



Upgrade to the new ATK platform

Guide for experienced users of ATK 2008.10

Version 12.2.0

Upgrade to the new ATK platform: Guide for experienced users of ATK 2008.10

Version 12.2.0

Copyright © 2008–2012 QuantumWise A/S

Atomistix ToolKit Copyright Notice

All rights reserved.

This publication may be freely redistributed in its full, unmodified form. No part of this publication may be incorporated or used in other publications without prior written permission from the publisher.

TABLE OF CONTENTS

1. Introduction	1
2. Upgrading from Atomistix ToolKit 2008.10	2
Naming	2
Changes in the two-probe model	2
Backwards compatibility	6
Work flow	7
Where did that tool in VNL go?	8
New convergence criterion	10
Changes in Python	11
New spin features	14
New license system	14
What's new and what's missing	15
What's next	15

CHAPTER 1. INTRODUCTION

This document is intended for users familiar with ATK 2008.10, to document the changes that have been made in the new ATK platform, by which is meant the releases with version number 10.8 and higher. With this information, we hope that upgrading to the new versions will be quick and easy.

Here is a short list of the most important things:

- [Changes in naming](#)
- [Changes in the transport algorithm](#)
- [Voltage drop](#)
- [New definition of the two-probe geometry](#)
- [Initialize an anti-parallel spin calculation from the parallel one](#)
- [How do I import my old geometries?](#)
- [How do I calculate the current?](#)
- [New license system](#)
- [New features](#)

We strongly recommend having a look at the new [mini-tutorials](#) to quickly get acquainted with a lot of the new functionality in VNL.

CHAPTER 2. UPGRADING FROM ATOMISTIX TOOLKIT 2008.10

NAMING

The following changes are introduced in some important **names and definitions**:

- The command-line binary, earlier called **atk**, is now called **atkpython**. So, you run your calculations from the command-line as

```
atkpython script.py > script.log &
```

- In most places, we now refer to **devices**, rather than **two-probe systems**. The primary reason for this is that in the future, ATK will support calculations with one, two, three, or more electrodes. You will also note that there no longer are explicit references to "left" and "right" electrodes in NanoLanguage, instead the electrodes are provided as a list.
- The two modules **ATK.KohnSham** and **ATK.TwoProbe** have been merged into one, called **NanoLanguage**. One can however omit the **import** statement altogether, since ATK will load it automatically.

CHANGES IN THE TWO-PROBE MODEL

The new ATK platform introduces several important changes to two-probe systems, or device configurations. These are designed to improve the physical and numerical consistency of the model, make it easier to set up the geometry, and to ensure that the calculations converge easier.

NEW TRANSPORT ALGORITHM

Compare the two pictures below – they illustrate the fundamental assumptions made in the calculation of transport properties in ATK 2008.10 and the new platform, respectively.

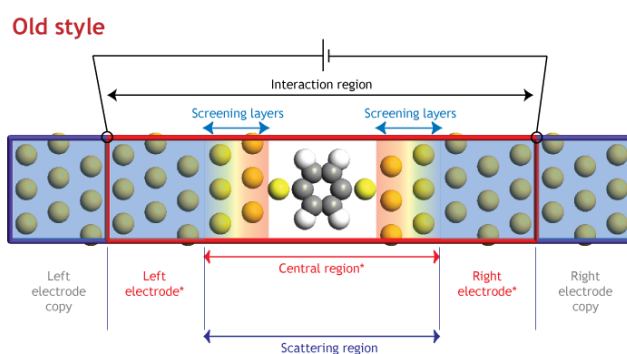


Figure 2.1:Old-style two-probe algorithm (as used in ATK 2008.10 and earlier).

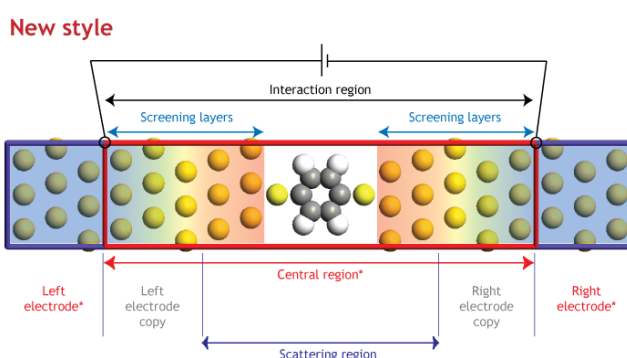


Figure 2.2:New-style two-probe algorithm (as used in ATK 10.8 and later).

