# ATIP Documentation

***Release 1.0***

**Tobyn Nicholls**

**Jul 28, 2021**

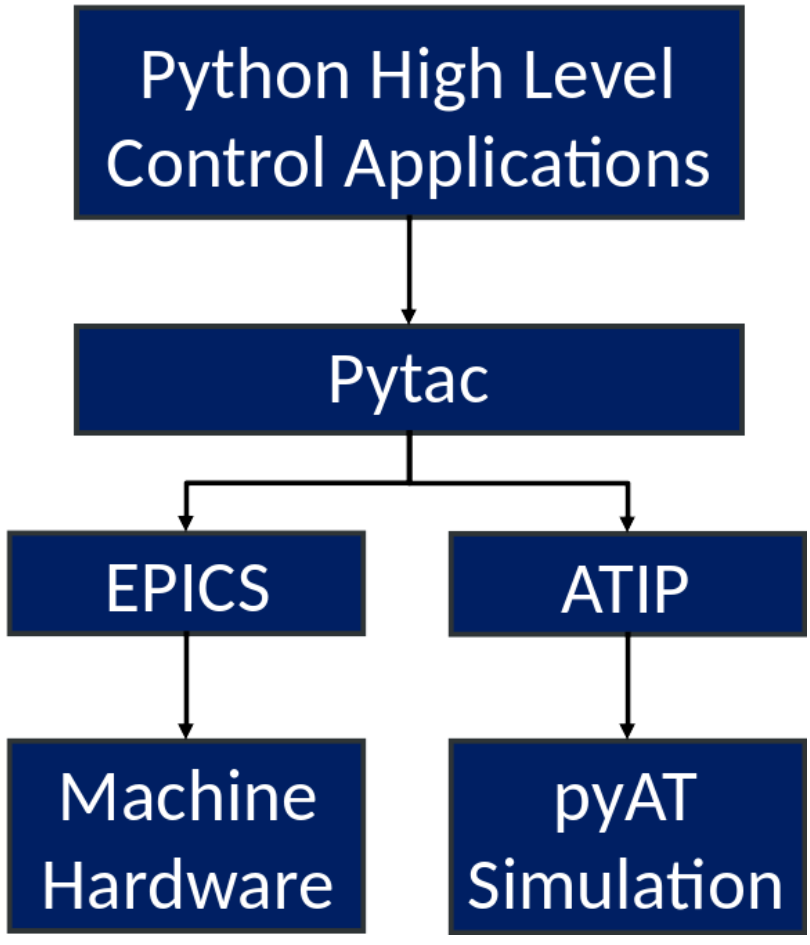
# Contents

ATIP is an addition to Pytac, a framework for controlling particle accelerators. ATIP adds a simulator to Pytac, which can be used and addressed in the same way as a real accelerator. This enables the easy offline testing of high level accelerator controls applications.

ATIP is hosted on Github here.

The python implementation of Accelerator Toolbox (pyAT) is used for the simulation.

**How ATIP fits into the combined control structure.**




ATIP allows an AT lattice to be fitted into the simulation data source of a Pytac lattice. This integrated lattice acts like a normal Pytac lattice, and enables the AT simulator to react and respond to changes as the real accelerator would.

ATIP also makes use of a Cothread thread to recalculate and update the stored physics data any time a change is made to the lattice.

ATIP can also be run in a standalone application as a "virtual accelerator", publishing the same control system interface as the live machine. At Diamond Light Source this has been implemented with EPICS, using PythonSoftIOC. This functionality is not documented here but an explanation of how it works and how to use it may be found in the `.rst` files inside ATIP's `virtac` directory.

# CHAPTER 1

# Example

Note that you need an AT lattice that is compatible with Pytac. Some are provided in `atip/rings/`, otherwise try running the Matlab function `atip/rings/create_lattice_matfile.m` with an AT lattice loaded.

```
>>> import pytac
>>> import atip
>>> # Load the DIAD lattice from Pytac.
>>> lat = pytac.load_csv.load('DIAD')
>>> # Load the AT sim into the Pytac lattice.
>>> atip.load_sim.load_from_filepath(lat, 'atip/rings/diad.mat')
>>> # Use the sim by default.
>>> lat.set_default_data_source(pytac.SIM)
>>> # The initial beam position is zero.
>>> lat.get_value('x')
array([0., 0., 0., ..., 0., 0., 0.])
>>> # Get the first horizontal corrector magnet and set its current to 1A.
>>> hcor1 = lat.get_elements('HSTR')[0]
>>> hcor1.set_value('x_kick', 1, units=pytac.ENG)
>>> # Now the x beam position has changed.
>>> lat.get_value('x')
array([0.00240101, 0.00240101, 0.00239875, ..., 0.00240393, 0.00240327,
       0.00240327])
>>>
```

