ITER IMAS ECOSYSTEM TUTORIAL

IMAS-Validator Guide

A comprehensive hands-on tutorial for validating Interface Data Structures (IDSs) and ensuring physics data quality across the ITER ecosystem.

What is IMAS-Validator?

Core Purpose

IMAS-Validator is a specialized tool designed to evaluate **Interface Data Structures (IDSs)** against both the official IMAS Data Dictionary specification and custom physical or logical validation rules.

Prior to database ingestion, it enforces logical consistency, structural interdependency, and domain-specific rules.

Key Advantages

Validation occurs in two stages: pre-write validation before storage, and post-write validation prior to final database ingestion.

Ensures seamless multi-institutional interoperability between modeling codes and experimental databases.

1. Map & Post-process data

Convert raw simulation or diagnostic data into IMAS IDSs and apply physics post-processing.

2. Validate

Run IMAS-Validator to verify schema completeness and physics constraints.

3. Review Validation Report

Inspect generated validation metrics and resolve warnings/error nodes.

4. IMAS Data Ingest

Write certified data into HDF5/MDS+ database for official user access.

Core Capability Pillars

Schema Validation

Complements compile-time language guarantees by validating runtime data compliance against the IMAS Data Dictionary: required node presence, dimensional/coordinate consistency and field interdependencies.

Physics Constraints

Validates physical constraints: physical magnitude plausibility (order-of-magnitude and unit sanity), valid boundary ranges, non-negativity of physical quantities (e.g., Te, Bt), global conservation laws (e.g., quasi-neutrality $n_e \approx \sum Z_i n_i$), and strict temporal monotonicity.

CLI & Pipeline Ready

Includes an intuitive standalone CLI for batch file verification and a full Python API for integration inside automated scientific pipelines.

Presentation Agenda

01 Installing IMAS-Validator

02 Defining Validation Rules

03 Using the IMAS-Validator

04 Lessons learned from WEST data

01. Installing IMAS-Validator

Requirements & Set-up

Installing the IMAS-Validator

PyPI Package Installation

To get started quickly, you can install the validator directly from : `pypi.org`

```
pip install imas-validator
```

Local Installation from Sources

We recommend using a `venv` . Then, clone the IMAS-Validator repository and run : `pip install`

```
python3 -m venv ./venv
. venv/bin/activate

git clone git@github.com:iterorganization/IMAS-Validator.git
cd IMAS-Validator
pip install --upgrade pip
pip install --upgrade wheel setuptools
pip install .[all]
```

Verify Installation

```
python -c "import imas_validator; print(imas_validator.__version__)"
python -m pytest
```

02. Defining Custom Validation Rules

Grouping, Directory Structure, Decorators, and Assertions

Rule Grouping & Hierarchy

Rule

A set of checks that logically belong together, defined as a Python function and applied to a single target IDS.

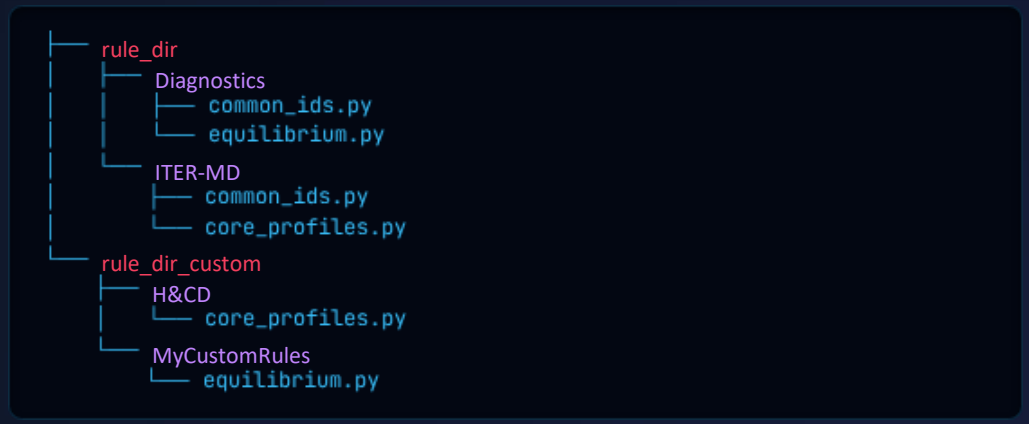
Rule File

Multiple rules collected inside a single `.py` file. Grouped by target IDS (e.g., `equilibrium.py`) or specific functionality.