Before discussing the differences and similarities between the two approaches, let us review some important definitions and explain the colors in the figures above:

- Areas labeled in **red** and with an asterisk * are defined by the user, as input. That is, the two **electrodes** and the **central region**.
- Areas with a solid blue shading are, in both pictures, regions where the Hamiltonian is assumed to be bulk-like. The rainbow-colored areas define the **screening** regions.
- The **interaction region** is where the effective potential and electron density are treated self-consistently. This is also the region across which the bias is applied. The two-probe calculation, with open boundary conditions, is carried out for the atoms in this region. It also corresponds to the **equivalent bulk system**, which can be converged in a preliminary step to provide an initial guess for the two-probe calculation.
- The electrodes and the electrode copies have a bulk-like geometry.
- The electrodes have a bulk-like electronic structure.
- The **central region** is the area where the electronic structure is not bulk-like.
- The **scattering region** is the part of the geometry that is not just a simple extension of the electrodes (not geometrically bulk-like). For instance, it contains the molecular part of a metal-molecule-metal junction, or the portion of a nanotube with a dopant or impurity. This region also contains **surface layers** of the metal, which may be relaxed to describe a realistic surface configuration.

- The **screening** layers essentially shield the electrodes from the electrostatic potential in the scattering region, to allow the electronic structure to approach its bulk-like form in a smooth way, from the non-bulk scattering region towards the electrodes.

Clearly there are many similarities between the two methods. The primary **difference** is the following:

! Important

- In the old algorithm, the Hamiltonian was treated as bulk-like in the electrodes. At the same time, the potential and sometimes also the density matrix (depending on the [constraint](#)) were computed self-consistently. This could cause a mismatch between these quantities, unless you had a lot of screening layers.
- In the new method **the entire interaction region is treated fully self-consistently**. This represents the most consistent and physically correct way to treat this kind of systems to date, and it improves the accuracy of the results as well as the self-consistent convergence quite a bit.

i Tip

In the new version, it is usually only necessary to set the k-point sampling and electronic temperature to achieve convergence. For other parameters, that you may have tried changing in ATK 2008.10, we recommend first trying the new defaults.

In the old-style method, an electrode copy is automatically placed outside the interaction region to provide the boundary condition for the electrostatic potential. In the new-style approach, the user has to take care to include a copy of the electrode inside the central region.

Although one needs to enter more coordinates into the central region atom list now, the new algorithm is not heavier than the old one; it's exactly the same number of atoms in both cases, they are just labeled differently.

📌 Note

In some cases it may be possible to use fewer atoms in the new method, since the electrode copies also provide screening. On the other hand, the electrode copies in the central region must have a bulk geometry, so if a relaxed surface is desired, dedicated surface layers will still be necessary.

See also the [section below on the device geometry definition](#).

VOLTAGE DROP

In ATK 2008.10 it was necessary to use a (not documented) constraint called **DensityMatrix** to get a correct voltage drop. Thanks to the new algorithm mentioned just above, a correct voltage drop can now be obtained without any constraints (i.e., using the default constraint **Off**).

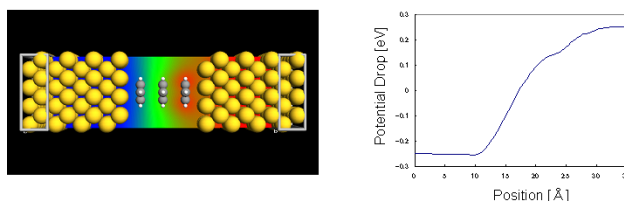


Figure 2.3: Voltage drop computed with ATK 10.8 with the default constraint (**Off**).

The **DensityMatrix** constraint is still available, for cases of very difficult convergence, where it might help to reach a self-consistent solution. The recommendation is, to then restart the calculation from this solution, and converge it again without any constraints.