CHAPTER 2

Contents:

## 2.1 API Documentation

ATIP: Accelerator Toolbox Interface for Pytac. See README.rst & INSTALL.rst for more information.

### 2.1.1 atip.load_sim module

Module responsible for handling the loading of simulator data sources.

atip.load_sim.**load**(*pytac_lattice*, *at_lattice*, *callback=None*)
Load simulator data sources onto the lattice and its elements.

**Parameters**

- **pytac_lattice** (*pytac.lattice.Lattice*) – An instance of a Pytac lattice.
- **at_lattice** (*at.lattice_object.Lattice*) – An instance of an Accelerator Toolbox lattice object.
- **callback** (*callable*) – To be called after completion of each round of physics calculations.

**Returns** The same Pytac lattice object, but now with a simulator data source fully loaded onto it.

**Return type** pytac.lattice.Lattice

atip.load_sim.**load_from_filepath**(*pytac_lattice*, *at_lattice_filepath*, *callback=None*)
Load simulator data sources onto the lattice and its elements.

**Parameters**

- **pytac_lattice** (*pytac.lattice.Lattice*) – An instance of a Pytac lattice.
- **at_lattice_filepath** (*str*) – The path to a .mat file from which the Accelerator Toolbox lattice can be loaded.

- **callback** (*callable*) – To be called after completion of each round of physics calculations.

**Returns** The same Pytac lattice object, but now with a simulator data source fully loaded onto it.

**Return type** pytac.lattice.Lattice

### 2.1.2 atip.sim_data_sources module

Module containing the pytac data sources for the AT simulator.

**class** atip.sim_data_sources.**ATElementDataSource**(*at_element*, *index*, *atsim*, *fields=None*)
Bases: pytac.data_source.DataSource

A simulator data source to enable AT elements to be addressed using the standard Pytac syntax.

**Attributes**

**units**
pytac.ENG or pytac.PHYS, pytac.PHYS by default.

**Type** str

---

**Note:** This data source, currently, cannot understand the simulated equivalent of shared devices on the live machine, or multiple devices that address the same field/attribute for that matter.

---

**Parameters**

- **at_element** (*at.elements.Element*) – The AT element corresponding to the Pytac element which this data source is attached to.
- **index** (*int*) – The element's index in the ring, starting from 1.
- **atsim** (ATSimulator) – An instance of an ATSimulator object.
- **fields** (*list, optional*) – The fields found on this element.

**Raises** ValueError – if an unsupported field is passed, i.e. a field not in _field_funcs.keys().

**Methods:**

**add_field**(*field*)
Add a field to this data source. This is normally done automatically when adding a device, however since the simulated data sources do not use devices this method is needed.

**Parameters** **field** (*str*) – The name of a supported field that is not already on this data_source.

**Raises** FieldException – if the specified field is already present or if it is not supported.

**get_fields**()
Get all the fields that are defined for the data source on this element.

**Returns** A list of all the fields that are present on this element.

**Return type** list

**get_value**(*field*, *handle=None*, *throw=True*)
Get the value for a field.

**Parameters**

- **field** (*str*) – The requested field.
- **handle** (*str, optional*) – Handle is not needed and is only here to conform with the structure of the DataSource base class.
- **throw** (*bool, optional*) – If the check for completion of outstanding calculations times out, then: if True, raise a ControlSystemException; if False, log a warning and return the potentially out of date data anyway.

**Returns** The value of the specified field on this data source.

**Return type** float

**Raises**

- `FieldException` – if the specified field does not exist.
- `ControlSystemException` – if the calculation completion check fails, and throw is True.

**set_value**(*field*, *value*, *throw=None*)

Set the value for a field. The field and value go onto the queue of changes on the ATSimulator to be passed to make_change when the queue is emptied.