Rule Set

Folder containing rule files for specific scenarios (e.g., ITER Machine Limits, H&CD workflow).

```python
@validator("gyrokinetics_local")
def validate_gyrokinetics_electron_definition(gk):
  """Validate that there is an electron species in the species AoS."""
  for species in gk.species:
    if species.charge_norm != -1:
      continue
    assert species.mass_norm == 2.724437108e-4
    assert species.temperature_norm == 1.0
    assert species.density_norm == 1.0
    break
  else:
    assert False, "No electron species found"
```



Rule Definition & Syntax

@validator Decorator

Targets specific IDSs and optional occurrences:

```
@validator('summary')
@validator('summary:0')
@validator('summary:0', 'equilibrium:0')
```

Targets all IDSs and all occurrences:

```
@validator("*")
```

Unique Rule Identifier

Constructed as :

<rule_set>/<rule_file>/<rule_name>

Function Body & Assertions

```
@validator('core_profiles')
def validate_electron_temp(cp):
    """Ensure electron temperature is non-negative."""
    for p1d in cp.profiles_1d:
        assert (p1d.electrons.temperature ≥ 0).all(), \
            "Negative electron temperature detected"
```

Docstring describes the test purpose and guides users on fixing failures. Optional error messages clarify complex check expressions when assertions evaluate to `False` .

Important

In contrast to regular Python `assert` statements, the validation rule continues to be evaluated after a failed `assert` . This allows to catch multiple validation failures in a single rule, instead of stopping after the first. It may, however, be surprising to regular Python developers:

Rules continue evaluation after a failed assert

```
@validator("core_profiles")
def validate_profiles_1d(cp):
  assert len(cp.profiles_1d) > 0
  # In regular Python, we don't reach this line when profiles_1d is empty.
  # However, this is a validation rule and we could get an IndexError
  # because evaluation continues even when len(cp.profiles_1d) == 0
  first_profiles = cp.profiles_1d[0]
  ...
```

Cross-Validating Multiple IDSs

Multi-IDS Validation Concept

It is possible to write rules that cross-validate physical quantities defined across multiple **Interface Data Structures (IDSs)** to verify data agreement.

Occurrence Requirement: While specifying the occurrence number in the `@validator` decorator is optional for single IDS validation, it is **mandatory** for multi-IDS validation (e.g., `"summary:0"`).

Typical Use Cases

- Verifying time trace alignment between global summary and core profiles.
- Checking total plasma current (`Ip`) consistency across equilibrium and core IDSs.

```python
@validator("summary:0", "core_profiles:0")
def cross_validate_summary_and_core_profiles(summary,
core_profiles):
    """
    Validate that quantities defined in both
    summary and core_profiles are in agreement.
    """

    assert Approx(summary.time, core_profiles.time)
    assert Approx(
        summary.global_quantities.ip.value,
        core_profiles.global_quantities.ip
    )
```