DEVICE GEOMETRY DEFINITION

There are two primary changes in device geometries vs. two-probe configurations:

- The electrodes that are placed inside the interaction region must be **exact copies** of the actual electrodes. If the left electrode has n atoms, the the first n atoms in the central region must be an exact copy of the left electrode atoms. If there are m atoms in the right electrode the last m atoms in the central region must be a copy of the right electrode, however, with a displaced z coordinate such that the distance to the right face of the unit cell is the same in the electrode and the central region.
- Instead of using equivalent atoms to implicitly align the interaction region with the electrodes, the **alignment is absolute** in the new ATK platform. Basically, you just set up three periodic configurations, one for each electrode and one for the the central region, and when defining the two-probe geometry, ATK will simply line up the three unit cells next to each other.



Note

The positions of the atoms should be given relative to the local origin of each subsystem; do not add the length of the left electrode to the z coordinates of the interaction region!

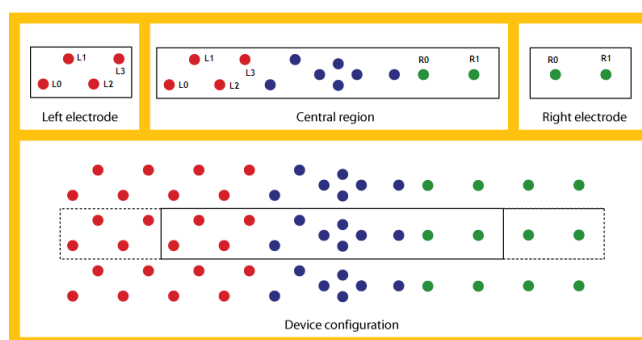


Figure 2.4: Upper panel: The constituents of a device system. Although this is a schematic picture, we can imagine it to be a molecule between two, different, metal surfaces. Red atoms define the left electrode, green the right, and blue the scattering region. Note how the left-most and right-most atoms in the central region are positioned identically to the atoms in the respective electrodes, as indicated by the indices ("L" for left, "R" for right). The two left-most and the one right-most "blue" atoms also belong to the metallic surfaces; they provide screening (as do the red/green atoms in the central region), and in addition they may be relaxed (the indexed atoms may not, as their positions must match exactly their counterparts in the electrodes). **Lower panel:** The resulting device configuration, after the unit cells have been aligned.



Note

In the Python definition of the device geometry, the first keyword is now the **central_region**, and the **electrodes** are the second. In ATK 2008.10 it was opposite.

```
central_region = ...
left_electrode = ...
right_electrode = ...
device_configuration = DeviceConfiguration(
    central_region,
```

```
[left_electrode, right_electrode]
)
```

Consult the [manual](#) for some more details.


PRACTICAL GUIDELINES

As a consequence, it is best to approach the geometry construction of device configurations a bit differently than before.

In VNL 2008.10 you would define the electrodes by cleaving a bulk crystal, and the surface screening was provided by layers "grown" out from the electrodes.

The new recommended approach is to build the complete interaction region (i.e. the so-called **equivalent bulk system**), and then specify which part, left and right, that is the respective **electrode copy**. These can then be turned into the actual electrodes.

In practice, this is most easiest done using the **Builder** in VNL. Once the interaction region has

been built, click on the "Device" icon  to the left. The Builder will attempt to identify any periodicity towards the edges of the configuration, and suggest various possible electrode lengths. Naturally, for this to work, there must be a periodicity; it is thus not enough to have 3 gold layers for fcc cleaved in [111], there must be at least 4.

Tip

Actually, you can set the system up with only 3 layers, but in that case you must specify the electrode length manually, which is always possible.

For a concrete example that illustrates these points, see the [mini-tutorial on how to build a 3x3 gold \[111\] system](#).

BACKWARDS COMPATIBILITY

- The geometries of all kinds of systems built in older versions can be imported in VNL 10.8 and later, provided they exist in the form of **Python scripts**. Simply drop any Python script on any of the instruments in VNL to import the structure defined in the script.