**Parameters**

- **field** (*str*) – The requested field.
- **value** (*float*) – The value to be set.
- **throw** (*bool, optional*) – Throw is not needed and is only here to conform with the structure of the DataSource base class.

**Raises**

- `HandleException` – if the specified field cannot be set to.
- `FieldException` – if the specified field does not exist.

**class** atip.sim_data_sources.**ATLatticeDataSource**(*atsim*)

Bases: pytac.data_source.DataSource

A simulator data source to allow the physics data of the AT lattice to be addressed using the standard Pytac syntax.

**Attributes**

**units**

pytac.ENG or pytac.PHYS, pytac.PHYS by default.

**Type** str

---

**Note:** Though not currently supported, there are plans to add get_element_values and set_element_values methods to this data source in future.

---

**Parameters** **atsim** (ATSimulator) – An instance of an ATSimulator object.

**Methods:**

**get_fields**()

Get all the fields that are defined for this data source on the Pytac lattice.

**Returns** A list of all the fields that are present on this element.

**Return type** list

**get_value** (*field*, *handle=None*, *throw=True*)
Get the value for a field on the Pytac lattice.

**Parameters**

- **field** (*str*) – The requested field.
- **handle** (*str, optional*) – Handle is not needed and is only here to conform with the structure of the DataSource base class.
- **throw** (*bool, optional*) – If the check for completion of outstanding calculations times out, then: if True, raise a ControlSystemException; if False, log a warning and return the potentially out of date data anyway.

**Returns** The value of the specified field on this data source.

**Return type** float

**Raises**

- `FieldException` – if the specified field does not exist.
- `ControlSystemException` – if the calculation completion check fails, and throw is True.

**set_value** (*field*, *value*, *throw=None*)
Set the value for a field.

**Note:** Currently, a HandleException is always raised.

**Parameters**

- **field** (*str*) – The requested field.
- **value** (*float*) – The value to be set.
- **throw** (*bool, optional*) – Throw is not needed and is only here to conform with the structure of the DataSource base class.

**Raises** `HandleException` – as setting values to Pytac lattice fields is not currently supported.

### 2.1.3 atip.simulator module

Module containing an interface with the AT simulator.

**class** atip.simulator.**ATSimulator** (*at_lattice*, *callback=None*, *emit_calc=True*)
Bases: object

A centralised class which makes use of AT to simulate the physics data for the copy of the AT lattice which it holds. It works as follows, when a change is made to the lattice in Pytac it is added to the queue attribute of this class. When the queue has changes on it a recalculation is triggered, all the changes are applied to the lattice and then the physics data calculated. This ensures that the physics data is up to date.

**Attributes**

**up_to_date**
A flag that indicates if the physics data is up to date with all the changes made to the AT lattice.

**Type** cothread.Event

**Note:** To avoid errors, the physics data must be initially calculated here, during creation, otherwise it could be accidentally referenced before the attributes _emitdata and _lindata exist due to delay between class creation and the end of the first calculation in the thread.

**Parameters**

- **at_lattice** (*at.lattice_object.Lattice*) – An instance of an AT lattice object.
- **callback** (*callable*) – Optional, if passed it is called on completion of each round of physics calculations.
- **emit_calc** (*bool*) – Whether or not to perform the beam envelope based emittance calculations.

**Methods:**

**get_alpha**()
Return the alpha vector at every element in the AT lattice.

**Returns** The alpha vector for each element.

**Return type** numpy.array

**get_at_element**(*index*)
Return the AT element corresponding to the given index.

**Parameters** **index** (*int*) – The index of the AT element to return.

**Returns** The element specified by the given index.

**Return type** at.elements.Element

**get_at_lattice**()
Return a copy of the AT lattice object.

**Returns** A copy of the AT lattice object.

**Return type** at.lattice_object.Lattice

**get_beta**()
Return the beta vector at every element in the AT lattice.

**Returns** The beta vector for each element.

**Return type** numpy.array

**get_chromaticity**(*field=None*)
Return the chromaticity for the AT lattice for the specified plane.

**Parameters** **field** (*str*) – The desired field (x or y) of chromaticity, if None return both chromaticity dimensions.