03. Using the IMAS-Validator

Executing validation via Command Line and Python API

Exploring Rulesets via Command Line

Explore active rules

$ imas_validator **explore**

```
(python-3.11) [LF218007@canismajor ~]$ imas_validator explore
08:50:45 INFO     Parsing data dictionary version 3.42.2 @dd_zip.py:89
Explore Tool
└─ rulesets
   ├─ generic
   │  ├─ Folder for bundled generic IMAS-Validator tests
   │  ├─ generic.py
   │  │  ├─ Generic rules applying to all IDSs
   │  │  ├─ validate_homogeneous_time
   │  │  │  ├─ Applies to IDSs: *
   │  │  │  └─ Validation function for the time mode
   │  │  │     (ids_properties.homogeneous_time)....
   │  │  ├─ validate_increasing_time
   │  │  │  ├─ Applies to IDSs: *
   │  │  │  └─ Validate that all non-empty time vectors are strictly
   │  │  │     increasing....
   │  │  └─ validate_min_max
   │  │     ├─ Applies to IDSs: *
   │  │     └─ Validate that ``*_min`` values are lower than ``*_max``
   │  │        values, and the related...
```

```
   ├─ iter
   │  ├─ Ruleset with data validation rules specific to the ITER tokamak.
   │  │
   │  │  These rules test that key physics values meet with the ones expected
   │  │  in ITER.
   │  │  Please refer to the web page[1].
   │  │
   │  │  [1]
   │  │  https://confluence.iter.org/display/IMP/Required+fields+in+a+dataset
   │  │  +to+be+imported+in+a+scenario+database # noqa: E501
   │  │
   │  ├─ summary.py
   │  │  ├─ ITER-specific validation rules for the ``summary`` IDS.
   │  │  └─ validate_mandatory_values
   │  │     ├─ Applies to IDSs: summary
   │  │     └─ Validate data in IDS/summary against rulesets for ITER
```

```
   └─ scenarios
      ├─ Ruleset with data validation rules specific to the ITER tokamak.
      │
      │  These rules test that some key physics values are provided.
      │  Please refer to the web page[1].
      │
      │  [1]
      │  https://confluence.iter.org/display/IMP/Required+fields+in+a+dataset
      │  +to+be+imported+in+a+scenario+database # noqa: E501
      │
      ├─ edge_transport.py
      │  ├─ Validation rules of ITER scenario database for the
      │  │  ``edge_transport`` IDS.
      │  └─ validate_mandatory_values
      │     ├─ Applies to IDSs: edge_transport
```

```python
@validator("*")
def validate_temperature_positive(ids):
    """Validate that temperature and energy values are positive"""
    for node in Select(ids, "^((?!_error_).)*$", has_value=True):
        if node.metadata.units == "eV" and not (
            "derivative" in str(node.metadata.path)
        ):
            assert node >= 0, "Negative value found for a temperature or energy"
```

```python
"""ITER-specific validation rules for the ``summary`` IDS."""

@validator("summary")
def validate_mandatory_values(ids):
    """Validate data in IDS/summary against rulesets for ITER scenario."""

    # global_quantities
    assert -8.0 <= ids.global_quantities.b0.value <= 0.0
    assert 4.1 <= ids.global_quantities.r0.value <= 8.5
    assert -17000000.0 <= ids.global_quantities.ip.value <= 0.0
    assert 0 <= ids.global_quantities.q_95.value
```

Running Validations via Command Line (CLI)

Core Command & Basic Usage

The simplest way to audit IDS data is using the CLI. The only required positional parameter is the `imas_uri` .

bash

```
$ imas_validator validate  <imas_uri>
```

Add `--verbose` to stream detailed evaluation logs directly to your terminal screen:

```
$ imas_validator validate <imas_uri>  --verbose
```

Custom Rulesets & Directories

Include custom validation rules by passing directory lookup paths and target ruleset folder names:

-e, --extra-rule-dirs Directory containing custom ruleset sub-folders

-r, --ruleset Name of the specific ruleset folder to apply

```
# Execute validation using custom rule folder & ruleset
$ imas_validator validate \
  'imas:hdf5?path=/Imas_public/public/imasdb/west/3/54924/0' \
  -e path/to/my_rule_folder \
  -r my_ruleset
```

Data Ingestion Note: The validator loads all nodes within the target data entry into memory. Depending on the size and slice depth of your data entry, initial loading and rule execution times may vary.

CLI Advanced Features: Filtering & Debugging

Selective Rule Execution Flags

-f, --filter_name `<pattern>`

Filters rules by name pattern: `<ruleset>/<file>:<rule>`

-f, --filter_ids `<ids_name>`

Filters rules matching specific IDS target structure

-b, --no-bundled

Disables loading rulesets bundled with `imas_validator`

-g, --no-generic

Whether or not to use tests for all IDSs

```
# Run core_profiles rules excluding generic tests
$ imas_validator validate <DBENTRY_URI> \
  --filter_ids core_profiles -g
```

Interactive Debugging with `pdb`

When investigating test failures, the validator can automatically drop into a live **Python Debugger (pdb)** console upon hitting an assertion error.