The import may fail, on occasion. If so, try to "wash" the script by opening it in the **Atomic Manipulator** in VNL 2008.10, and saving it as a new script from there. This is particularly important for two-probe systems! Sometimes it might be necessary to save these as **equivalent bulk systems**, and then import them into the builder in the new version, and convert them to a device configuration in a similar manner as described [above](#).

We have been forced to sacrifice some backwards compatibility when designing the new platform. This is a *one-time only thing*; in the future, backwards compatibility across new version will be paramount.

- **VNL files** cannot be read on the new platform. One must first open the VNL file in the Atomic Manipulator in VNL 2008.10, and export the geometry as a Python script. The script can then be imported in the new platform, as described just above.


Data in VNL files, such as electron densities or transmission spectra stored in VNL files cannot be understood by the new platform.

- **NC files** from versions earlier than 10.8 cannot be read by the new platform.

WORK FLOW


Not much has changed in the basic work flow in VNL:

1. Set up the structure using one of the various builders.
2. Move it over to the Scriptor to define the parameters for the calculation.
3. Save the script to run it on a cluster (or locally) via the command line, or drop it directly on the Job Manager for direct execution (small scripts).

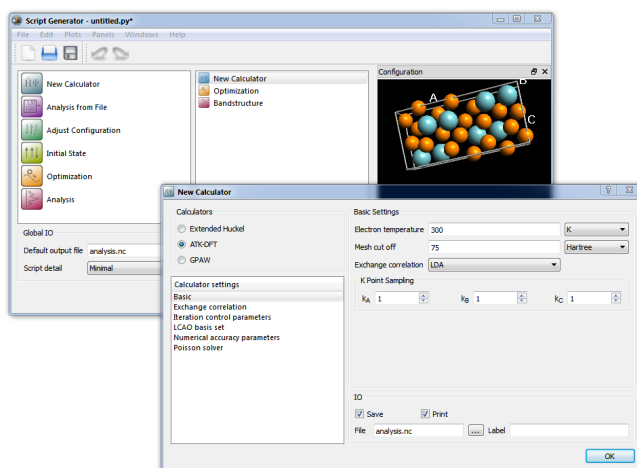
A very convenient new method for "sending" geometries and scripts between different tools is available via the "Send to" icon . The "drag-and-drop" functionality, via the icon



is also still available. Both icons are located in the lower right-hand corner of each window.


The **Script Generator** , or **Scripter** for short, has received a complete makeover. You

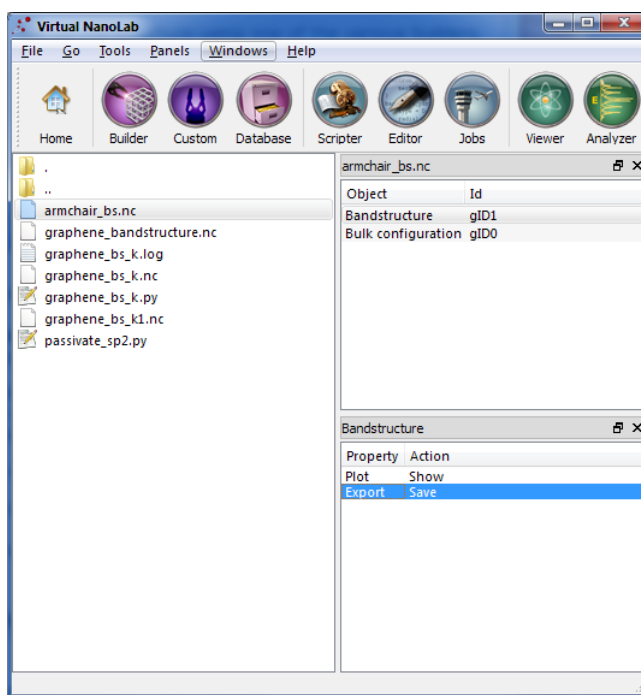
now operate on a meta-level with different building blocks such as **Initial state**, **Optimization** and various analysis options. You can in principle insert the blocks in any order you want (that makes sense), and the generator will create the complete Python script for you. This tool will continue to evolve in future releases to support e.g. looping over variables and user-defined blocks.



