

---

# **PyEPR Documentation**

***Release 0.6.1***

**Antonio Valentino**

July 21, 2012



# CONTENTS



# LIST OF FIGURES

**HomePage** <http://avalentino.github.com/pyepr>

**Author** Antonio Valentino

**Contact** [antonio.valentino@tiscali.it](mailto:antonio.valentino@tiscali.it)<sup>1</sup>

**Date** 26/04/2012

**Copyright** 2011-2012, Antonio Valentino

**Version** 0.6.1

---

<sup>1</sup>[antonio.valentino@tiscali.it](mailto:antonio.valentino@tiscali.it)

# USER MANUAL

## 1.1 Quick start

PyEPR<sup>1</sup> provides Python<sup>2</sup> bindings for the ENVISAT Product Reader C API (EPR API<sup>3</sup>) for reading satellite data from ENVISAT<sup>4</sup> ESA<sup>5</sup> (European Space Agency) mission.

PyEPR<sup>6</sup>, as well as the EPR API<sup>7</sup> for C, supports ENVISAT<sup>8</sup> MERIS, AATSR Level 1B and Level 2 and also ASAR data products. It provides access to the data either on a geophysical (decoded, ready-to-use pixel samples) or on a raw data layer. The raw data access makes it possible to read any data field contained in a product file.

Full access to the Python EPR API is provided by the `epr` module that have to be imported by the client program e-g- as follows:

```
import epr
```

The following snippet open an ASAR product and dumps the “Main Processing Parameters” record to the standard output:

```
import epr
```

```
product = epr.Product(  
    'ASA_IMP_1PNUPA20060202_062233_000000152044_00435_20529_3110.N1')  
dataset = product.get_dataset('MAIN_PROCESSING_PARAMS_ADS')  
record = dataset.read_record(0)  
print record
```

---

<sup>1</sup><https://github.com/avalentino/pyepr>

<sup>2</sup><http://www.python.org>

<sup>3</sup><https://github.com/bcdev/epr-api>

<sup>4</sup><http://envisat.esa.int>

<sup>5</sup><http://earth.esa.int>

<sup>6</sup><https://github.com/avalentino/pyepr>

<sup>7</sup><https://github.com/bcdev/epr-api>

<sup>8</sup><http://envisat.esa.int>

## 1.2 Requirements

In order to use PyEPR it is needed that the following software are correctly installed and configured:

- [Python2](#)<sup>9</sup>  $\geq 2.6$  or [Python3](#)<sup>10</sup>  $\geq 3.1$
- [numpy](#)<sup>11</sup>  $\geq 1.3.0$
- [EPR API](#)<sup>12</sup>  $\geq 2.2$
- a reasonably updated C compiler <sup>13</sup> (build only)
- [Cython](#)<sup>14</sup>  $\geq 0.13$  <sup>15</sup> (build only)

---

**Note:** In order to build [PyEPR](#)<sup>16</sup> for [Python3](#)<sup>17</sup> it is required [Cython](#)<sup>18</sup>  $\geq 0.15$

---

## 1.3 Download

Official source tarballs can be downloaded from [PyPi](#)<sup>19</sup>:

<http://pypi.python.org/pypi/pyepr>

The source code of the development versions is available on the [GitHub](#)<sup>20</sup> project page

<https://github.com/avalentino/pyepr>

To clone the [git](#)<sup>21</sup> repository the following command can be used:

```
$ git clone https://github.com/avalentino/pyepr.git
```

---

<sup>9</sup><http://www.python.org>

<sup>10</sup><http://www.python.org>

<sup>11</sup><http://www.numpy.org>

<sup>12</sup><https://github.com/bcdev/epr-api>

<sup>13</sup> [PyEPR](#) (<https://github.com/avalentino/pyepr>) has been developed and tested with [gcc](#) (<http://gcc.gnu.org>) 4.

<sup>14</sup><http://cython.org>

<sup>15</sup> The source tarball of official releases also includes the C extension code generated by [cython](#) (<http://cython.org>) so users don't strictly need [cython](#) (<http://cython.org>) to install [PyEPR](#) (<https://github.com/avalentino/pyepr>).

It is only needed to re-generate the C extension code (e.g. if one wants to build a development version of [PyEPR](#) (<https://github.com/avalentino/pyepr>)).

<sup>16</sup><https://github.com/avalentino/pyepr>

<sup>17</sup><http://www.python.org>

<sup>18</sup><http://cython.org>

<sup>19</sup><http://pypi.python.org>

<sup>20</sup><https://github.com>

<sup>21</sup><http://git-scm.com>



## 1.4 Installation

The easier way to install [PyEPR](#)<sup>22</sup> is using tools like [easy\\_install](#)<sup>23</sup>, [pip](#)<sup>24</sup>:

```
$ easy_install pyepr
```

or:

```
$ easy_install -U --prefix=<TARGET DIRECTORY>
```

[PyEPR](#)<sup>25</sup> uses the standard [Python](#)<sup>26</sup> [distutils](#)<sup>27</sup> so it can be installed from sources using the following command:

```
$ python setup.py install
```

For a user specific installation use:

```
$ python setup.py install --user
```

To install [PyEPR](#)<sup>28</sup> in a non-standard path:

```
$ python setup.py install --prefix=<TARGET_PATH>
```

just make sure that `<TARGET_PATH>/lib/pythonX.Y/site-packages` is in the `PYTHONPATH`<sup>29</sup>.

For all installation methods described above it is assumed that the [EPR API](#)<sup>30</sup> C library is properly installed in the system (see the [Requirements](#) section).

It is also possible to use the [EPR API](#)<sup>31</sup> C sources directly to build PyEPR by specifying the `--epr-api-src` option:

```
$ python setup.py install --epr-api-src=../epr-api/src
```

## 1.5 Testing

[PyEPR](#)<sup>32</sup> package comes with a complete test suite but in order to run it the ENVISAT sample product used for testing

---

<sup>22</sup><https://github.com/avalentino/pyepr>

<sup>23</sup><http://pypi.python.org/pypi/setuptools#using-setuptools-and-easyinstall>

<sup>24</sup><http://pypi.python.org/pypi/pip>

<sup>25</sup><https://github.com/avalentino/pyepr>

<sup>26</sup><http://www.python.org>

<sup>27</sup><http://docs.python.org/distutils>

<sup>28</sup><https://github.com/avalentino/pyepr>

<sup>29</sup><http://docs.python.org/using/cmdline.html#envvar-PYTHONPATH>

<sup>30</sup><https://github.com/bcdev/epr-api>

<sup>31</sup><https://github.com/bcdev/epr-api>

<sup>32</sup><https://github.com/avalentino/pyepr>

MER\_LRC\_2PTGMV20000620\_104318\_00000104X000\_00000\_00000\_0001.N1<sup>33</sup> have to be downloaded from the ESA<sup>34</sup> website, saved in the `test` directory and decompressed.

On GNU Linux platforms the following shell commands can be used:

```
$ cd pyepr-0.X/test
$ wget http://earth.esa.int/services/sample_products/meris/LRC/L2/\
MER_LRC_2PTGMV20000620_104318_00000104X000_00000_00000_0001.N1.gz
$ gunzip MER_LRC_2PTGMV20000620_104318_00000104X000_00000_00000_0001.N1.gz
```

After installation the test suite can be run using the following command in the `test` directory:

```
$ python test_all.py
```

## 1.6 Python vs C API

The Python<sup>35</sup> EPR API is fully object oriented. The main structures of the C API have been implemented as objects while C function have been logically grouped and mapped onto object methods.

The entire process of defining an object oriented API for Python<sup>36</sup> has been quite easy and straightforward thanks to the good design of the C API,

Of course there are also some differences that are illustrated in the following sections.

## 1.7 Memory management

Being Python<sup>37</sup> a very high level language users have never to worry about memory allocation/deallocation. They simply have to instantiate objects:

```
product = epr.Product('filename.N1')
```

and use them freely.

Objects are automatically destroyed when there are no more references to them and memory is deallocated automatically.

Even better, each object holds a reference to other objects it depends on so the user never have to worry about identifiers validity or about the correct order structures have to be feed.

For example: the `C EPR_DatasetId` structure has a field (`product_id`) that points to the `product` descriptor `EPR_productId` to which it belongs to.

The reference to the parent product is used, for example, when one wants to read a record using the `epr_read_record` function:

---

<sup>33</sup>[http://earth.esa.int/services/sample\\_products/meris/LRC/L2/MER\\_LRC\\_2PTGMV20000620\\_104318\\_00000104X000\\_00000\\_00000\\_0001.N1.gz](http://earth.esa.int/services/sample_products/meris/LRC/L2/MER_LRC_2PTGMV20000620_104318_00000104X000_00000_00000_0001.N1.gz)

<sup>34</sup><http://earth.esa.int>

<sup>35</sup><http://www.python.org>

<sup>36</sup><http://www.python.org>

<sup>37</sup><http://www.python.org>

```
EPR_SRecord* epr_read_record(EPR_SDatasetId* dataset_id, ...);
```

The function takes a *EPR\_SDatasetId* as a parameter and assumes all fields (including `dataset->product_id`) are valid. It is responsibility of the programmer to keep all structures valid and free them at the right moment and in the correct order.

This is the standard way to go in C but not in [Python](#)<sup>38</sup>.

In [Python](#)<sup>39</sup> all is by far simpler, and the user can get a *dataset* object instance:

```
dataset = product.get_dataset('MAIN_PROCESSING_PARAMS_ADS')
```

and then forget about the *product* instance it depends on. Even if the *product* variable goes out of scope and is no more directly accessible in the program the *dataset* object keeps staying valid since it holds an internal reference to the *product* instance it depends on.

When *record* is destroyed automatically also the parent `epr.Product` object is destroyed (assumed there is no other reference to it).

The entire machinery is completely automatic and transparent to the user.

## 1.8 Arrays

[PyEPR](#)<sup>40</sup> uses [numpy](#)<sup>41</sup> in order to manage efficiently the potentially large amount of data contained in [ENVISAT](#)<sup>42</sup> products.