- Grants terminal access to active stack traces and scope variables.
- Allows live inspection of erroneous IDS node values at run-time.

bash (pdb interactive mode)

```
# Enable Python Debugger on failure
$ imas_validator validate <DBENTRY_URI> -d

AssertionError: Negative electron temperature detected

(Pdb) p cp.profiles_1d[0].electrons.temperature
```

Flag Reference: Use `-d` or `--debug` to activate the interactive console mode.

Example using explore command with filtering and 2 rulesets, disabling bundled rulesets:

```
imas_validator explore -f summary -e rule_sets -r rules_sets1 rules_sets2 -b
```

→ Displays all rules targeting ‘summary’ IDS from rules_sets1 and rules_sets2

Validating Rulesets via Python API

ValidateOptions Parameters

- **IDS URL:** Target data entry URI
- **rulesets:** List of rule sets to apply
- **use_bundled_rulesets:** Include default rulesets
- **extra_rule_dirs:** Custom lookup paths
- **apply_generic:** Enable/disable standard DD rules
- **use_pdb:** Drop into debugger on failures
- **rule_filter:** Filter by rule name or IDS target

Logging & Environment Setup

Set logging level `INFO` to for debugging details or default `WARNING` .

```
# Environment variable alternative for path discovery:
export RULESET_PATH=path/to/rulesets:another/path/rulesets_custom
```

```python
import logging
from imas_validator.validate_options import ValidateOptions,
RuleFilter
from imas_validator.validate.validate import validate

# Configure logger level
logger = logging.getLogger('imas_validator')
logger.setLevel(logging.INFO)

imas_uri = "imas:hdf5?path=path/to/data/entry"

validate_options = ValidateOptions(
    rulesets=['ITER-MD', 'MyCustomRules'],
    use_bundled_rulesets=True,
    extra_rule_dirs=[
        'path/to/my/custom/rule/dirs/rulesets',
        'another/path/rulesets_custom'
    ],
    apply_generic=True,
    use_pdb=False,
    rule_filter=RuleFilter(name=['time'], ids=['core_profile']),
)

results = validate(imas_uri=imas_uri,
validate_options=validate_options)
```

Rule Filtering Mechanism

Selective Rule Execution

When executing the validator, you can target a specific subset of validation rules using the `RuleFilter` class.

Filter Options

- **name:** Filters rules by checking if strings are contained in the full rule identifier.
  Identifier: ruleset/file/function_name

- **ids:** Filters rules by target IDS names.
  Target: ('ids_name_1', 'ids_name_2')

Filter Conjunction: Only rules that adhere to **all** supplied conditions (AND logic) will be executed.

Python Usage Examples

Run only rules with "time" in their name:

```
rule_filter = RuleFilter (name = ['time'])
```

Target core_profile IDSs only:

```
rule_filter = RuleFilter (ids = ['core_profile'])
```

Match rules containing both "core" AND "density":

```
rule_filter = RuleFilter (name = ['core', 'density'])
```

Match "temperature" rules for equilibrium IDSs:

```
rule_filter = RuleFilter (name = ['temperature'], ids = ['equilibrium'])
```

04. Lessons learned from WEST

Using IMAS-Validator on WEST shots

Detailed Validation Report & The Campaign-Scale Challenge

WEST Shot #57658 (Validation Failure Log)

1,531 Rule Failures

```
Summary Report :
Tested URI : imas:hdf5?path=/Imas_public/imas_simulation/public/imasdb/west/3/57658/0
Number of tests carried out : 14,983
Number of successful tests : 13,452
Number of failed tests : 1,531

PASSED IDSs:
+ IDS summary occurrence 0
+ IDS summary occurrence 1
+ IDS temporary occurrence 0

FAILED IDSs:
- IDS core_profiles occurrence 0
RULE: generic/core_profiles.py:validate_electroneutrality
MESSAGE: Electron density profile not consistent with ion charge (quasi-neutrality
violation)
TRACEBACK: FrameSummary file .../generic/core_profiles.py, line 43
NODES COUNT: 15 ['profiles_1d[901]/electrons/density', 'profiles_1d[901]/ion[0]/density',
...]
RULE: generic/core_profiles.py:validate_n_i_total_over_n_e
MESSAGE: n_i_total_over_n_e not consistent with ion and electron densities
TRACEBACK: FrameSummary file .../generic/core_profiles.py, line 159
NODES COUNT: 9 ['profiles_1d[901]/electrons/density', ...]
RULE: generic/core_profiles.py:validate_pressure_thermal_electron_core_profiles
MESSAGE: Electron thermal pressure not consistent with density_thermal * temperature
```