Post-convergence analysis has also changed a little bit. First of all, *there are no longer any VNL files*; all result data is saved in the **NetCDF file** together with the calculation itself. It is also much easier to do analysis with the GUI; just choose **Analysis from file** in the Scriptor to define the NC file, and then insert the relevant analysis quantities you want to compute. Moreover, anything that goes into the NC file can easily be extracted from it, either using scripting or via the result browser in VNL.

The main window of VNL now contains a **file/result browser** which directly shows you the contents of the NC files. Each quantity saved in the file can be inspected, e.g. a band structure


can be plotted (select it and click **Show** next to "Plot" in the lower panel) or an electron density can be visualized in 3D (drag it onto the **Viewer** icon ). The data from the NC files can also easily be **exported** for plotting elsewhere.

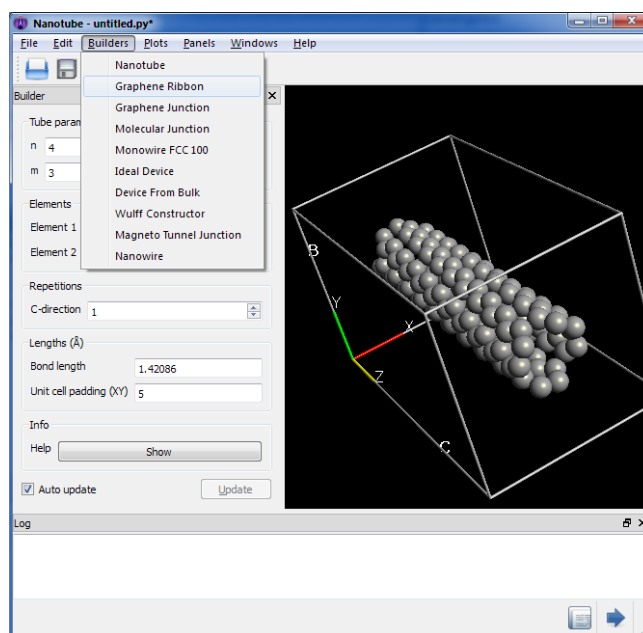


WHERE DID THAT TOOL IN VNL GO?

NANOTUBE GROWER / MTJ BUILDER


Both are now part of the Custom Builder framework. While the MTJ builder is a bit simplified, the nanotube builder has been upgraded, and now allows you to make boron-nitride tubes too, and apply repetitions directly.

In addition, there are a whole lot of other, new builders under the "Custom" icon  (click it, and open the Builder menu), which let you build graphene nanoribbons, metallic nanoclusters and nanowires, and many other systems.

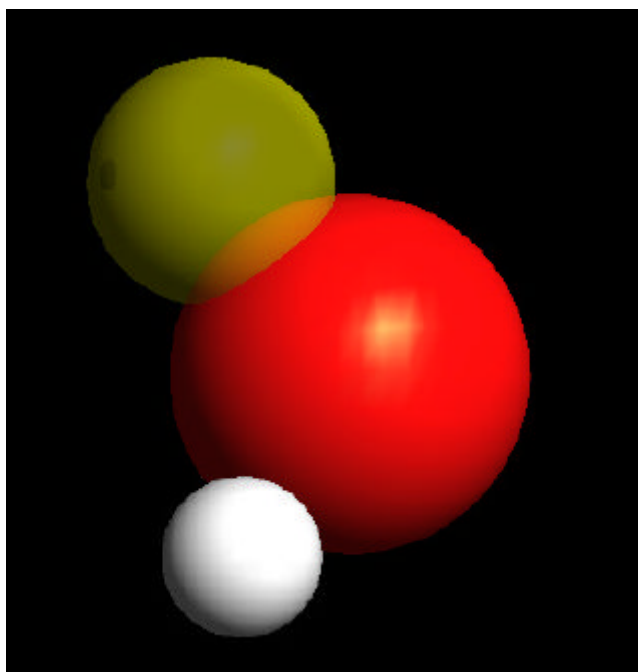


But that's not all: the Custom Builder, as the name indicates, is actually a plug-in tool. You can write your own builder and use it in the graphical environment in VNL. For more info see some of the other [tutorials](#).

