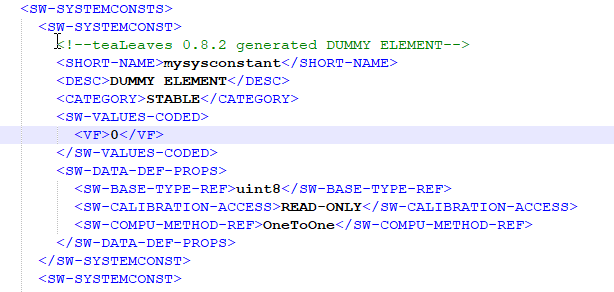


Figure After solving

* Missing required processes:
  + Process are stubbed based on reference TDATA file in SC PaVaSt.
  + If reference file is not available, process will be added based on read-write analysis
* Double definition of Element
* Missing open Interface (Message, CALPRM, Class, Compu-method )
* Different ways to stub Interfaces:
  + .idb files for local PVER/BC/FC reference
  + .csv file

*Figure 8 Example .csv input file*

* + Architecture-Front-Planning-generated .xlxs files

*Figure 9 Example AFP .xlsx file*

* + Dummy definition

## Tool Usage

Follow below steps to run Tealeaves tool in Checker Mode:

* Open the command prompt (cmd) in your project folder or any other folder then invoke tealeaves tool using ‘tini tealeaves latest’.
* Type “tealeaves” if current directory is PVER root path or type “tealeaves -r <PVERPATH>” to trigger tealeaves for a specific PVER.

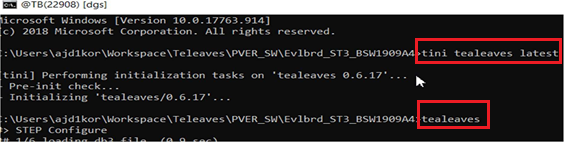


Figure How to run Tealeaves

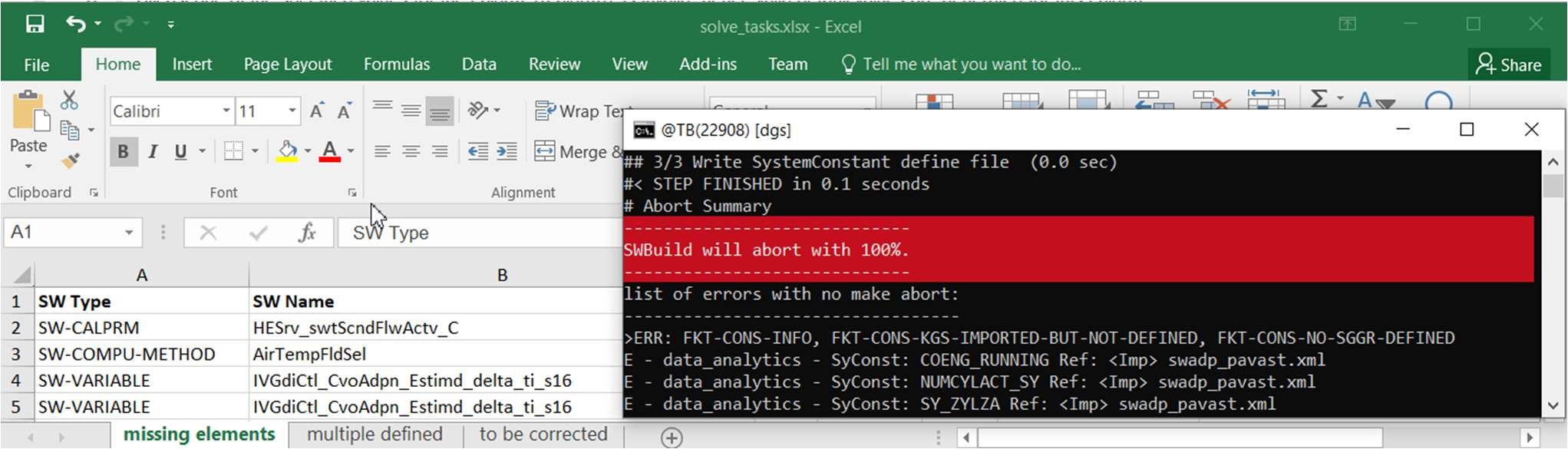
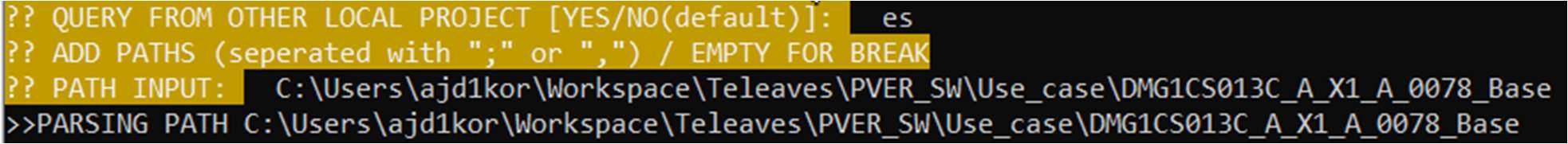
\*Note: If any error is found, abort message will be highlighted in RED background color. In that case, kindly look at “<PVERPATH>\\_tealeaves\solve\_tasks.xlsx” for more details about possible build errors.