Anatomy of a Validation Report

- **Concerned IDSs & Occurrences:** Explicit list of tested IDSs and occurrence index (e.g.,). `core_profiles:0`
- **Violated Rules:** Exact rule file and function name trigger (e.g.,). `generic/core_profiles.py:validate_zeff`
- **Error Trace & Description:** Docstring explanation and traceback line number explaining the cause.
- **Impacted Nodes & Paths:** Exact list of failing leaf/array paths and node counts.

The Campaign-Scale Challenge

The WEST campaign C7 dataset comprises 1,428 pulses.

Manually exploring thousands of verbose, multi-page text logs across an entire campaign is unfeasible.

To detect systematic data quality trends and recurring physics anomalies, these individual report outputs must be aggregated and **synthesized** into a global database view.

IMAS-Validation-Manager: Batch Automation Suite

Purpose & Repository Setup

Designed to scale `imas_validator` across entire experimental campaigns by automating batch execution, retry mechanics, and multi-shot report aggregation.

```
# Clone the management suite repository
$ git clone git@github.com:fleuryl-ai/IMAS-Validation-Manager.git
```

> Integrates directly with `IMAS-Validator`.

Workflow Script Modules

1. imas_validator_executer.py — Batch Engine

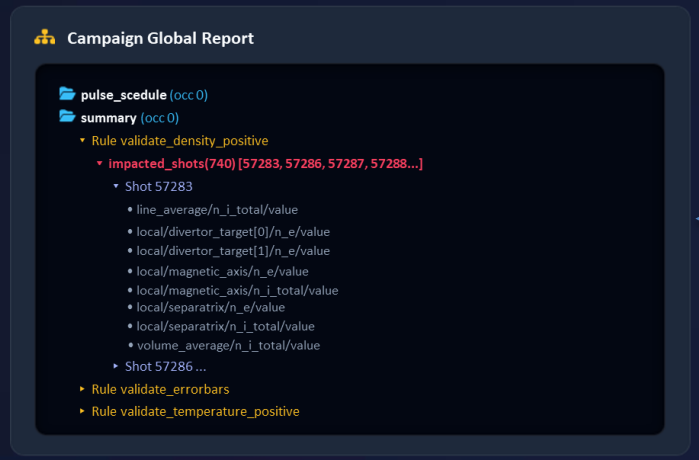
Executes validations across shot ranges (e.g., WEST C7 pulses) and handles automatic retry for interrupted shot runs.

2. imas_global_validator.py — Aggregator

Synthesizes thousands of single-shot validation logs into a unified global JSON database report (`my_global_report_v3.json`).

3. report_viewer.py — GUI Explorer

Interactive UI tree viewer for exploring global campaign anomalies down to individual data dictionary leaf nodes.



WEST Campaign C7 Simulation: Rule Refinement Impact

Comparative impact of **Initial Rules (with False Positives)** vs. **Improved Rules (NaN-Aware)** on the **WEST C7 Simulation Database**.

IDS NODE & VALIDATION RULE	⚠ INITIAL RULES (UNMODIFIED)	✓ IMPROVED RULES (NAN-AWARE)
core_profiles (occ 0)		
validate_electroneutrality	743 shots (58%)	154 shots (12% true failure)
validate_errorbars	364 shots (28% false positive)	0 shots (Passed)
validate_temperature_pos & density_pos	4–143 shots (0.3–11%)	0 shots (Passed)
validate_zeff / n_i_total / pressure_thermal	139–146 shots (~11%)	147–154 shots (~12%)
equilibrium (occ 0)		
validate_errorbars	743 shots (58% false positive)	0 shots (Passed)
validate_sign & validate_min_max	706–743 shots (55–58%)	769–817 shots (62–66%)
summary (occ 0 & 1)		
validate_density_pos / errorbars / temp_pos	667–743 shots (52–58% false positive)	0 shots (Passed)
validate_source	occ 0: 76, occ 1: 69 (~6%)	occ 0: 77, occ 1: 71 (~6%)
core_sources & pulse_schedule		
core_sources/z_ion & pulse_schedule/time	741 shots (57%)	815 shots (66%)

False Positive Elimination

Proper **NaN-aware handling** eliminated 100% of false errorbar and positivity failures across **summary** and **equilibrium**.