**Returns** The x or y chromaticity for the AT lattice.

**Return type** float

**Raises** FieldException – if the specified field is not valid for chromaticity.

**get_damping_partition_numbers**()
Return the damping partition numbers for the 3 normal modes.

**Returns** The damping partition numbers of the AT lattice.

**Return type** numpy.array

**get_damping_times**()
Return the damping times for the 3 normal modes. [tx, ty, tz] = (2*E0*T0)/(U0*[Jx, Jy, Jz]) [1] [1] A.Wolski; CERN Accelerator School, Advanced Accelerator Physics Course, Low Emittance Machines, Part 1: Beam Dynamics with Synchrotron Radiation; August 2013; eqn. 68

**Returns** The damping times of the AT lattice.

**Return type** numpy.array

**get_dispersion**(*field=None*)
Return the dispersion at every element in the AT lattice for the specified plane.

**Parameters** **field** (*str*) – The desired field (x, px, y, or py) of dispersion, if None return whole dispersion vector.

**Returns** The eta x, eta prime x, eta y or eta prime y for the AT lattice as an array of floats the length of the AT lattice.

**Return type** numpy.array

**Raises** FieldException – if the specified field is not valid for dispersion.

**get_emittance**(*field=None*)
Return the emittance for the AT lattice for the specified plane.

---

**Note:** The emittance at the entrance of the AT lattice as it is constant throughout the lattice, and so which element's emittance is returned is arbitrary.

---

**Parameters** **field** (*str*) – The desired field (x or y) of emittance, if None return both emittance dimensions.

**Returns** The x or y emittance for the AT lattice.

**Return type** float

**Raises** FieldException – if the specified field is not valid for emittance.

**get_energy**()
Return the energy of the AT lattice. Taken from the AT attribute.

**Returns** The energy of the AT lattice.

**Return type** float

**get_energy_loss**()
Return the energy loss per turn of the AT lattice.

**Returns** The energy loss of the AT lattice.

**Return type** float

**get_energy_spread**()
Return the energy spread for the AT lattice.

**Returns** The energy spread for the AT lattice.

**Return type** float

**get_horizontal_emittance**()
Return the horizontal emittance for the AT lattice calculated from the radiation integrals, as opposed to the beam envelope formalism used by AT's ohmi_envelope function.

**Returns** The horizontal ('x') emittance for the AT lattice.

**Return type** float

**get_linear_dispersion_action**()
Return the Linear Dispersion Action ("curly H") for the AT lattice.

**Returns** Curly H for the AT lattice

**Return type** float

**get_m44**()
Return the 4x4 transfer matrix for every element in the AT lattice.

**Returns** The 4x4 transfer matrix for each element.

**Return type** numpy.array

**get_momentum_compaction**()
Return the linear momentum compaction factor for the AT lattice.

**Returns** The linear momentum compaction factor of the AT lattice.

**Return type** float

**get_mu**()
Return mu at every element in the AT lattice.

**Returns** The mu array for each element.

**Return type** numpy.array

**get_orbit**(*field=None*)
Return the closed orbit at each element in the AT lattice for the specified plane.

**Parameters** **field** (*str*) – The desired field (x, px, y, or py) of closed orbit, if None return whole orbit vector.

**Returns** The x, x phase, y or y phase for the AT lattice as an array of floats the length of the AT lattice.

**Return type** numpy.array

**Raises** FieldException – if the specified field is not valid for orbit.

**get_radiation_integrals**()
Return the 5 Synchrotron Integrals for the AT lattice.

**Returns** The 5 radiation integrals.

**Return type** numpy.array

**get_s**()
Return the s position of every element in the AT lattice

**Returns** The s position of each element.

**Return type** list

**get_total_absolute_bend_angle**()
Return the total absolute bending angle of all the dipoles in the AT lattice.

**Returns** The total absolute bending angle for the AT lattice.

**Return type** float

**get_total_bend_angle**()
Return the total bending angle of all the dipoles in the AT lattice.

**Returns** The total bending angle for the AT lattice.

**Return type** float

**get_tune**(*field=None*)
Return the tune for the AT lattice for the specified plane.

**Note:** A special consideration is made so only the fractional digits of the tune are returned.

**Parameters** **field** (*str*) – The desired field (x or y) of tune, if None return both tune dimensions.

**Returns** The x or y tune for the AT lattice.

**Return type** float

**Raises** FieldException – if the specified field is not valid for tune.

**pause_calculations**()

**queue_set**(*func*, *field*, *value*)
Add a change to the queue, to be applied when the queue is emptied.

**Parameters**

- **func** (*callable*) – The function to be called to apply the change.
- **field** (*str*) – The field to be changed.
- **value** (*float*) – The value to be set.