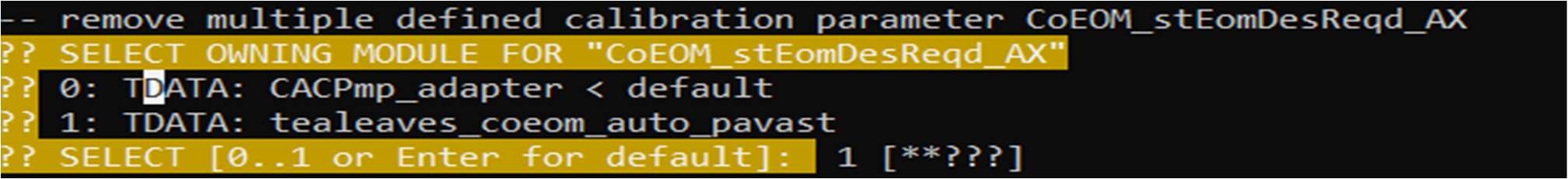
Figure Run finish with errors

Steps to run Tealeaves in Solver Mode:

* Open the command prompt (cmd) in your project folder or any other folder then invoke tealeaves tool using ‘tini tealeaves latest’.
* Type “tealeaves --solve” if current directory is PVER root path or type “tealeaves -r <PVERPATH> --solve” to trigger tealeaves for a specific PVER.
* If any error is found, it will do below steps:
  + Ask user to input a PVER for local references if there is any. Otherwise it will take from .idb file.



* + Ask user to enter values for Systemconstant.
  + Add missing Interfaces and required Proc.
  + Ask if user wants to add dummy definition when definitions are not found.
  + Ask user to remove double definition



\*Note: The newly created file will be named “XXXX\_pavast\_solved.xml”. You must compare and replace the content in original file before running SW build or it will produce same error.

Steps to run Tealeaves in Solver Mode-GUI:

* In Command Prompt run command “tealeaves -r <PVERPATH> --solve --solve-gui” as shown below:

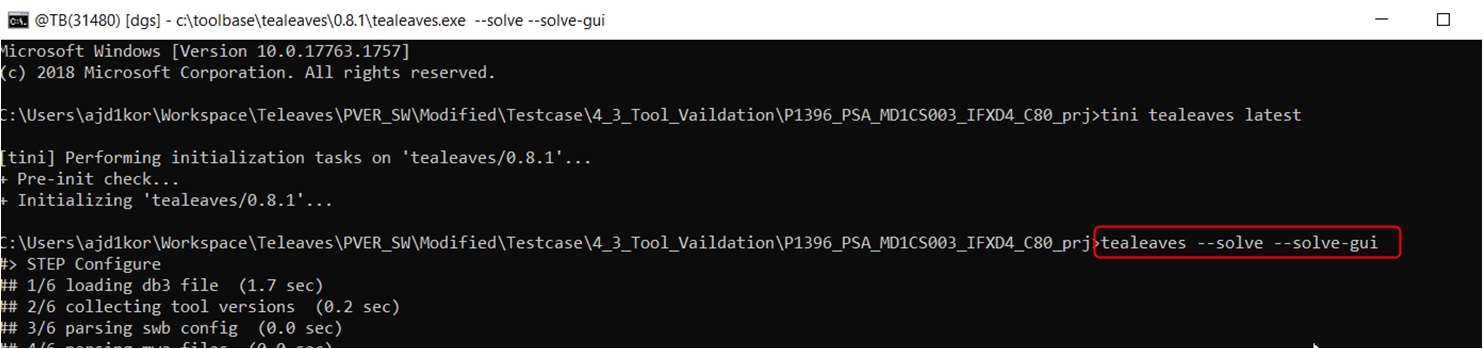


Figure Solver Mode GUI option

* UI window will popup if abort errors are detected:

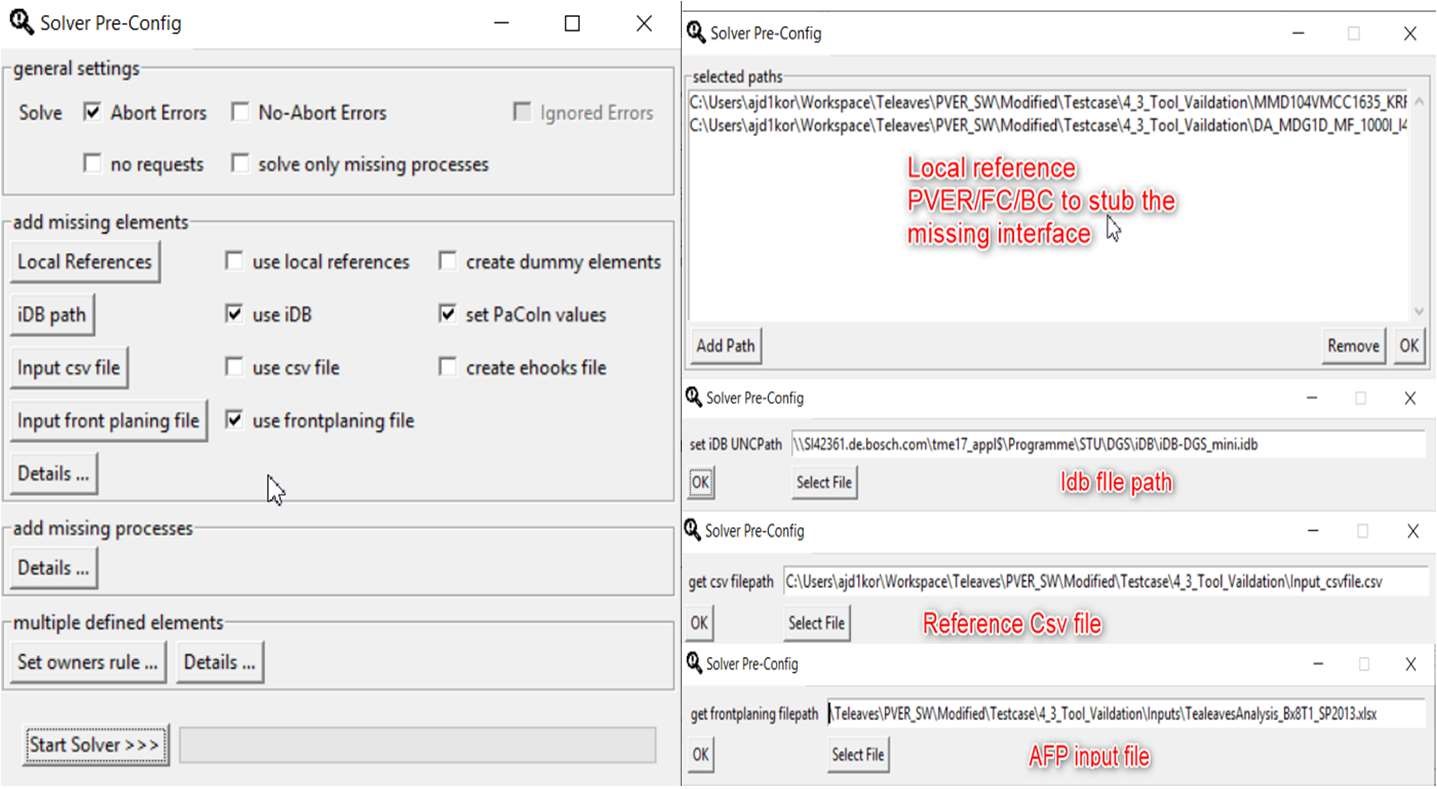


Figure Tealeaves Solver Mode GUI

## Summary

After running “tealeaves” command, if swb\_build abort predicted as 100 percent and solve\_tasks.xlsx is not empty => your project has a build abort therefore it needs correction.