NANOSCOPE

It is now called the **Viewer** . You will note some nice improvements in the visual appearance as well as the graphics performance, plus that you can now change the individual color and radius of individual atoms, and (optionally) use orthographic projection instead of stereographic, which can be very useful when looking at layered structures.

Below is a visualization of a water molecule, where the color, radius and transparency of the atoms has been changed.



CUPBOARDS

The Crystal and Molecular Cupboards have been merged into the **Database** tool.



There's also a fullerene cupboard in there now!

RESULT BROWSER

Now integrated with the file browser in the main VNL window. See [screen shot above](#).

ATOMIC MANIPULATOR / MOLECULAR BUILDER / BULK BUILDER

The new VNL version has one single Builder to handle all kinds of systems. It contains a Z-matrix editor, useful for molecules, although it doesn't have the same advanced bond recognition as in VNL 2008.10.

The functionality for building crystals has improved a lot thanks to improved selection and grouping methods, plus "repeat", "mirror", "translate", "rotate" and "copy/paste" operations that can be applied to selected atoms. The Z-matrix functionality is also available for periodic systems.

NEW CONVERGENCE CRITERION

The default tolerance criterion has been changed. The measured quantity for determining when the self-consistent loop has converged is now the **largest change in the Hamiltonian matrix elements**. The default tolerance is $5.0e-4$ (Hartree).

The new criterion means that calculations might use a different number of steps to converge, compared to before. Generally speaking, the new criterion is stricter, i.e. calculations are more properly converged now than when using the default tolerance in ATK 2008.10.

CHANGES IN PYTHON

For users who write their own Python code, there are a couple of fundamentally important changes in how ATK operates. We recommend using the Script Generator to create some templates to learn how the new functions and methods work for simple cases, before writing your own advanced scripts.

METHOD VS. CONFIGURATION + ANALYSIS

The new ATK platform adopts certain aspects of the Python methodology used in [ASE](#) from [CAMd](#). This differs from the way things were done previously in ATK in two main ways.

- The method is attached to the configuration before the self-consistent loop is run:

```
configuration = ...
calculator = LCAOCalculator()
configuration.setCalculator(calculator)
```

- There is a method called **update()** on the configuration. This works essentially as the old **executeSelfConsistentCalculation()**.

```
configuration = ...
calculator = LCAOCalculator(...)
configuration.setCalculator(calculator)
configuration.update()
nlsave('file.nc', configuration)
```

- The configuration is clever enough to find out if a self-consistent loop already has been performed when you ask for some physical quantity to be calculated. Therefore, there is actually no real need to explicitly run the self-consistent loop; just ask the configuration for the transmission spectrum, and it will perform the self-consistent loop if required, and then return the spectrum:

```
configuration = DeviceConfiguration(...)
calculator = DeviceLCAOCalculator(...)
configuration.setCalculator(calculator)
transmission = TransmissionSpectrum(...)
nlsave('file.nc', configuration)
nlsave('file.nc', transmission)
```

All this might sound a bit complex, but the basic idea is simple. This is how you would calculate the band structure of gold:

```
lattice = FaceCenteredCubic(4.07825*Angstrom)
bulk_configuration = BulkConfiguration(
    bravais_lattice=lattice,
    elements=[Gold],
    fractional_coordinates=[[ 0., 0., 0.]]
)
numerical_accuracy_parameters = NumericalAccuracyParameters(
    k_point_sampling=(9, 9, 9),
)
calculator = LCAOCalculator(
    numerical_accuracy_parameters=numerical_accuracy_parameters,
)
bulk_configuration.setCalculator(calculator)
bandstructure = Bandstructure(
```

```
configuration=bulk_configuration,
route=['G', 'X', 'W', 'L', 'G', 'K', 'X', 'U', 'W', 'K', 'L'],
points_per_segment=20,
bands_above_fermi_level=All
)
nlsave('gold.nc', bulk_configuration, labels=['Gold'])
nlsave('gold.nc', bandstructure, labels=['Gold'])
```

This means that the configuration itself holds both the geometry and the result of the self-consistent calculation. Thus it acts as the old "scf" objects in 2008.10, but the object has many more methods and interfaces which opens up for several different advanced manipulations that were not possible before.