- `epr.Field.get_elems()` return an 1D array containing elements of the field
- the *Raster.data* property is a 2D array exposes data contained in the `epr.Raster` object in form of `numpy.ndarray`<sup>43</sup>

---

**Note:** `epr.Raster.data` directly exposes `epr.Raster` i.e. shares the same memory buffer with `epr.Raster`:

```
>>> raster.get_pixel(i, j)
5
>>> raster.data[i, j]
5
>>> raster.data[i, j] = 3
>>> raster.get_pixel(i, j)
3
```

---

<sup>38</sup><http://www.python.org>

<sup>39</sup><http://www.python.org>

<sup>40</sup><https://github.com/avalentino/pyepr>

<sup>41</sup><http://www.numpy.org>

<sup>42</sup><http://envisat.esa.int>

<sup>43</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>

- `epr.Band.read_as_array()` is an additional method provided by the [Python](#)<sup>44</sup> EPR API (does not exist any correspondent function in the C API). It is mainly a facility method that allows users to get access to band data without creating an intermediate `epr.Raster` object. It read a slice of data from the `epr.Band` and returns it as a 2D `numpy.ndarray`<sup>45</sup>.

## 1.9 Enumerators

[Python](#)<sup>46</sup> does not have *enumerators* at language level. Enumerations are simply mapped as module constants that have the same name of the C enumerate but are spelled all in capital letters.

For example:

C	Pythn
<code>e_tid_double</code>	<code>E_TID_DOUBLE</code>
<code>e_smod_1OF1</code>	<code>E_SMOD_1OF1</code>
<code>e_smid_log</code>	<code>E_SMID_LOG</code>

## 1.10 Error handling and logging

Currently error handling and logging functions of the EPR C API are not exposed to python.

Internal library logging is completely silenced and errors are converted to [Python](#)<sup>47</sup> exceptions. Where appropriate standard [Python](#)<sup>48</sup> exception types are use in other cases custom exception types (e.g. `epr.EPRError`, `epr.EPRValueError`) are used.

## 1.11 Library initialization

Differently from the C API library initialization is not needed: it is performed internally the first time the `epr` module is imported in [Python](#)<sup>49</sup>.

## 1.12 High level API

[PyEPR](#)<sup>50</sup> provides some utility method that has no correspondent in the C API:

- `epr.Record.fields()`

---

<sup>44</sup><http://www.python.org>

<sup>45</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>

<sup>46</sup><http://www.python.org>

<sup>47</sup><http://www.python.org>

<sup>48</sup><http://www.python.org>

<sup>49</sup><http://www.python.org>

<sup>50</sup><https://github.com/avalentino/pyepr>

- `epr.Record.get_field_names()`
- `epr.Dataset.records()`
- `epr.Product.get_dataset_names()`
- `epr.Product.get_band_names()`
- `epr.Product.datasets()`
- `epr.Product.bands()`

Example:

```
for dataset in product.datasets():
    for record in dataset.records():
        print record
        print
```

Another example:

```
if 'proc_data' in product.band_names():
    band = product.get_band('proc_data')
    print band
```

## 1.13 Special methods

The Python<sup>51</sup> EPR API also implements some *special method*<sup>52</sup> in order to make EPR programming even handy and, in short, *pythonic*<sup>53</sup>.

The `__repr__` methods have been overridden to provide a little more information with respect to the standard implementation.

In some cases `__str__` method have been overridden to output a verbose string representation of the objects and their contents.

If the EPR object has a `print_` method (like e.g. `epr.Record.print_()` and `epr.Field.print_()`) then the string representation of the object will have in the same format used by the `print_` method. So writing:

```
fd.write(str(record))
```

giver the same result of:

```
record.print_(fd)
```

Of course the `epr.Record.print_()` method is more efficient for writing to file.

---

<sup>51</sup><http://www.python.org>

<sup>52</sup><http://docs.python.org/reference/datamodel.html>

<sup>53</sup>[http://www.cafepy.com/article/be\\_pythonic](http://www.cafepy.com/article/be_pythonic)

Also `epr.Dataset` and `epr.Record` classes implement the `__iter__` special method<sup>54</sup> for iterating over records and fields respectively. So it is possible to write code like the following:

```
for record in dataset:
    for index, field in enumerate(record):
        print index, field
```

`epr.DSD` and `epr.Filed` classes implement the `__eq__` and `__ne__` methods for objects comparison:

```
if filed1 == field2:
    print 'field 1 and field2 are equal'
    print field1
else:
    print 'field1:', field1
    print 'field2:', field2
```

`epr.Field` object also implement the `__len__` special method that returns the number of elements in the field:

```
if field.get_type() != epr.E_TID_STRING:
    assert field.get_num_elems() == len(field)
else:
    assert len(field) == len(field.get_elem())
```

---

**Note:** differently from the `epr.Field.get_num_elems()` method `len(field)` return the number of elements if the field type is not `epr.E_TID_STRING`. If the field contains a string then the string length is returned.

---

---

<sup>54</sup><http://docs.python.org/reference/datamodel.html>

# TUTORIALS

## 2.1 Interactive use of PyEPR

In this tutorial it is showed an example of how to use [PyEPR](#)<sup>1</sup> interactively to open, browse and display data of an [ENVISAT](#)<sup>2</sup> [ASAR](#)<sup>3</sup> product.

For the interactive session it is used the [IPython](#)<sup>4</sup> interactive shell an started with the *ipython -pylab* option to enable interactive plotting provided by the [matplotlib](#)<sup>5</sup> package.

The [ASAR](#)<sup>6</sup> product used in this example is a [free sample](#)<sup>7</sup> available at the [ESA](#)<sup>8</sup> web site.

### 2.1.1 epr module and classes

After starting the ipython shell with the following command:

```
$ ipython -pylab
```

one can import the [epr](#) module and start start taking confidence with available classes and functions:

```
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object'. ?object also works, ?? prints more.

Welcome to pylab, a matplotlib-based Python environment.
```

---

<sup>1</sup><https://github.com/avalentino/pyepr>

<sup>2</sup><http://envisat.esa.int>

<sup>3</sup><http://envisat.esa.int/handbooks/asar>

<sup>4</sup><http://ipython.scipy.org/moin>

<sup>5</sup><http://matplotlib.sourceforge.net>

<sup>6</sup><http://envisat.esa.int/handbooks/asar>

<sup>7</sup>[http://earth.esa.int/services/sample\\_products/asar/IMP/ASA\\_IMP\\_1PNUPA20060202\\_062233\\_000000152044\\_00435\\_20529\\_](http://earth.esa.int/services/sample_products/asar/IMP/ASA_IMP_1PNUPA20060202_062233_000000152044_00435_20529_)

<sup>8</sup><http://earth.esa.int>

For more information, type `'help(pylab)'`.

```
In [1]: import epr
```

```
In [2]: epr?
```

```
Base Class:      <type 'module'>
String Form:    <module 'epr' from 'epr.so'>
Namespace:      Interactive
File:           /home/antonio/projects/pyepr/epr.so
Docstring:
    Python bindings for ENVISAT Product Reader C API

    PyEPR_ provides Python_ bindings for the ENVISAT Product Reader C API
    ('EPR API') for reading satellite data from ENVISAT_ ESA_ (European
    Space Agency) mission.

    PyEPR_ is fully object oriented and, as well as the 'EPR API' for C,
    supports ENVISAT_ MERIS, AATSR Level 1B and Level 2 and also ASAR data
    products. It provides access to the data either on a geophysical
    (decoded, ready-to-use pixel samples) or on a raw data layer.
    The raw data access makes it possible to read any data field contained
    in a product file.

    .. _PyEPR: http://avalentino.github.com/pyepr
    .. _Python: http://www.python.org
    .. _'EPR API': https://github.com/bcdev/epr-api
    .. _ENVISAT: http://envisat.esa.int
    .. _ESA: http://earth.esa.int
```

```
In [3]: epr.__version__, epr.EPR_C_API_VERSION
```

```
Out[3]: ('0.7', '2.2')
```

Docstrings are available for almost all classes, methods and functions in the `epr` and they can be displayed using the `help()`<sup>9</sup> `python`<sup>10</sup> command or the `? IPython`<sup>11</sup> shortcut as showed above.

Also `IPython`<sup>12</sup> provides a handy tab completion mechanism to automatically complete commands or to display available functions and classes:

```
In [4]: product = epr. [TAB]
epr.Band                                epr.E_TID_TIME
epr.__builtins__                        epr.E_TID_UCHAR
epr.__class__                           epr.E_TID_UINT
epr._CLib                               epr.E_TID_UNKNOWN
epr._close_api                          epr.E_TID_USHORT
epr.collections                          epr.Field
```

---

<sup>9</sup><http://docs.python.org/library/functions.html#help>

<sup>10</sup><http://www.python.org>

<sup>11</sup><http://ipython.scipy.org/moin>

<sup>12</sup><http://ipython.scipy.org/moin>



<code>epr.create_bitmask_raster</code>	<code>epr.__file__</code>
<code>epr.create_raster</code>	<code>epr.__format__</code>
<code>epr.Dataset</code>	<code>epr.__getattr__</code>
<code>epr.data_type_id_to_str</code>	<code>epr.get_data_type_size</code>
<code>epr.__delattr__</code>	<code>epr.get_sample_model_name</code>
<code>epr.__dict__</code>	<code>epr.get_scaling_method_name</code>
<code>epr.__doc__</code>	<code>epr.__hash__</code>
<code>epr.DSD</code>	<code>epr.__init__</code>
<code>epr.EPR_C_API_VERSION</code>	<code>epr.__name__</code>
<code>epr.EPRError</code>	<code>epr.__new__</code>
<code>epr.EprObject</code>	<code>epr.np</code>
<code>epr.EPRTime</code>	<code>epr.open</code>
<code>epr.EPRValueError</code>	<code>epr.__package__</code>
<code>epr.E_SMID_LIN</code>	<code>epr.Product</code>
<code>epr.E_SMID_LOG</code>	<code>epr.Raster</code>
<code>epr.E_SMID_NON</code>	<code>epr.Record</code>
<code>epr.E_SMOD_1OF1</code>	<code>epr.__reduce__</code>
<code>epr.E_SMOD_1OF2</code>	<code>epr.__reduce_ex__</code>
<code>epr.E_SMOD_2OF2</code>	<code>epr.__repr__</code>
<code>epr.E_SMOD_2TOF</code>	<code>epr.__revision__</code>
<code>epr.E_SMOD_3TOI</code>	<code>epr.__setattr__</code>
<code>epr.E_TID_CHAR</code>	<code>epr.__sizeof__</code>
<code>epr.E_TID_DOUBLE</code>	<code>epr.so</code>
<code>epr.E_TID_FLOAT</code>	<code>epr.__str__</code>
<code>epr.E_TID_INT</code>	<code>epr.__subclasshook__</code>
<code>epr.E_TID_SHORT</code>	<code>epr.sys</code>
<code>epr.E_TID_SPARE</code>	<code>epr.__test__</code>
<code>epr.E_TID_STRING</code>	<code>epr.__version__</code>