However, in case no sw\_build abort is found and solve\_tasks.xlsx file is empty and=> then SW Build is most likely to go through.

# Command syntax:

Usage: **tealeaves --<command> [--<command-options>]**

The following commands are available:

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Command option** | **Description** | **Example** |
| –solve | N/A | Solve issues interactively. | tealeaves --solve |
| --solve-no-request | Use always default values option for the interface. | tealeaves --solve --solve-no-request |
| –solve-frontplanning-input=<FILE\_PATH> | To use architecture front planning (AFP) input for stubbing. | tealeaves --solve –solve-frontplanning-input=path/to/file/fp.xlsx |
| --solve-add-dummy | Add dummy if not found. | tealeaves --solve --solve-add-  dummy |
| --solve-no-abort | Solves errors with no-make-abort. | tealeaves --solve --solve-no-abort |
| --solve-ignored | Solves ignored errors. | tealeaves --solve --solve-ignored |
| --solve-no-idb | Not query from local Idb file, instead use SDOM iDB. If definition still can’t be found, add dummy definition. | tealeaves --solve --solve-no-idb |
| --solve-module=<SOLVE\_MODULE> | Module name to be used(default: SWAdp) | tealeaves --solve --solve-module=  Swadp\_veh |
| --solve-parent=  <SOLVE\_PARENT> | Parent name of module to be used (default: Conf) | tealeaves --solve --solve-parent=Conf |
| --solve-ref-prj=<SOLVE\_REF\_PRJ> | We can pass the reference project root paths separated with ';' just as CLI option. | tealeaves --solve --solve-ref-prj=path/pver1;path/pver2 |
| --solve-csv=FILE\_PATH> | To stub the interfaces using project-specific that user defines | Tealeaves --solve --solve-csv =” path/to/csv” |
| --solve-proc | To solve only missing-process-related errors. | tealeaves --solve --solve-proc |
| --solve-gui | To start solver in GUI mode. | tealeaves --solve --solve-gui |
| --solve-no-cal | To solve but not mapping to calibration labels. | tealeaves --solve --solve-no-cal |
| --solve-add-ehooks | creates 'ehooks\_config.ini' in output folder | tealeaves --solve --solve-add-ehooks |
| --solve-ignore-import-defs | To solve but ignores SW-VARIABLE definitions from importing modules. | tealeaves --solve --solve-ignore-import-defs |
| --solve-multipledefine-prio-user | To solve but add additional user specific owner priority on multiple definitions SDOM\_CLASS:SDOM\_NAME;... | tealeaves --solve --solve-multipledefine-prio-user=”BC;FC” |
| --solve-frontplanning =<FILE-PATH> | Input report file from FrontPlanningTool. | tealeaves --solve --solve-frontplaning =<FILE-PATH> |
| --idb-unc-filepaths=<IDB-FILE-PATH> | Configures default idbfilepaths, seperated with ';' | tealeaves –solve --idb-unc-filepaths= path/to/idbfile.idb |
| --cal-assign | Choose how the stubbed CALPRM will be assigned, either “0” or “\_C” only.  (default is "\_C") | tealeaves --solve --cal-assign=”0” |
| --detailed-solver-json | Show complete detailed of stubbed element in content of solver\_changed.json | tealeaves --solve --detailed-solver-json |
| -r <PVER-PATH>  or  --root <PVER-PATH> | N/A | Provide path for PVER | tealeaves -r d:/path/to/pver --solve  or  tealeaves -root d:/path/to/pver --solve |
| -w  or  --wait | N/A | Wait for key pressed before exit | tealeaves – w  or  tealeaves – wait |
| -s  or  --silent | N/A | Silent mode | tealeaves – s  or  tealeaves --silent |
| --output-path <OUTPUT-PATH> | N/A | Output path | tealeaves --output-path d:/path/to/output\_folder |
| --patch | N/A | If pre\_bct.bat/pre\_mdgb.bat exists, execution will be requested | tealeaves --patch |
| --patch-force | N/A | If pre\_bct.bat/pre\_mdgb.bat exists, execution will be started | tealeaves --patch-force |
| --patch-ignore | N/A | Ignores 'errorlevel' got from patch file execution | tealeaves --patch-ignore |
| --from | N/A | Consider errors from CORE|DATA step on | tealeaves --from=”CORE” |
| --to | N/A | Consider errors to CORE|DATA|BUILD | tealeaves --to=”DATA” |
| --resolve | N/A | Compare FC with different systemconstant. | tealeaves --resolve |
| --resolve-output <OUTPUT\_PATH> | N/A | Change output path for compare output | tealeaves --resolve-output <OUTPUT\_PATH> |
| --ignore-swb | N/A | Ignores unsupported SWB Vx.x.x | tealeaves --ignore-swb |
| --forward-ignored-errors | N/A | Adds ignored errors to solve tasks | tealeaves --forward-ignored-errors |
| --forward-abort-only | N/A | Disable forwarding errors with no make abort to solve tasks | tealeaves --forward-abort-only |
| --query-help | N/A | iDB tools: List examples and syntax descriptions | tealeaves --query-help |
| --query-idb=<QUERY\_IDB> | N/A | iDB tools: Query labels from iDB and add definition to newly created idb\_tools\_pavast.xml | tealeaves –query-idb=”SC:CAMSFT\_NOMINOVLMIS  F\_SC” |
| --query-idb-file=<QUERY\_IDB\_FILE> | N/A | Used to provide local idb files as reference for iDB query. | tealeaves --query-Idb-file=path/to/idb |
| --dynamic-output | N/A | Forces dynamic output. Uses multithreading | tealeaves --dynamic-output |
| -c <MAX\_CORE>  or  --max-core=< MAX\_CORE> | N/A | Define maximum number of cores which will be used (Default: all available cores) | tealeaves --max-core=7  or  tealeaves -c 7 |
| --log | N/A | Store internal logging | tealeaves --log |
| --analyse-dysched-rules | N/A | Analyses dysched rules | tealeaves --analyse-dysched-rules |
| --extract-ecuc | N/A | Extracts arxml ecuc-values files from conf ar structure | tealeaves --extract-ecuc=”” |
| --dysched-tasks | N/A | Tasks which shell be analysed (regex)  (default is "Ini|\_(5|10|20|100|200)ms|DSM|SyncS[01]") | tealeaves --dysched-tasks=”Ini|SyncS” |
| --create-db | N/A | Generate pickled data for analysis with external Python modules (using protocol 4) | tealeaves --create-db |
| --default-task | N/A | Set default mapping task for stubbed procs (default is  "10MS") | tealeaves --default-task=”10MS” |