**toggle_calculations**()
Pause or unpause the physics calculations by setting or clearing the _paused flag. N.B. this does not pause the emptying of the queue.

**trigger_calculation**()

**unpause_calculations**()

**wait_for_calculations**(*timeout=10*)
Wait until the physics calculations have taken account of all changes to the AT lattice, i.e. the physics data is fully up to date.

**Parameters** **timeout** (*float, optional*) – The number of seconds to wait for.

**Returns** False if the timeout elapsed before the calculations concluded, else True.

**Return type** bool

### 2.1.4 atip.utils module

atip.utils.**get_atsim**(*target*)
Get the ATSimulator object being used by a unified Pytac lattice.

**Parameters** **target** (*pytac.lattice.Lattice or ATSimulator*) – An ATSimulator object or a Pytac lattice from which an ATSimulator object can be extracted.

**Returns** The simulator object performing the physics calculations.

**Return type** *ATSimulator*

atip.utils.**get_sim_lattice**(*target*)
Get the AT lattice that the simulator is using.

**Parameters** **target** (*pytac.lattice.Lattice or ATSimulator*) – An ATSimulator object or a Pytac lattice from which an ATSimulator object can be extracted.

**Returns** The corresponding AT lattice used by the simulator.

**Return type** at.lattice.Lattice

atip.utils.**load_at_lattice**(*mode='DIAD'*, ***kwargs*)
Load an AT lattice from a .mat file in the 'rings' directory.

---

**Note:** I add custom attributes 'Index' and 'Class' to each of the elements in the AT lattice as I find them useful for debugging.

---

**Parameters**

- **mode** (*str*) – The lattice operation mode.
- **kwargs** – any keyword arguments are passed to the AT lattice creator.

**Returns** An AT lattice object.

**Return type** at.lattice.Lattice

atip.utils.**loader**(*mode='DIAD'*, *callback=None*)
Load a unified lattice of the specifed mode.

---

**Note:** A unified lattice is a Pytac lattice where the corresponding AT lattice has been loaded into the Pytac lattice's simulator data source by means of ATIP.

---

**Parameters**

- **mode** (*str*) – The lattice operation mode.
- **callback** (*callable*) – Callable to be called after completion of each round of physics calculations in ATSimulator.

**Returns**

**A Pytac lattice object with the simulator data** source loaded.

**Return type** pytac.lattice.Lattice

atip.utils.**preload**(*pytac_lat*)
Load the elements onto an 'elems' object's attributes by family so that groups of elements of the same family can be more easily accessed, e.g. 'elems.bpm' will return a list of all the BPMs in the lattice. As a special case 'elems.all' will return all the elements in the lattice.

**Parameters** **pytac_lat** (*pytac.lattice.Lattice*) – The Pytac lattice object from which to get the elements.

**Returns** The elems object with the elements loaded onto it by family.

**Return type** obj

atip.utils.**preload_at**(*at_lat*)

Load the elements onto an 'elems' object's attributes by type so that groups of elements of the same type (class) can be more easily accessed, e.g. 'elems.dipole' will return a list of all the dipoles in the lattice. As a special case 'elems.all' will return all the elements in the lattice.

**Parameters** **at_lat** (*at.lattice.Lattice*) – The AT lattice object from which to get the elements.

**Returns** The elems object with the elements loaded onto it by type.

**Return type** obj

atip.utils.**toggle_thread**(*target*)

Pause or unpause the ATSimulator calculation thread.

**Parameters** **target** (*pytac.lattice.Lattice or ATSimulator*) – An ATSimulator object or a Pytac lattice from which an ATSimulator object can be extracted.

atip.utils.**trigger_calc**(*target*)

Manually trigger a recalculation of the physics data on the ATSimulator object of the given unified Pytac lattice.

**Parameters** **target** (*pytac.lattice.Lattice or ATSimulator*) – An ATSimulator object or a Pytac lattice from which an ATSimulator object can be extracted.

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## a

# Index

## Q



## S



## T



## U



## W