## 2.1.2 `epr.Product` navigation

The first thing to do is to use the `epr.open()` function to get an instance of the desired ENVISAT<sup>13</sup> `epr.Product`:

```
In [4]: product = epr.open(\
'ASA_IMP_1PNUPA20060202_062233_000000152044_00435_20529_3110.N1')
```

<code>In [4]: product.</code>	
<code>product.bands</code>	<code>product.get_num_dsds</code>
<code>product.__class__</code>	<code>product.get_scene_height</code>
<code>product.datasets</code>	<code>product.get_scene_width</code>
<code>product.__delattr__</code>	<code>product.get_sph</code>
<code>product.__doc__</code>	<code>product.__hash__</code>
<code>product.file_path</code>	<code>product.id_string</code>
<code>product.__format__</code>	<code>product.__init__</code>
<code>product.__getattr__</code>	<code>product.meris_iodd_version</code>
<code>product.get_band</code>	<code>product.__new__</code>
<code>product.get_band_at</code>	<code>product.read_bitmask_raster</code>

---

<sup>13</sup><http://envisat.esa.int>

product.get_band_names	product.__reduce__
product.get_dataset	product.__reduce_ex__
product.get_dataset_at	product.__repr__
product.get_dataset_names	product.__setattr__
product.get_dsd_at	product.__sizeof__
product.get_mph	product.__str__
product.get_num_bands	product.__subclasshook__
product.get_num_datasets	product.tot_size

```
In [5]: product.tot_size / 1024.**2
```

```
Out [5]: 132.01041889190674
```

```
In [6]: print product
```

```
epr.Product(ASA_IMP_1PNUPA20060202_ ...) 7 datasets, 5 bands
```

```
epr.Dataset(MDS1_SQ_ADS) 1 records
epr.Dataset(MAIN_PROCESSING_PARAMS_ADS) 1 records
epr.Dataset(DOP_CENTROID_COEFFS_ADS) 1 records
epr.Dataset(SR_GR_ADS) 1 records
epr.Dataset(CHIRP_PARAMS_ADS) 1 records
epr.Dataset(GEOLOCATION_GRID_ADS) 11 records
epr.Dataset(MDS1) 8192 records
```

```
epr.Band(slant_range_time) of epr.Product(ASA_IMP_1PNUPA20060202_ ...)
epr.Band(incident_angle) of epr.Product(ASA_IMP_1PNUPA20060202_ ...)
epr.Band(latitude) of epr.Product(ASA_IMP_1PNUPA20060202_ ...)
epr.Band(longitude) of epr.Product(ASA_IMP_1PNUPA20060202_ ...)
epr.Band(proc_data) of epr.Product(ASA_IMP_1PNUPA20060202_ ...)
```

A short summary of product contents can be displayed simply printing the `epr.Product` object as showed above. Being able to display contents of each object it is easy to keep browsing and get all desired information from the product:

```
In [7]: dataset = product.get_dataset('MAIN_PROCESSING_PARAMS_ADS')
```

```
In [8]: dataset
```

```
Out [8]: epr.Dataset(MAIN_PROCESSING_PARAMS_ADS) 1 records
```

```
In [9]: record = dataset[TAB]
```

dataset.__class__	dataset.get_name	dataset.__reduce__
dataset.create_record	dataset.get_num_records	dataset.__reduce_ex__
dataset.__delattr__	dataset.__hash__	dataset.__repr__
dataset.description	dataset.__init__	dataset.__setattr__
dataset.__doc__	dataset.__iter__	dataset.__sizeof__
dataset.__format__	dataset.__new__	dataset.__str__
dataset.__getattr__	dataset.product	dataset.__subclasshook__
dataset.get_dsd	dataset.read_record	
dataset.get_dsd_name	dataset.records	

```
In [9]: record = dataset.read_record(0)
```

```
In [10]: record
Out[10]: <epr.Record object at 0x33570f0> 220 fields
```

```
In [11]: record.get_field_names()[:20]
```

```
Out[11]:
['first_zero_doppler_time',
 'attach_flag',
 'last_zero_doppler_time',
 'work_order_id',
 'time_diff',
 'swath_id',
 'range_spacing',
 'azimuth_spacing',
 'line_time_interval',
 'num_output_lines',
 'num_samples_per_line',
 'data_type',
 'spare_1',
 'data_analysis_flag',
 'ant_elev_corr_flag',
 'chirp_extract_flag',
 'srgr_flag',
 'dop_cen_flag',
 'dop_amb_flag',
 'range_spread_comp_flag']
```

```
In [12]: field = record.get_field('range_spacing')
```

```
In [13]: field.get [TAB]
```

field.get_description	field.get_name	field.get_unit
field.get_elem	field.get_num_elems	
field.get_elems	field.get_type	

```
In [13]: field.get_description()
```

```
Out[13]: 'Range sample spacing'
```

```
In [14]: epr.data_type_id_to_str(field.get_type())
```

```
Out[14]: 'float'
```

```
In [15]: field.get_num_elems()
```

```
Out[15]: 1
```

```
In [16]: field.get_unit()
```

```
Out[16]: 'm'
```

```
In [17]: print field
```

```
range_spacing = 12.500000
```

### 2.1.3 Iterating over epr objects

`epr.Record` objects are also [iterable](http://docs.python.org/glossary.html#term-iterable)<sup>14</sup> so one can write code like the following:

```
In [18]: for field in record:
         if field.get_num_elems() == 4:
             print '%s: %d elements' % (field.get_name(), len(field))

         ....:
nominal_chirp.1.nom_chirp_amp: 4 elements
nominal_chirp.1.nom_chirp_phs: 4 elements
nominal_chirp.2.nom_chirp_amp: 4 elements
nominal_chirp.2.nom_chirp_phs: 4 elements
nominal_chirp.3.nom_chirp_amp: 4 elements
nominal_chirp.3.nom_chirp_phs: 4 elements
nominal_chirp.4.nom_chirp_amp: 4 elements
nominal_chirp.4.nom_chirp_phs: 4 elements
nominal_chirp.5.nom_chirp_amp: 4 elements
nominal_chirp.5.nom_chirp_phs: 4 elements
beam_merge_sl_range: 4 elements
beam_merge_alg_param: 4 elements
```

### 2.1.4 Image data

Dealing with image data is simple as well:

```
In [19]: product.get_band_names()
```

```
Out[19]: ['slant_range_time',
          'incident_angle',
          'latitude',
          'longitude',
          'proc_data']
```

```
In [19]: band = product.get_band('proc_data')
```

```
In [20]: data = band. [TAB]
```

<code>band.bm_expr</code>	<code>band.read_as_array</code>
<code>band.__class__</code>	<code>band.read_raster</code>
<code>band.create_compatible_raster</code>	<code>band.__reduce__</code>
<code>band.data_type</code>	<code>band.__reduce_ex__</code>
<code>band.__delattr__</code>	<code>band.__repr__</code>
<code>band.description</code>	<code>band.sample_model</code>
<code>band.__doc__</code>	<code>band.scaling_factor</code>
<code>band.__format__</code>	<code>band.scaling_method</code>
<code>band.__getattr__</code>	<code>band.scaling_offset</code>
<code>band.get_name</code>	<code>band.__setattr__</code>
<code>band.__hash__</code>	<code>band.__sizeof__</code>
<code>band.__init__</code>	<code>band.spectr_band_index</code>

---

<sup>14</sup><http://docs.python.org/glossary.html#term-iterable>

```
band.lines_mirrored      band.__str__
band.__new__             band.__subclasshook__
band.product             band.unit
band.__pyx_vtable__
```

```
In [20]: data = band.read_as_array(1000, 1000, xoffset=100, yoffset=6500, \
xstep=2, ystep=2)
```

```
In [21]: data
```

```
Out [21]:
array([[ 146.,  153.,  134., ...,  51.,  55.,  72.],
       [ 198.,  163.,  146., ...,  26.,  54.,  57.],
       [ 127.,  205.,  105., ...,  64.,  76.,  61.],
       ...,
       [  64.,   78.,   52., ...,   96., 176., 159.],
       [  66.,   41.,   45., ..., 200., 153., 203.],
       [  64.,   71.,   88., ..., 289., 182., 123.]], dtype=float32)
```

```
In [22]: data.shape
```

```
Out [22]: (500, 500)
```

```
In [23]: imshow(data, cmap=cm.gray, vmin=0, vmax=1000)
```

```
Out [23]: <matplotlib.image.AxesImage object at 0x60dcf10>
```

```
In [24]: title(band.description)
```

```
Out [24]: <matplotlib.text.Text object at 0x67e9950>
```

```
In [25]: colorbar()
```

```
Out [25]: <matplotlib.colorbar.Colorbar instance at 0x6b18cb0>
```