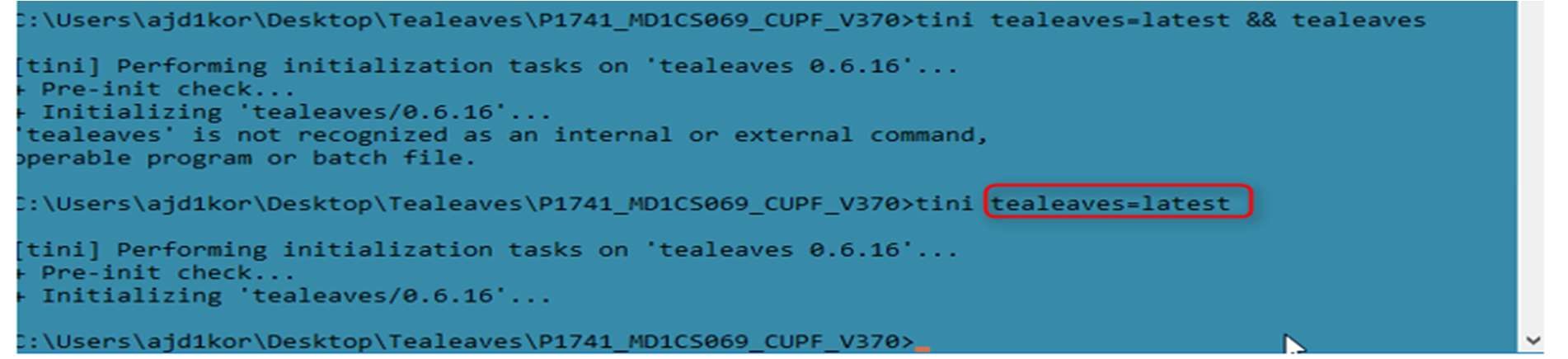
# Output files explaination

Following output files will be stored in the subfolder "\_tealeaves" of the scanned PVER:

|  |  |  |
| --- | --- | --- |
| **Output File** | **Description** | **Corresponding CLI command** |
| \*\_solved.xml | (semi-)automatic corrected files.  Currently supports:   * Adding missing MSR elements * Removing multiple defined MSR (owned) elements * Adding missing required processes in scheduling configuration | tealeaves --solve |
| solver\_replace.bat | Bat script to replace all \*\_solved.xml file into the original file. | tealeaves --solve |
| solver\_restore.bat | Bat script to restore original files | tealeaves --solve |
| tealeaves.idb | Create an iDB file of target project. Supported tables are:   * Since version 0.4.4:   + Label\_Properties   + Label\_Structures   + Lable\_Types   + Lable\_Scopes   + Labels * Since version 0.5.0:   + Functions   + Interfaces   + Methods   + Calls   + Accesses   + Access\_Types   + Message\_Uses | tealeaves --create-idb |
| ignore\_template.csv | .csv file contains all found errors, sorted.  The content of this file can be used for ignoring unwanted error listing. In order to do that, you have to:   * Rename this file to "tealeaves\_ignore.csv" * This does not need to be registered in LWS * "tealeaves\_ignore.csv" has to be located somewhere in the PVER folder   Example:  *Figure 10 Example .csv ignore template* | tealeaves |
| solve\_tasks.xlsx | Excel report file with all detected errors (inherited errors excluded) | tealeaves |
| solve\_tasks.json | json-type file usable for tool-based/automated correction | tealeaves |
| Solver\_changed.xlsx | Excel report file with all solved errors | tealeaves --solve |
| Solver\_changed.json | json-type file with all solved errors | tealeaves --solve |
| conf\_ar.xml, conf\_ar\_resolved.xml, | Merged and resolved configuration structure of parsed AR EcuCValues | tealeaves --conf-out="" |
| conf\_ar\_rules.xml | Can be used for analyzing PVER or creating new test cases | tealeaves –conf-out=”” |
| conf.xml, conf\_doc.xml | Merged and Resolved configuration structure of parsed CONFDATA files.  Can be used for analyzing PVER or creating new test cases | tealeaves --conf-out="" |
| os\_rules.xml | Merged DySched rules | tealeaves --conf-out="" |
| os\_rules\_resolved.xml | Check for erroneous Systemconditions/missing Systemconstant | tealeaves --conf-out=”<XPATH>” |
| idb\_tools\_pavast.xml | Pseudo MSR PaVaSt file containing all found elements in iDB. Supported elements:   * Variables * Systemconstant * CALPRM * Compu-Methods * PaCoIn (if available)   Use option "--query-help" for more detailed usage description with examples.  Use option "--query-idb-file=<tealeaves idb file>" to use local iDB file. | tealeaves --queryidb="<label\_name>" |
| \_out | Generate new SCTADATA, after checking RW analysis for each Required Proc in OS\_sched. | tealeaves --analyse-dysched-rules |
| profile | Enable internal profiling containing number of calls made to a file and time taken to parse a file. | tealeaves --profile |
| tealeaves.log | Show summarized detected errors | tealeaves |
| cached\_data.bin | Internal cached resources of Tealeaves | tealeaves |
| sysconst\_defines.txt | Used for modifying or adding new Systemconstant.  Missing Systemconstant will be listed like following (separated by new line):  **MISSING\_SYSCONST\_SC=**  **MY\_NEW\_SYSCONST=**  You can:   * Fill in a decimal value (1:1 computation method is used) and restart Tealeaves * Add an existing Systemconstant to a new line | tealeaves |
| debug.log | Will be created if a tool intern error happened or "--log" option is used. | tealeaves/tealeaves --log |

# Headless mode / swb\_shell Invocation:

Open the swb\_shell in your project folder or any other folder. Invoke Tealeaves using “tini tealeaves latest” along with --help option to find various options supported for the use cases mentioned above.