True Electroneutrality Rate

Electroneutrality failures dropped from **58% down to 12% (154 shots)**, isolating genuine physical profile inconsistencies.

Persistent Systematic Issues

Real systematic anomalies in **core_sources** (z_ion) and **pulse_schedule** time ordering remain consistently detected (~66%).

Validation Rules Status & Refinement Progress

Overview of validation rule updates, NaN-handling corrections, pending physics decisions, and observed bugs.

VALIDATION RULE	CURRENT STATUS	DETAILS / OBSERVATIONS
Zeff	Patched	Updated to account for 1D profiles of Z and Z^2.
Electroneutrality	Patched	Updated to account for 1D profiles of Z.
z_ion	Patched	No reference to `element.atoms_n` was found in WEST processing codes. Correction required.
pressure_thermal_electron_core_profiles	Pending decision	Charge constant question: Should we adopt the METIS value ($1.602176462 \times 10^{-19}$)?
validate_n_i_total_over_n_e	Patch needed	Bug observed; correction required.
Positive temperature	Patched rule	Rule updated to ignore `NaN` values (eliminates false positives).
Positive density	Patched rule	Rule updated to ignore `NaN` values (eliminates false positives).
min_max	Partially corrected	Initial rule corrected (sets value to 0 if below `UNDERFLOW_THRESHOLD` = 1.0×10^{-100}). Outliers persist for `b_min` and `b_max`.
validate_increasing_time	Failed	Rule not always satisfied: 529 pulses affected for `pulse_schedule` and 3 pulses for `summary` (occ 1).
validate_errors_bars	Patched rule	No longer generating errors. Rule now ignores `NaN` values.

Equilibrium IDS Sign Validation

1. SETUP & PRE-PROCESSING

I_p & B_0 Sign Extraction

Time-slice loop; linear interpolation (`np.interp`) aligns B_0 to time slices.

Poloidal Field Orientation Convention relative to Flux Gradient (σ_{Bp})

Checks `version_put` : $\sigma_{Bp} = +1$ for IMAS v3.x, and $\sigma_{Bp} = -1$ for higher versions.

2. PHYSICAL CONSISTENCY RULES

Diamagnetic Function

$$\text{sign}(f) = \text{sign}(B_0)$$

Matches vacuum toroidal magnetic field sign across 1D profile.

Toroidal Flux Profile

$$\Phi_0 \approx 0 \;\&\; \text{sign}(\Phi_{1:}) = \text{sign}(B_0)$$

Axis flux is zero (`atol=1e-2`); remaining profile follows B_0.

Safety Factor

$$\text{sign}(q_{95}) = \text{sign}(I_p) \cdot \text{sign}(B_0)$$

Product consistency between plasma current and toroidal field.

Kinetic Pressure

$$\text{sign}(P) \geq 0$$

Plasma pressure profile must be strictly non-negative.

Poloidal Flux Gradient

$$\text{sign}(\Psi_{edge} - \Psi_{axis}) = \text{sign}(I_p) \cdot \sigma_{Bp}$$

Evaluated when $|\Delta\Psi| > 10^{-6}$ to prevent non-physical cases.

Questions?

Thank you for your attention.

IMAS-Validator

Repository: github.com/iterorganization/IMAS-Validator
Documentation: imas-validator.readthedocs.io
Training: imas-validator.readthedocs.io/en/latest/courses/basic_user_training.html

IMAS-Validation Manager

https://github.com/fleuryl-ai/IMAS-Validation-Manager
A management suite leveraging IMAS-Validator to automate batch campaigns and visualize results.