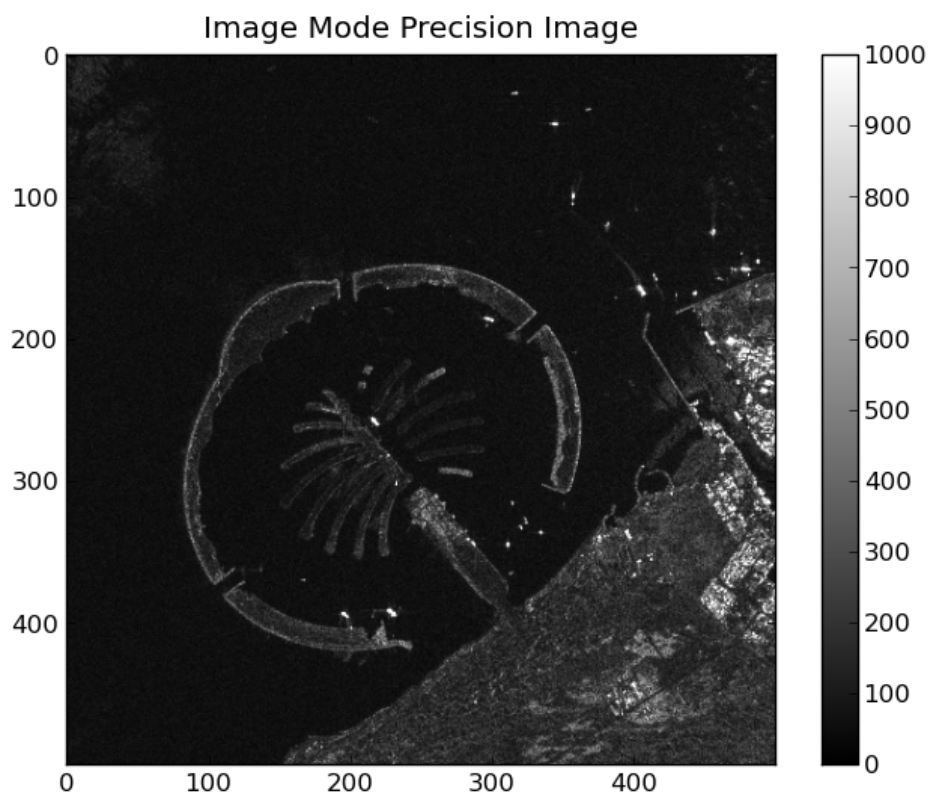


Figure 2.1: Image data read from the “proc\_data” band

## 2.2 Exporting band data

This tutorial shows how to convert [ENVISAT](#)<sup>15</sup> raster information from dataset and generate flat binary rasters using [PyEPR](#)<sup>16</sup>.

The program generates as many raster as the dataset specified in input.

The example code (`examples/write_bands.py`) is a direct translation of the C sample program `write_bands.c`<sup>17</sup> bundled with the EPR API distribution.

The program is invoked as follows:

```
$ python write_bands.py <envisat-product> \
<output directory for the raster file> <dataset name 1> \
[<dataset name 2> ... <dataset name N>]
```

### 2.2.1 Import section

To use the [Python](#)<sup>18</sup> EPR API one have to import `epr` module.

At first import time the underlying C library is opportunely initialized.

```
#!/usr/bin/env python

# This program is a direct translation of the sample program
# "write_bands.c" bundled with the EPR-API distribution.
#
# Source code of the C program is available at:
# https://github.com/bcdev/epr-api/blob/master/src/examples/write_bands.c

import os
import sys
import epr
```

### 2.2.2 The main program

The main program is quite simple (this is just an example).

```
def main(*argv):
    '''A program for converting producing ENVI raster information from
    dataset.

    It generates as many raster as there are dataset entrance parameters.

    Call::
```

---

<sup>15</sup><http://envisat.esa.int>

<sup>16</sup><https://github.com/avalentino/pyepr>

<sup>17</sup>[https://github.com/bcdev/epr-api/blob/master/src/examples/write\\_bands.c](https://github.com/bcdev/epr-api/blob/master/src/examples/write_bands.c)

<sup>18</sup><http://www.python.org>

```
$ write_bands.py <envisat-product>
                    <output directory for the raster file>
                    <dataset name 1>
                    [<dataset name 2> ... <dataset name N>]
```

Example::

```
$ write_bands.py \
MER_RR__1PNPDK20020415_103725_000002702005_00094_00649_1059.N1 \
. latitude

'''

if not argv:
    argv = sys.argv

if len(argv) <= 3:
    print('Usage: write_bands.py <envisat-product> <output-dir> '
          '<dataset-name-1>')
    print('          [<dataset-name-2> ... <dataset-name-N>]')
    print('  where envisat-product is the input filename')
    print('  and output-dir is the output directory')
    print('  and dataset-name-1 is the name of the first band to be '
          'extracted (mandatory)')
    print('  and dataset-name-2 ... dataset-name-N are the names of '
          'further bands to be extracted (optional)')
    print('Example:')
    print('  write_bands MER_RR__2P_TEST.N1 . latitude')
    print
    sys.exit(1)

product_file_path = argv[1]
output_dir_path = argv[2]

# Open the product; an argument is a path to product data file
product = epr.open(product_file_path)

for band_name in argv[3:]:
    write_raw_image(output_dir_path, product, band_name)

if __name__ == '__main__':
    main()
```

It performs some basic command line arguments handling and then open the input product.

```
# Open the product; an argument is a path to product data file
product = epr.open(product_file_path)
```

Finally the core function (`write_raw_image()`) is called on each band specified on the command:

```
for band_name in argv[3:]:
    write_raw_image(output_dir_path, product, band_name)
```



### 2.2.3 The `write_raw_image()` core function

The core function is `write_raw_image()`.

```
def write_raw_image(output_dir, product, band_name):
    '''Generate the ENVI binary pattern image file for an actual DS.

    The first parameter is the output directory path.

    '''

    # Build ENVI file path, DS name specifically
    image_file_path = os.path.join(output_dir, band_name + '.raw')

    band = product.get_band(band_name)
    source_w = product.get_scene_width()
    source_h = product.get_scene_height()
    source_step_x = 1
    source_step_y = 1

    raster = band.create_compatible_raster(source_w, source_h,
                                           source_step_x, source_step_y)

    print 'Reading band "%s"...' % band_name
    raster = band.read_raster(0, 0, raster)

    out_stream = open(image_file_path, 'wb')

    for line in raster.data:
        out_stream.write(line.tostring())
    # or better: raster.data.tofile(out_stream)

    out_stream.close()

    print 'Raw image data successfully written to "%s".' % image_file_path
    print ('C data type is "%s", element size %u byte(s), '
          'raster size is %u x %u pixels.' % (
              epr.data_type_id_to_str(raster.data_type),
              raster.get_elem_size(),
              raster.get_width(),
              raster.get_height()))
```

It generates a flat binary file with data of a single band whose name is specified as input parameter.

First the output file name is computed using the `os`<sup>19</sup> module.

```
# Build ENVI file path, DS name specifically
image_file_path = os.path.join(output_dir, band_name + '.raw')
```

Then the desired band is retrieved using the `epr.Product.get_band()` method and some of its parameters are loaded in to local variables:

```
band = product.get_band(band_name)
source_w = product.get_scene_width()
source_h = product.get_scene_height()
```

---

<sup>19</sup><http://docs.python.org/library/os.html#os>

Band data are accessed by means of a `epr.Raster` object.

**See Also:**

```
epr.Band.read_as_array()
```

The `epr.Band.create_compatible_raster()` is a facility method that allows to instantiate a `epr.Raster` object with a data type compatible with the band data:

```
raster = band.create_compatible_raster(source_w, source_h,
                                       source_step_x, source_step_y)
```

Then data are read using the `epr.Band.read_raster()` method:

```
print 'Reading band "%s"...' % band_name
raster = band.read_raster(0, 0, raster)
```

Then the output file object is created (in binary mode of course)

```
out_stream = open(image_file_path, 'wb')
```

and data are copied to the output file one line at time

```
for line in raster.data:
    out_stream.write(line.tostring())
```

Please note that it has been used `epr.Raster.data` attribute of the `epr.Raster` objects that exposes `epr.Raster` data with the powerful `numpy.ndarray`<sup>20</sup> interface.

---

**Note:** Copying one line at time is not the best way to perform the task in Python<sup>21</sup>. It has been done just to mimic the original C code:

```
out_stream = fopen(image_file_path, "wb");
if (out_stream == NULL) {
    printf("Error: can't open '%s'\n", image_file_path);
    return 3;
}

for (y = 0; y < (uint)raster->raster_height; y++) {
    numwritten = fwrite(epr_get_raster_line_addr(raster, y),
                       raster->elem_size,
                       raster->raster_width,
                       out_stream);

    if (numwritten != raster->raster_width) {
        printf("Error: can't write to %s\n", image_file_path);
        return 4;
    }
}
fclose(out_stream);
```

A by far more *pythonic*<sup>22</sup> solution would be:

---

<sup>20</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>

<sup>21</sup><http://www.python.org>

<sup>22</sup>[http://www.cafepy.com/article/be\\_pythonic](http://www.cafepy.com/article/be_pythonic)

```
raster.data.tofile(out_stream)
```

---

## 2.3 Exporting bitmasks

This tutorial shows how to generate bit masks from ENVISAT<sup>23</sup> flags information as “raw” image using PyEPR<sup>24</sup>.

The example code (`examples/write_bitmask.py`) is a direct translation of the C sample program `write_bitmask.c`<sup>25</sup> bundled with the EPR API distribution.