# Known Restrictions:

* Compiler and Linker errors are not detected.
* Predicted errors may occur since few files are generated during PVER build.
* Prediction error may occur due to severity change in different DAMOS version.

# Help:

Docupedia: <https://inside-docupedia.bosch.com/confluence/pages/viewpage.action?pageId=288638730#teaLeaves-Output>

In case your query is not addressed in the above link, mail to **Tool-Hotline Tealeaves**:

[BanM\_MS\_Tool-Hotline\_Tealeaves@bosch.com](mailto:BanM_MS_Tool-Hotline_Tealeaves@bosch.com)

# Others:

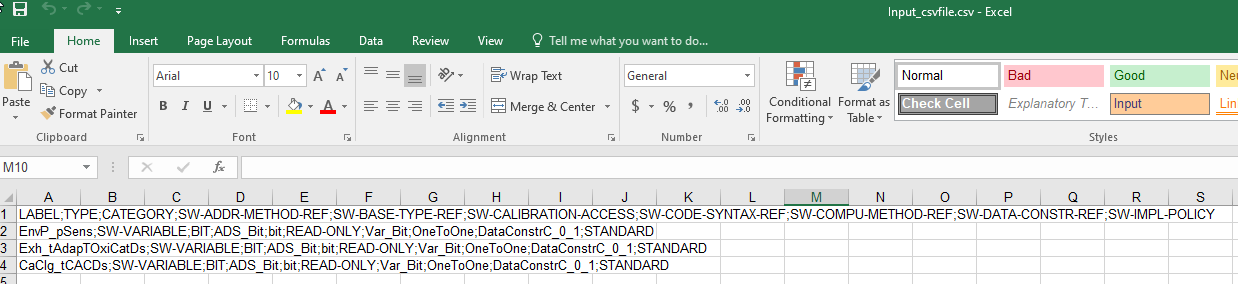


Figure Example .csv input file

Example csv data:

|  |
| --- |
| LABEL;TYPE;CATEGORY;SW-ADDR-METHOD-REF;SW-BASE-TYPE-REF;SW-CALIBRATION-ACCESS;SW-CODE-SYNTAX-REF;SW-COMPU-METHOD-REF;SW-DATA-CONSTR-REF;SW-IMPL-POLICY |
| EnvP\_pSens;SW-VARIABLE;BIT;ADS\_Bit;bit;READ-ONLY;Var\_Bit;OneToOne;DataConstrC\_0\_1;STANDARD |