The program is invoked as follows:

```
$ python write_bitmask.py <envisat-product> <bitmask-expression> \
<output-file>
```

The `examples/write_bitmask.py` code consists in a single function that also includes command line arguments handling:

```
1  #!/usr/bin/env python
2
3  # This program is a direct translation of the sample program
4  # "write_bitmask.c" bundled with the EPR-API distribution.
5  #
6  # Source code of the C program is available at:
7  # https://github.com/bcdev/epr-api/blob/master/src/examples/write_bitmask.c
8
9  '''Generates bit mask from ENVISAT flags information as "raw" image
10 for (e.g.) Photoshop
11
12 Call::
13
14     $ python write_bitmask.py <envisat-product> <bitmask-expression>
15     <output-file>
16
17 Example to call the main function::
18
19     $ python write_bitmask.py MER_RR__2P_TEST.N1 \
20     'l2_flags.LAND and !l2_flags.BRIGHT' my_flags.raw
21
22 '''
23
24 import sys
25 import epr
26
27
28 def main(*argv):
29     if not argv:
30         argv = sys.argv
31
```

---

<sup>23</sup><http://envisat.esa.int>

<sup>24</sup><https://github.com/avalentino/pyepr>

<sup>25</sup>[https://github.com/bcdev/epr-api/blob/master/src/examples/write\\_bitmask.c](https://github.com/bcdev/epr-api/blob/master/src/examples/write_bitmask.c)

```
32     if len(argv) != 4:
33         print('Usage: write_bitmask <envisat-product> <bitmask-expression> '
34               '<output-file>')
35         print('  where envisat-product is the input filename')
36         print('  and bitmask-expression is a string containing the bitmask '
37               'logic')
38         print('  and output-file is the output filename.')
39         print('Example:')
40         print("  MER_RR__2P_TEST.N1 'l2_flags.LAND and !l2_flags.BRIGHT' "
41               "my_flags.raw")
42     print
43     sys.exit(1)
44
45     product_file_path = argv[1]
46     bm_expr = argv[2]
47     image_file_path = argv[3]
48
49     # Open the product; an argument is a path to product data file
50     product = epr.open(product_file_path)
51
52     offset_x = 0
53     offset_y = 0
54     source_width = product.get_scene_width()
55     source_height = product.get_scene_height()
56     source_step_x = 1
57     source_step_y = 1
58
59     bm_raster = epr.create_bitmask_raster(source_width, source_height,
60                                           source_step_x, source_step_y)
61
62     product.read_bitmask_raster(bm_expr, offset_x, offset_y, bm_raster)
63
64     with open(image_file_path, 'wb') as out_stream:
65         bm_raster.data.tofile(out_stream)
66
67     print 'Raw image data successfully written to "%s".' % image_file_path
68     print 'Data type is "byte", size is %d x %d pixels.' % (source_width,
69                                                             source_height)
70
71
72 if __name__ == '__main__':
73     main()
```

In order to use the [Python](http://www.python.org)<sup>26</sup> EPR API the `epr` module is imported:

```
import epr
```

As usual the [ENVISAT](http://www.esa.int)<sup>27</sup> product is opened using the `epr.open()` function that returns an `epr.Product` instance.

```
# Open the product; an argument is a path to product data file
product = epr.open(product_file_path)
```

Scene size parameters are retrieved from the `epr.Product` object using the

---

<sup>26</sup><http://www.python.org>

<sup>27</sup><http://envisat.esa.int>

`epr.Product.get_scene_width()` and `epr.Product.get_scene_height()` methods:

```
source_width = product.get_scene_width()
source_height = product.get_scene_height()
```

The EPR API allows to manage data by means of `epr.Raster` objects, so the function `epr.create_bitmask_raster()`, specific for bitmasks, is used to create a `epr.Raster` instance.

**See Also:**

`epr.create_raster()`

Data are actually read using the `epr.Product.read_bitmask_raster()` method of the `epr.Product` class:

```
product.read_bitmask_raster(bm_expr, offset_x, offset_y, bm_raster)
```

The `epr.Product.read_bitmask_raster()` method receives in input the *bm\_expr* parameter that is set via command line:

```
bm_expr = argv[2]
```

*bm\_expr* is a string that define the logical expression for the definition of the bit-mask. In a bit-mask expression, any number of the flag-names (found in the DDDB) can be composed with “(, )”, “NOT”, “AND”, “OR”.

Valid bit-mask expression are for example:

```
flags.LAND OR flags.CLOUD
```

or:

```
NOT flags.WATER AND flags.TURBID_S
```

Finally data are written to disk as a flat binary file using the `numpy.ndarray.tofile()`<sup>28</sup> method of the `epr.Raster.data` attribute of the `epr.Raster` objects that exposes data via the `numpy.ndarray`<sup>29</sup> interface:

```
with open(image_file_path, 'wb') as out_stream:
    bm_raster.data.tofile(out_stream)
```

## 2.4 NDVI computation

This tutorial shows how to use `PyEPR`<sup>30</sup> to open a `MERIS`<sup>31</sup> L1B product, compute the *Normalized Difference Vegetation Index* (NDVI) and store it into a flat binary file.

---

<sup>28</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.tofile.html#numpy.ndarray.tofile>

<sup>29</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>

<sup>30</sup><https://github.com/avalentino/pyepr>

<sup>31</sup><http://envisat.esa.int/handbooks/meris>

The example code (`examples/write_ndvi.py`) is a direct translation of the C sample program `write_ndvi.c`<sup>32</sup> bundled with the EPR API distribution.

The program is invoked as follows:

```
$ python write_ndvi.py <envisat-oroduct> <output-file>
```

The code have been kept very simple and it consists in a single function (`main()`) that also performs a minimal command line arguments handling.

```
1  #!/usr/bin/env python
2
3  # This program is a direct translation of the sample program
4  # "write_ndvi.c" bundled with the EPR-API distribution.
5  #
6  # Source code of the C program is available at:
7  # https://github.com/bcdev/epr-api/blob/master/src/examples/write_ndvi.c
8
9
10 '''Example for using the epr-api
11 Demonstrates how to open a MERIS L1b product and calculate the NDVI.
12 This example does not demonstrate how to write good and safe code.
13 It is reduced to the essentials for working with the epr-api.
14 Calling sequence::
15 
16     $ python write_ndvi.py <envisat-product> <output-file>
17 for example::
18 
19     $ python write_ndvi.py MER_RR__1P_test.N1 my_ndvi.raw
20 '''
21
22 import sys
23 import struct
24 import logging
25
26 import epr
27
28 def main(*argv):
29     if not argv:
30         argv = sys.argv
31
32     if len(argv) != 3:
33         print('Usage: write_ndvi <envisat-product> <output-file>')
34         print('  where envisat-product is the input filename')
35         print('  and output-file is the output filename')
36         print('Example: MER_RR__1P_TEST.N1 my_ndvi.raw')
37         print
38         sys.exit(1)
39
40     # Open the product
```

---

<sup>32</sup>[https://github.com/bcdev/epr-api/blob/master/src/examples/write\\_ndvi.c](https://github.com/bcdev/epr-api/blob/master/src/examples/write_ndvi.c)

```
47     product = epr.open(argv[1])
48
49     # The NDVI shall be calculated using bands 6 and 8.
50     band1_name = 'radiance_6'
51     band2_name = 'radiance_10'
52
53     band1 = product.get_band(band1_name)
54     band2 = product.get_band(band2_name)
55
56     # Allocate memory for the rasters
57     width = product.get_scene_width()
58     height = product.get_scene_height()
59     subsampling_x = 1
60     subsampling_y = 1
61     raster1 = band1.create_compatible_raster(width, height,
62                                             subsampling_x, subsampling_y)
63     raster2 = band2.create_compatible_raster(width, height,
64                                             subsampling_x, subsampling_y)
65
66     # Read the radiance into the raster.
67     offset_x = 0
68     offset_y = 0
69
70     logging.info('read "%s" data' % band1_name)
71     band1.read_raster(offset_x, offset_y, raster1)
72
73     logging.info('read "%s" data' % band2_name)
74     band2.read_raster(offset_x, offset_y, raster2)
75
76     # Open the output file
77     logging.info('write ndvi to "%s"' % argv[2])
78     out_stream = open(argv[2], 'wb')
79
80     # Loop over all pixel and calculate the NDVI.
81     #
82     # @NOTE: looping over data matrices is not the best solution.
83     # It is done here just for demonstrative purposes
84     for j in range(height):
85         for i in range(width):
86             rad1 = raster1.get_pixel(i, j)
87             rad2 = raster2.get_pixel(i, j)
88             if (rad1 + rad2) != 0.0:
89                 ndvi = (rad2 - rad1) / (rad2 + rad1)
90             else:
91                 ndvi = -1.0
92             out_stream.write(struct.pack('f', ndvi))
93     logging.info('ndvi was written success')
94
95     out_stream.close()
96
97
98 if __name__ == '__main__':
99     main()
```

The ENVISAT<sup>33</sup> `epr.Product` is opened using the `epr.open()` function.

---

<sup>33</sup><http://envisat.esa.int>

```
product = epr.open(argv[1])
```

The name of the product is in the first argument of the program call. In order to keep the code simple no check is performed to ensure that the product is a valid L1B product.

The NDVI is calculated using bands 6 and 8 (the names of these bands are “radiance\_6” and “radiance\_10”). `epr.Band` objects are retrieved using the `epr.Product.get_band()` method:

```
# The NDVI shall be calculated using bands 6 and 8.
band1_name = 'radiance_6'
band2_name = 'radiance_10'

band1 = product.get_band(band1_name)
band2 = product.get_band(band2_name)
```

`band1` and `band2` are used to read the calibrated radiances into the `epr.Raster` objects that allow to access data matrices with the radiance values.

Before reading data into the `epr.Raster` objects they have to be instantiated specifying their size and data type in order to allow the library to allocate the correct amount of memory.

For sake of simplicity `epr.Raster` object are created with the same size of the whole product (with no sub-sampling) using the `epr.Band.create_compatible_raster()` method of the `epr.Band` class:

```
# Allocate memory for the rasters
width = product.get_scene_width()
height = product.get_scene_height()
subsampling_x = 1
subsampling_y = 1
raster1 = band1.create_compatible_raster(width, height,
                                         subsampling_x, subsampling_y)
raster2 = band2.create_compatible_raster(width, height,
                                         subsampling_x, subsampling_y)
```

Then data are actually loaded into memory using the `epr.Band.read_raster()` method. Since `epr.Raster` objects have been defined to match the whole product, offset parameters are set to zero (data are read starting from specified offset):

```
# Read the radiance into the raster.
offset_x = 0
offset_y = 0

logging.info('read "%s" data' % band1_name)
band1.read_raster(offset_x, offset_y, raster1)

logging.info('read "%s" data' % band2_name)
band2.read_raster(offset_x, offset_y, raster2)
```

---

**Note:** In this simplified example it is assumed that there is enough system memory to hold the two `epr.Raster` objects.

---

After opening (in binary mode) the stream for the output



```
# Open the output file
logging.info('write ndvi to "%s"' % argv[2])
out_stream = open(argv[2], 'wb')
```

the program simply loops over all pixel and calculate the NDVI with the following formula:

$$NDVI = \frac{radiance_{10} - radiance_8}{radiance_{10} + radiance_8}$$

```
# Loop over all pixel and calculate the NDVI.
#
# @NOTE: looping over data matrices is not the best solution.
# It is done here just for demonstrative purposes
for j in range(height):
    for i in range(width):
        rad1 = raster1.get_pixel(i, j)
        rad2 = raster2.get_pixel(i, j)
        if (rad1 + rad2) != 0.0:
            ndvi = (rad2 - rad1) / (rad2 + rad1)
        else:
            ndvi = -1.0
        out_stream.write(struct.pack('f', ndvi))
logging.info('ndvi was written success')
```

This part of the code tries to mimic closely the original C code (`write_ndvi.c`<sup>34</sup>)

```
out_stream = fopen(argv[2], "wb");
for (j = 0; j < height; j++) {
    for (i = 0; i < width; i++) {
        rad1 = epr_get_pixel_as_float(raster1, i, j);
        rad2 = epr_get_pixel_as_float(raster2, i, j);
        if ((rad1 + rad2) != 0.0) {
            ndvi = (rad2 - rad1) / (rad2 + rad1);
        } else {
            ndvi = -1.0;
        }
        status = fwrite( & ndvi, sizeof(float), 1, out_stream);
    }
}
epr_log_message(e_log_info, "ndvi was written success");
```

and uses the `epr.Raster.get_pixel()` method to access pixel values and perform computation.

The Python<sup>35</sup> `struct.pack()`<sup>36</sup> function together with `file.write()`<sup>37</sup> is used to write the NDVI of the pixel n the file in binary format.

<sup>34</sup>[https://github.com/bcdev/epr-api/blob/master/src/examples/write\\_ndvi.c](https://github.com/bcdev/epr-api/blob/master/src/examples/write_ndvi.c)

<sup>35</sup><http://www.python.org>

<sup>36</sup><http://docs.python.org/library/struct.html#struct.pack>

<sup>37</sup><http://docs.python.org/library/stdtypes.html#file.write>

```
out_stream.write(struct.pack('f', ndvi))
```

---

**Note:** The entire solution is quite not [pythonic](#)<sup>38</sup>. As an alternative implementation it could be used the `numpy.ndarray`<sup>39</sup> interface of `epr.Raster` objects available via the `epr.Raster.data` property. The NDVI index is computed on all pixels altogether using vectorized expressions:

```
# Initialize the entire matrix to -1
ndvi = numpy.zeros((height, width), 'float32') - 1

aux = raster2.data + raster1.data

# indexes of pixel with non null denominator
idx = numpy.where(aux != 0)

# actual NDVI computation
ndvi[idx] = (raster2.data[idx] + raster1.data[idx]) / aux[idx]
```

Finally data can be saved to file simply using the `numpy.ndarray.tofile()`<sup>40</sup> method:

```
ndvi.tofile(out_stream)
```

---

---

<sup>38</sup>[http://www.cafepy.com/article/be\\_pythonic](http://www.cafepy.com/article/be_pythonic)

<sup>39</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>

<sup>40</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.tofile.html#numpy.ndarray.tofile>

## API REFERENCE

PyEPR<sup>1</sup> provides Python<sup>2</sup> bindings for the ENVISAT Product Reader C API (EPR API<sup>3</sup>) for reading satellite data from ENVISAT<sup>4</sup> ESA<sup>5</sup> (European Space Agency) mission.

PyEPR<sup>6</sup> is fully object oriented and, as well as the EPR API<sup>7</sup> for C, supports ENVISAT<sup>8</sup> MERIS, AATSR Level 1B and Level 2 and also ASAR data products. It provides access to the data either on a geophysical (decoded, ready-to-use pixel samples) or on a raw data layer. The raw data access makes it possible to read any data field contained in a product file.

### 3.1 Classes

#### 3.1.1 Product

**class** `epr.Product`  
ENVISAT product

The Product class provides methods and properties to get information about an ENVISAT product file.

**See Also:**

`open()`

**file\_path**  
The file's path including the file name

**id\_string**  
The product identifier string obtained from the MPH parameter 'PRODUCT'

---

<sup>1</sup><https://github.com/avalentino/pyepr>

<sup>2</sup><http://www.python.org>

<sup>3</sup><https://github.com/bcdev/epr-api>

<sup>4</sup><http://envisat.esa.int>

<sup>5</sup><http://earth.esa.int>

<sup>6</sup><https://github.com/avalentino/pyepr>

<sup>7</sup><https://github.com/bcdev/epr-api>

<sup>8</sup><http://envisat.esa.int>

The first 10 characters of this string identify the product type, e.g. “MER\_1P\_\_FR” for a MERIS Level 1b full resolution product. The rest of the string decodes product instance properties.

**meris\_iodd\_version**

For MERIS L1b and RR and FR to provide backward compatibility

**tot\_size**

The total size in bytes of the product file

**get\_band** (*name*)

Gets the band corresponding to the specified name.

**Parameters** *name* – the name of the band

**Returns** the requested `Band` instance, or raises a `EPRValueError` if not found

**get\_band\_at** (*index*)

Gets the band at the specified position within the product

**Parameters** *index* – the index identifying the position of the band, starting with 0, must not be negative

**Returns** the requested `Band` instance, or raises a `EPRValueError` if not found

**get\_dataset** (*name*)

Gets the dataset corresponding to the specified dataset name

**Parameters** *name* – the dataset name

**Returns** the requested `Dataset` instance

**get\_dataset\_at** (*index*)

Gets the dataset at the specified position within the product

**Parameters** *index* – the index identifying the position of the dataset, starting with 0, must not be negative

**Returns** the requested `Dataset`

**get\_dsd\_at** (*index*)

Gets the `DSD` at the specified position

Gets the `DSD` (dataset descriptor) at the specified position within the product.

**Parameters** *index* – the index identifying the position of the `DSD`, starting with 0, must not be negative

**Returns** the requested `DSD` instance

**get\_num\_bands** ()

Gets the number of all bands contained in a product

**get\_num\_datasets** ()

Gets the number of all datasets contained in a product

**get\_num\_dsds()**

Gets the number of all [DSDs](#) (dataset descriptors) contained in the product

**get\_scene\_height()**

Gets the product's scene height in pixels

**get\_scene\_width()**

Gets the product's scene width in pixels

**get\_mph()**

The [Record](#) representing the main product header (MPH)

**get\_sph()**

The [Record](#) representing the specific product header (SPH)

**get\_dataset\_names()**

Return the list of names of the datasets in the product

---

**Note:** this method has no correspondent in the C API

---

**read\_bitmask\_raster()**

Calculates a bit-mask raster

Calculates a bit-mask, composed of flags of the given product and combined as described in the given bit-mask expression, for the a certain dimension and sub-sampling as defined in the given raster.

#### Parameters

- **bm\_expr** – a string holding the logical expression for the definition of the bit-mask. In a bit-mask expression, any number of the flag-names (found in the DDDb) can be composed with “(”, “)”, “NOT”, “AND”, “OR”. Valid bit-mask expression are for example `flags.LAND OR flags.CLOUD` or `NOT flags.WATER AND flags.TURBID_S`
- **xoffset** – across-track co-ordinate in pixel co-ordinates (zero-based) of the upper right corner of the source-region
- **yoffset** – along-track co-ordinate in pixel co-ordinates (zero-based) of the upper right corner of the source-region
- **raster** – the raster for the bit-mask. The data type of the raster must be either [E\\_TID\\_UCHAR](#) or [E\\_TID\\_CHAR](#)

**Returns** zero for success, an error code otherwise

**See Also:**

[create\\_bitmask\\_raster\(\)](#)

**get\_band\_names()**

Return the list of names of the bands in the product

---

**Note:** this method has no correspondent in the C API

---

---

**datasets ()**

Return the list of dataset in the product

---

**Note:** this method has no correspondent in the C API

---

**bands ()**

Return the list of bands in the product

---

**Note:** this method has no correspondent in the C API

---

**Implemented special methods**

- `__repr__`
- `__str__`

### 3.1.2 Dataset

**class** `epr.Dataset`  
ENVISAT dataset

The Dataset class contains information about a dataset within an ENVISAT product file which has been opened with the `open ()` function.

A new Dataset instance can be obtained with the `Product.get_dataset ()` or `Product.get_dataset_at ()` methods.

**description**

A short description of the band's contents

**product**

The `Product` instance to which this dataset belongs to

**get\_name ()**

Gets the name of the dataset

**get\_dsd ()**

Gets the dataset descriptor (DSD)

**get\_dsd\_name ()**

Gets the name of the DSD (dataset descriptor)

**get\_num\_records ()**

Gets the number of records of the dataset

**create\_record ()**

Creates a new record

Creates a new, empty record with a structure compatible with the dataset. Such a record is typically used in subsequent calls to `Dataset.read_record()`.

**Returns** the new record instance

**read\_record**(*index*[, *record*])

Reads specified record of the dataset

The record is identified through the given zero-based record index. In order to reduce memory reallocation, a record (pre-)created by the method `Dataset.create_record()` can be passed to this method. Data is then read into this given record.

If no record (`None`) is given, the method initiates a new one.

In both cases, the record in which the data is read into will be returned.

#### Parameters

- **index** – the zero-based record index
- **record** – a pre-created record to reduce memory reallocation, can be `None` (default) to let the function allocate a new record

**Returns** the record in which the data has been read into or raises an exception (`EPRValueError`) if an error occurred

**records**()

Return the list of records contained in the dataset

---

**Note:** this method has no correspondent in the C API

---

#### Implemented special methods

- `__repr__`
- `__str__`
- `__iter__`

### 3.1.3 Record

**class** `epr.Record`

Represents a record read from an ENVISAT dataset

A record is composed of multiple fields.

**See Also:**

`Field`

**get\_field**(*name*)

Gets a field specified by name

The field is here identified through the given name. It contains the field info and all corresponding values.