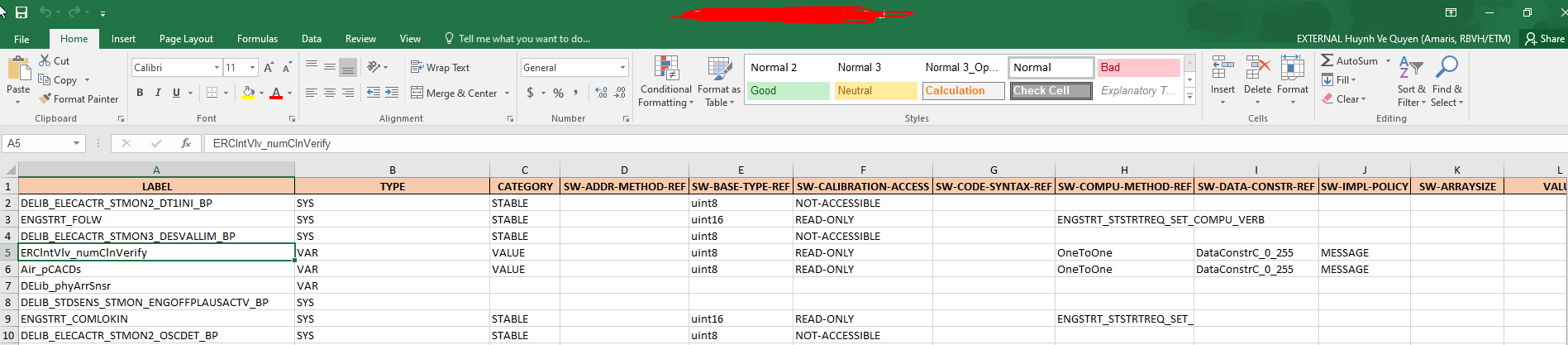


Figure Example AFP .xlsx file

Example AFP data:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **LABEL** | **TYPE** | **CATEGORY** | **SW-ADDR-METHOD-REF** | **SW-BASE-TYPE-REF** | **SW-CALIBRATION-ACCESS** | **SW-CODE-SYNTAX-REF** | **SW-COMPU-METHOD-REF** | **SW-DATA-CONSTR-REF** | **SW-IMPL-POLICY** | **SW-ARRAYSIZE** | **VALUE** |
| DELIB\_ELECACTR\_STMON2\_DT1INI\_BP | SYS | STABLE |  | uint8 | NOT-ACCESSIBLE |  |  |  |  |  |  |
| ENGSTRT\_FOLW | SYS | STABLE |  | uint16 | READ-ONLY |  | ENGSTRT\_STSTRTREQ\_SET\_COMPU\_VERB |  |  |  |  |

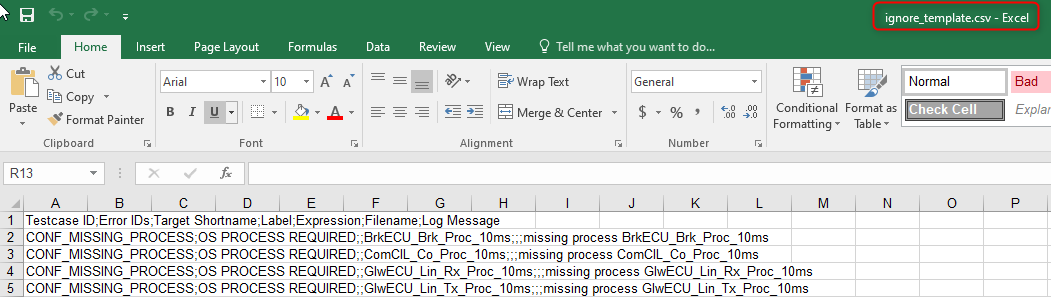


Figure Example .csv ignore template

Example ignore data:

|  |
| --- |
| Testcase ID;Error IDs;Target Shortname;Label;Expression;Filename;Log Message |
| CONF\_MISSING\_PROCESS;OS PROCESS REQUIRED;;BrkECU\_Brk\_Proc\_10ms;;;missing process BrkECU\_Brk\_Proc\_10ms |

What are the various categories available for SC and brief about each with usage? Which category must be used for Variant Type C coding?

System constants can be defined with three categories: FIXED, STABLE, ADJUSTABLE. The following conventions and processes shall be followed.

Table Category types

|  |  |  |
| --- | --- | --- |
| **Category** | **Description** | **Usage** |
| FIXED | The convention is that for fixed system constants the value never changes. If this system constant is used by two or more BCs, then it should be defined and owned to central elements by clearing request in the Architecture Clearing Portal Architecture Clearing Portal. If this system constant is used only in one BC, then it has to be defined and owned inside of this BC. | FIXED system constants shall only contain references to system constants of category FIXED. |
| STABLE | They are defined and owned inside the BC (package local /decentral). Responsible for clearing is the belonging package responsible. It is neither part of central elements nor CONF package. If the value of the system constant is changed this normally requires a new variant of the BC. | STABLE system constants may contain references to system constants of category ADJUSTABLE, STABLE and FIXED. |
| ADJUSTABLE | Used as variation points or "switchable features". All ADJUSTABLE system constants are centrally defined. Element declarations are done in the eASEE Normalization Editor by a clearing request in the Architecture Clearing Portal Architecture Clearing Portal. Value definition is done project specific in eASEE Implementation Editor. | ADJUSTABLE system constants may not contain any references to other system constants, regardless of which category (ADJUSTABLE, STABLE and FIXED). ADJUSTABLE system constants contain always values. (This rule is controlled by the eASEE system constant editor, this means only values are supported for ADJUSTABLE system constants.)  System constants with category ADJUSTABLE should be used for Variant Type C coding. |