**Parameters** *name* – the the name of required field

**Returns** the specified `Field` or raises an exception (`EPRValueError`) if an error occurred

**get\_field\_at** (*index*)

Gets a field at the specified position within the record

**Parameters** *index* – the zero-based index (position within record) of the field

**Returns** the field or raises and exception (`EPRValueError`) if an error occurred

**get\_num\_fields** ()

Gets the number of fields contained in the record

**print\_** ([*ostream*])

Write the record to specified file (default: `sys.stdout`<sup>9</sup>)

This method writes formatted contents of the record to specified *ostream* text file or (default) the ASCII output is be printed to standard output (`sys.stdout`<sup>10</sup>)

**Parameters** *ostream* – the (opened) output file object

---

**Note:** the *ostream* parameter have to be a *real* file not a generic stream object like `StringIO.StringIO`<sup>11</sup> instances

---

**print\_element** (*field\_index*, *element\_index*[, *ostream*])

Write the specified field element to file (default: `sys.stdout`<sup>12</sup>)

This method writes formatted contents of the specified field element to the *ostream* text file or (default) the ASCII output will be printed to standard output (`sys.stdout`<sup>13</sup>)

**Parameters**

- **field\_index** – the index of field in the record
- **element\_index** – the index of element in the specified field
- **ostream** – the (opened) output file object

---

**Note:** the *ostream* parameter have to be a *real* file not a generic stream object like `StringIO.StringIO`<sup>14</sup> instances

---

---

<sup>9</sup><http://docs.python.org/library/sys.html#sys.stdout>

<sup>10</sup><http://docs.python.org/library/sys.html#sys.stdout>

<sup>11</sup><http://docs.python.org/library/stringio.html#StringIO.StringIO>

<sup>12</sup><http://docs.python.org/library/sys.html#sys.stdout>

<sup>13</sup><http://docs.python.org/library/sys.html#sys.stdout>

<sup>14</sup><http://docs.python.org/library/stringio.html#StringIO.StringIO>



---

**get\_field\_names()**

Return the list of names of the fields in the product

---

**Note:** this method has no correspondent in the C API

---

**fields()**

Return the list of fields contained in the record

---

**Note:** this method has no correspondent in the C API

---

### Implemented special methods

- `__repr__`
- `__str__`
- `__iter__`

## 3.1.4 Field

**class** `epr.Field`

Represents a field within a record

A field is composed of one or more data elements of one of the types defined in the internal `field_info` structure.

**See Also:**

[Record](#)

**get\_description()**

Gets the description of the field

**get\_name()**

Gets the name of the field

**get\_num\_elems()**

Gets the number of elements of the field

**get\_type()**

Gets the type of the field

**get\_unit()**

Gets the unit of the field

**get\_elem([index])**

Field single element access

This function is for getting the elements of a field.

**Parameters** *index* – the zero-based index of element to be returned, must not be negative. Default: 0.

**Returns** the typed value from given field

**get\_elems** ()

Field array element access

This function is for getting an array of field elements of the field.

**Returns** the data array (`numpy.ndarray`<sup>15</sup>) having the type of the field

**print\_** ([*ostream*])

Write the field to specified file (default: `sys.stdout`<sup>16</sup>)

This method writes formatted contents of the field to specified *ostream* text file or (default) the ASCII output is be printed to standard output (`sys.stdout`<sup>17</sup>)

**Parameters** *ostream* – the (opened) output file object

---

**Note:** the *ostream* parameter have to be a *real* file not a generic stream object like `StringIO.StringIO`<sup>18</sup> instances

---

### Implemented special methods

- `__repr__`
- `__str__`
- `__eq__`
- `__ne__`
- `__len__`<sup>19</sup>

## 3.1.5 DSD

**class** `epr.DSD`

Dataset descriptor

The DSD class contains information about the properties of a dataset and its location within an ENVISAT product file

**ds\_name**

The dataset name

---

<sup>15</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>

<sup>16</sup><http://docs.python.org/library/sys.html#sys.stdout>

<sup>17</sup><http://docs.python.org/library/sys.html#sys.stdout>

<sup>18</sup><http://docs.python.org/library/stringio.html#StringIO.StringIO>

<sup>19</sup> if the field is a `E_TID_STRING` field then the `__len__` () method returns the string length, otherwise the number of elements of the field is returned (same as `Field.get_num_elems()`)

**ds\_offset**

The offset of dataset-information the product file

**ds\_size**

The size of dataset-information in dataset product file

**ds\_type**

The dataset type descriptor

**dsrc\_size**

The size of dataset record for the given dataset name

**filename**

The filename in the DDDDB with the description of this dataset

**index**

The index of this DSD (zero-based)

**num\_dsr**

The number of dataset records for the given dataset name

**Implemented special methods**

- `__repr__`
- `__eq__`
- `__ne__`

### 3.1.6 Band

**class `epr.Band`**

The band of an ENVISAT product

The Band class contains information about a band within an ENVISAT product file which has been opened with the `open()` function.

A new Band instance can be obtained with the `Product.get_band()` method.

**bm\_expr**

A bit-mask expression used to filter valid pixels

All others are set to zero

**data\_type**

The data type of the band's pixels

Possible values are:

- `*` -> the datatype remains unchanged.
- `uint8_t` -> 8-bit unsigned integer
- `uint32_t` -> 32-bit unsigned integer

- `Float` -> 32-bit IEEE floating point

**description**

A short description of the band's contents

**lines\_mirrored**

Mirrored lines flag

If true (=1) lines will be mirrored (flipped) after read into a raster in order to ensure a pixel ordering in raster X direction from WEST to EAST.

**product**

The `Product` instance to which this band belongs to

**sample\_model**

The sample model operation

The sample model operation applied to the source dataset for getting the correct samples from the MDS (for example MERIS L2).

Possible values are:

- `*` -> no operation (direct copy)
- `1OF2` -> first byte of 2-byte interleaved MDS
- `2OF2` -> second byte of 2-byte interleaved MDS
- `0123` -> combine 3-bytes interleaved to 4-byte integer

**scaling\_factor**

The scaling factor

Possible values are:

- `*` -> no factor provided (implies `scaling_method=*`)
- `const` -> a floating point constant
- `GADS.field[.field2]` -> value is provided in global annotation dataset with name *GADS* in field *field*'. Optionally a second element index for multiple-element fields can be given too

**scaling\_method**

The scaling method which must be applied to the raw source data in order to get the 'real' pixel values in geo-physical units.

Possible values are:

- `*` -> no scaling applied
- `Linear_Scale` -> linear scaling applied:  
$$y = \text{offset} + \text{scale} * x$$
- `Log_Scale` -> logarithmic scaling applied:

```
y = log10(offset + scale * x)
```

**scaling\_offset**

Possible values are:

- \* -> no offset provided (implies scaling\_method=\*)
- const -> a floating point constant
- GADS.field[.field2] ` --> value is provided in global annotation dataset with name ``GADS in field field. Optionally a second element index for multiple-element fields can be given too

**spectr\_band\_index**

The (zero-based) spectral band index

-1 if this is not a spectral band

**unit**

The geophysical unit for the band's pixel values

**get\_name()**

Gets the name of the band

**create\_compatible\_raster** (*src\_width*, *src\_height*[, *xstep*, *ystep*])

Creates a raster which is compatible with the data type of the band

The created raster is used to read the data in it (see `Band.read_raster()`).

The raster is defined on the grid of the product, from which the data are read. Spatial subsets and under-sampling are possible) through the parameter of the method.

A raster is an object that allows direct access to data of a certain portion of the ENVISAT product that are read into the it. Such a portion is called the source. The complete ENVISAT product can be much greater than the source. One can move the raster over the complete ENVISAT product and read in turn different parts (always of the size of the source) of it into the raster. The source is specified by the parameters *height* and *width*.

A typical example is a processing in blocks. Lets say, a block has 64x32 pixel. Then, my source has a width of 64 pixel and a height of 32 pixel.

Another example is a processing of complete image lines. Then, my source has a widths of the complete product (for example 1121 for a MERIS RR product), and a height of 1). One can loop over all blocks read into the raster and process it.

In addition, it is possible to defined a sub-sampling step for a raster. This means, that the source is not read 1:1 into the raster, but that only every 2nd or 3rd pixel is read. This step can be set differently for the across track (*source\_step\_x*) and along track (*source\_step\_y*) directions.

**Parameters**

- **src\_width** – the width (across track dimension) of the source to be read into the raster
- **src\_height** – the height (along track dimension) of the source to be read into the raster
- **xstep** – the sub-sampling step across track of the source when reading into the raster. Default: 1.
- **ystep** – the sub-sampling step along track of the source when reading into the raster. Default: 1.

**Returns** the new raster instance or raises an exception (`EPRValueError`) if an error occurred

**read\_raster** (`[xoffset, yoffset, raster]`)

Reads (geo-)physical values of the band of the specified source-region

The source-region is a defined part of the whole ENVISAT product image, which shall be read into a raster. In this routine the co-ordinates are specified, where the source-region to be read starts. The dimension of the region and the sub-sampling are attributes of the raster into which the data are read.

#### Parameters

- **xoffset** – across-track source co-ordinate in pixel co-ordinates (zero-based) of the upper right corner of the source-region. Default 0.
- **yoffset** – along-track source co-ordinate in pixel co-ordinates (zero-based) of the upper right corner of the source-region. Default 0.
- **raster** – `Raster` instance set with appropriate parameters to read into. If not provided a new `Raster` is instantiated

**Returns** the `Raster` instance in which data are read

This method raises an instance of the appropriate `EPRError` sub-class if case of errors

#### See Also:

`Band.create_compatible_raster()` and `create_raster()`

**read\_as\_array** (`[width, height, xoffset, yoffset, xstep, ystep]`)

Reads the specified source region as an `numpy.ndarray`<sup>20</sup>

The source-region is a defined part of the whole ENVISAT product image, which shall be read into a raster. In this routine the co-ordinates are specified, where the source-region to be read starts. The dimension of the region and the sub-sampling are attributes of the raster into which the data are read.

#### Parameters

- **src\_width** – the width (across track dimension) of the source to be read into the raster. If not provided reads as much as possible

---

<sup>20</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>

- **src\_height** – the height (along track dimension) of the source to be read into the raster, If not provided reads as much as possible
- **xoffset** – across-track source co-ordinate in pixel co-ordinates (zero-based) of the upper right corner of the source-region. Default 0.
- **yoffset** – along-track source co-ordinate in pixel co-ordinates (zero-based) of the upper right corner of the source-region. Default 0.
- **xstep** – the sub-sampling step across track of the source when reading into the raster. Default: 1
- **ystep** – the sub-sampling step along track of the source when reading into the raster. Default: 1

**Returns** the `numpy.ndarray`<sup>21</sup> instance in which data are read

This method raises an instance of the appropriate `EPRError` sub-class if case of errors

---

**Note:** this method has no correspondent in the C API

---

**See Also:**

`Band.create_compatible_raster()`, `create_raster()` and  
`Band.read_raster()`

### Implemented special methods

- `__repr__`

## 3.1.7 Raster

**class** `epr.Raster`

Represents a raster in which data will be stored

All 'size' parameter are in PIXEL.

**data\_type**

The data type of the band's pixels

All `E_TID_*` types are possible

**source\_height**

The height of the source

**source\_width**

The width of the source

---

<sup>21</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>

**source\_step\_x**

The sub-sampling for the across-track direction in pixel

**source\_step\_y**

The sub-sampling for the along-track direction in pixel

**data**

Raster data exposed as `numpy.ndarray`<sup>22</sup> object

---

**Note:** this property shares the data buffer with the `Raster` object so any change in its contents is also reflected to the `Raster` object

---

---

**Note:** this property has no correspondent in the C API

---

**get\_pixel** (*x*, *y*)

Single pixel access

This function is for getting the values of the elements of a raster (i.e. pixel)

**Parameters**

- **x** – the (zero-based) X coordinate of the pixel
- **y** – the (zero-based) Y coordinate of the pixel

**Returns** the typed value at the given co-ordinate

**get\_elem\_size** ()

The size in byte of a single element (sample) of this raster's buffer

**get\_height** ()

Gets the raster's height in pixels

**get\_width** ()

Gets the raster's width in pixels

**Implemented special methods**

- `__repr__`

### 3.1.8 EPRTIME

**class** `epr.EPRTIME`

Convenience class for time data exchange.

`EPRTIME` is a `collections.namedtuple` with the following fields:

**days**

---

<sup>22</sup><http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray>



**seconds**

**microseconds**

## 3.2 Functions

`epr.open(filename)`

Opens the ENVISAT product

Opens the ENVISAT product file with the given file path, reads MPH, SPH and all DSDs, organized the table with parameter of line length and tie points number.

**Parameters** `product_file_path` – the path to the ENVISAT product file

**Returns** the `Product` instance representing the specified product. An exception (`exceptions.ValueError`<sup>23</sup>) is raised if the file could not be opened.

`epr.data_type_id_to_str()`

Gets the 'C' data type string for the given data type

`epr.get_data_type_size()`

Gets the size in bytes for an element of the given data type

`epr.get_sample_model_name()`

Return the name of the specified sample model

`epr.get_scaling_method_name()`

Return the name of the specified scaling method

`epr.create_raster(data_type, src_width, src_height[, xstep, ystep])`

Creates a raster of the specified data type

This function can be used to create any type of raster, e.g. for later use as a bit-mask.

### Parameters

- **data\_type** – the type of the data to stored in the raster, must be one of `E_TID_*`.

### See Also:

#### Data type Identifiers

- **src\_width** – the width (across track dimension) of the source to be read into the raster. See description of `Band.create_compatible_raster()`
- **src\_height** – the height (along track dimension) of the source to be read into the raster. See description of `Band.create_compatible_raster()`

---

<sup>23</sup><http://docs.python.org/library/exceptions.html#exceptions.ValueError>

- **xstep** – the sub-sampling step across track of the source when reading into the raster. Default: 1.
- **ystep** – the sub-sampling step along track of the source when reading into the raster. Default: 1.

**Returns** the new `Raster` instance

**See Also:**

description of `Band.create_compatible_raster()`

`epr.create_bitmask_raster(src_width, src_height[, xstep, ystep])`

Creates a raster to be used for reading bitmasks

The raster returned always is of type `byte`.

**Parameters**

- **src\_width** – the width (across track dimension) of the source to be read into the raster
- **src\_height** – the height (along track dimension) of the source to be read into the raster
- **xstep** – the sub-sampling step across track of the source when reading into the raster. Default: 1.
- **ystep** – the sub-sampling step along track of the source when reading into the raster. Default: 1.

**Returns** the new raster instance or raises an exception (`EPRValueError`) if an error occurred

**See Also:**

the description of `Band.create_compatible_raster()`

## 3.3 Exceptions

### 3.3.1 EPRError

**exception** `epr.EPRError`

EPR API error

**code**

EPR API error code

`__init__([message[, code, *args, **kwargs]])`

Initializer

**Parameters** `message` – error message

**Param** `code` EPR error code

### 3.3.2 EPRValueError

**exception** `epr.EPRValueError`

Inherits both `EPRError` and standard `exceptions.ValueError`<sup>24</sup>

## 3.4 Data

`epr.EPR_C_API_VERSION`

Version string of the wrapped `EPR API`<sup>25</sup> C library

### 3.4.1 Data type identifiers

`epr.E_TID_UNKNOWN`

`epr.E_TID_UCHAR`

`epr.E_TID_CHAR`

`epr.E_TID_USHORT`

`epr.E_TID_SHORT`

`epr.E_TID_UINT`

`epr.E_TID_INT`

`epr.E_TID_FLOAT`

`epr.E_TID_DOUBLE`

`epr.E_TID_STRING`

`epr.E_TID_SPARE`

`epr.E_TID_TIME`

### 3.4.2 Sample Models

`epr.E_SMOD_1OF1`

`epr.E_SMOD_1OF2`

`epr.E_SMOD_2OF2`

`epr.E_SMOD_3TOI`

`epr.E_SMOD_2TOF`

---

<sup>24</sup><http://docs.python.org/library/exceptions.html#exceptions.ValueError>

<sup>25</sup><https://github.com/bcdev/epr-api>

### 3.4.3 Scaling Methods

`epr.E_SMID_NON`

No scaling

`epr.E_SMID_LIN`

Linear pixel scaling

`epr.E_SMID_LOG`

Logarithmic pixel scaling

## CHANGE HISTORY

### 4.1 PyEPR 0.6.1 (26/04/2012)

- fix compatibility with [cython](http://www.cython.org)<sup>1</sup> 0.16
- added a new option to the setup script (`--epr-api-src`) to build PyEPR using the EPR-API C sources

### 4.2 PyEPR 0.6 (12/08/2011)

- full support for [Python 3](http://www.python.org)<sup>2</sup>
- improved code highlight in the documentation
- depend from `cython >= 0.13` instead of `cython >= 0.14.1`. Cythonizing `epr.pyx` with [Python 3](http://www.python.org)<sup>3</sup> requires `cython >= 0.15`

### 4.3 PyEPR 0.5 (25/04/2011)

- stop using `PyFile_AsFile()`<sup>4</sup> that is no more available in [Python 3](http://www.python.org)<sup>5</sup>
- now documentation uses [intersphinx](http://www.sphinx-doc.org)<sup>6</sup> capabilities
- code examples added to documentation
- tutorials added to documentation
- the `LICENSE.txt` file is now included in the source distribution
- the [cython](http://www.cython.org)<sup>7</sup> construct with `nogil` is now used instead of calling

---

<sup>1</sup><http://www.cython.org>

<sup>2</sup><http://docs.python.org/py3k>

<sup>3</sup><http://docs.python.org/py3k>

<sup>4</sup>[http://docs.python.org/c-api/file.html#PyFile\\_AsFile](http://docs.python.org/c-api/file.html#PyFile_AsFile)

<sup>5</sup><http://docs.python.org/py3k>

<sup>6</sup><http://sphinx.pocoo.org/latest/ext/intersphinx.html>

<sup>7</sup><http://www.cython.org>

`Py_BEGIN_ALLOW_THREADS()` and `Py_END_ALLOW_THREADS()` directly

- dropped old versions of [cython](http://www.cython.org)<sup>8</sup>; now [cython](http://www.cython.org)<sup>9</sup> 0.14.1 or newer is required
- suppressed several constness related warnings

## 4.4 PyEPR 0.4 (10/04/2011)

- fixed a bug in the `epr.Product.__str__()`, `Dataset.__str__()` and `erp.Band.__repr__()` methods (bad formatting)
- fixed `epr.Field.get_elems()` method for char and uchar data types
- implemented `epr.Product.read_bitmask_raster()`, now the `epr.Product` API is complete
- fixed segfault in `epr.Field.get_unit()` method when the field has no unit
- a smaller dataset is now used for unit tests
- a new tutorial section has been added to the user documentation

## 4.5 PyEPR 0.3 (01/04/2011)

- version string of the EPR C API is now exposed as module attribute `epr.EPR_C_API_VERSION`
- implemented `__repr__`, `__str__`, `__eq__`, `__ne__` and `__iter__` special methods
- added utility methods (not included in the C API) like:
  - `epr.Record.get_field_names()`
  - `epr.Record.fields()`
  - `epr.Dataset.records()`
  - `epr.Product.get_dataset_names()`
  - `epr.Product.get_band_names()`
  - `epr.Product.datasets()`
  - `epr.Product.bands()`
- fixed a logic error that caused empty messages in custom EPR exceptions

---

<sup>8</sup><http://www.cython.org>

<sup>9</sup><http://www.cython.org>

## 4.6 PyEPR 0.2 (20/03/2011)

- [sphinx](#)<sup>10</sup> documentation added
- added docstrings to all method and classes
- renamed some method and parameter in order to avoid redundancies and have a more *pythonic* API
- in case of null pointers a `epr.EPRValueError` is raised
- improved C library shutdown management
- introduced some utility methods to `epr.Product` and `epr.Record` classes

## 4.7 PyEPR 0.1 (09/03/2011)

Initial release

---

<sup>10</sup><http://sphinx.pocoo.org>





# LICENSE

Copyright (C) 2011-2012 Antonio Valentino <[antonio.valentino@tiscali.it](mailto:antonio.valentino@tiscali.it)<sup>1</sup>>

PyEPR is free software: you can redistribute it and/or modify it under the terms of the [GNU General Public License](#)<sup>2</sup> as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

PyEPR is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with PyEPR. If not, see <<http://www.gnu.org/licenses/>>.

---

<sup>1</sup>[antonio.valentino@tiscali.it](mailto:antonio.valentino@tiscali.it)

<sup>2</sup><http://www.gnu.org/licenses/gpl-3.0.html>