FILE I/O

Results are saved to NetCDF files via the `nlsave()` function. The results can be read back from the NC file using the function `nload()`. The two commands are transparent and reversible: what can be saved in NC files can be read from the file.

You can tag each object saved in the NC file on several different levels:

- **object_id**, must be unique within a file.
- **labels**, a list of strings to identify the quantity. Suggested use: assign the same label to the configuration and all results computed from it, to indicate that they belong to each other (see for example how we used the label "Gold" in [the gold band structure example above](#) [11]).
- **comment**, a free-text descriptive comment string.

For more information, see the [manual for nlsave\(\)](#).

MPI SAFETY

All internal functions in ATK are "MPI safe" meaning that they only perform I/O operations on the master node.

If you need to print something to the terminal/log file in scripts that are being run in parallel, we recommend using the function `nloadprint` instead of the usual `print` statement:

```
nloadprint("Only printed on the master node (1+1=%g)" % (1+1))
```

If other I/O operations, like writing to files, are required, you must protect them from being executed on the slave nodes by using the function `processIsMaster()`:

```
if processIsMaster():
    print "Only printed on the master node"
```

Warning

Make sure not to put any native ATK statements inside such a block, or they may hang the script, waiting for slave nodes to respond.

Note that *reading* from files should typically be done on all nodes, otherwise only the master will have the information.

RESULTS SAVED IN RICH OBJECTS

All analysis quantities that are computed are returned as object. These objects can be stored in (and restored from) the NC files, but note that they do not have any memory of which configuration that was used to compute them. This logic has to be maintained by the user, preferably via use of [labels](#).

The returned objects have rich interfaces, and often store supporting data. Thus, for example, when you compute the density of states, information is stored which makes it possible to project it on atoms and even angular momenta, without any further calculations. The transmission spectrum stores the transmission coefficients for each k-point and energy.

HOW DO I CALCULATE THE CURRENT?

As a consequence of the point just above, there is no need for a separate function to compute the current; the information needed for that is already available in the transmission spectrum. The calculation of the current is therefore done via a method on the transmission spectrum object:

```
transmission_spectrum = TransmissionSpectrum(
    configuration=device_configuration,
    energies=numpy.linspace(-0.5,0.5,10)*eV,
    kpoints=MonkhorstPackGrid(3,3)
)
current = transmission_spectrum.current()
```

By default this uses the electron temperatures specified in the method, for the two electrodes, but it can also be changed locally for the current calculation, in order to study electron thermoelectric transport.

For more detailed information, see the manual for [TransmissionSpectrum\(\)](#).

UNITS

In ATK 2008.10 and earlier, if you multiplied a list with a unit, you got a list of objects (each one a **PhysicalQuantity**). This meant, that to extract e.g. coordinates without unit, you had to loop over the list, which can be time-consuming and makes the script bulky.

In the new version, the unit is attached to the list only once, and the data is stored as an array. Therefore, you can ask for the raw numbers in a single statement:

```
a = [1,2,3]*Bohr
print a.inUnitsOf(Angstrom)
```

will return

```
[ 0.529177 1.058354 1.587531 ]
```

INTERACTIVE COMMAND-LINE MODE

Actually nothing has changed here, we just wish to point out that you can operate ATK interactively in command-line mode too. Just launch `atkpython` without any arguments, or

```
atkpython
```

```
atkpython -i script.py
```

to run `script.py` and then leave you inside Python. At the interactive prompt you can type any Python/NanoLanguage command you want and it will execute immediately. This is a good way to test new functionality, prototype a new script, or to do interactive analysis, since you can inspect the results directly, and if needed change the input parameters (k-points, or index in an array, for instance), without having to edit and rerun the whole script each time.

Note

If you ever used ATK 2008.10 in interactive command-line mode too, you will need to upgrade your local IPython settings. The program will inform you about this when you start it up the first time; as the instructions say, just run (inside `atkpython`)

```
%upgrade
```

Press **Ctrl-D** or type `exit()` to leave interactive mode!

Consult the [IPython manuals](#) to learn more.

NEW SPIN FEATURES

A very exciting new feature is the possibility to initialize an anti-parallel spin-polarized calculation from the converged state of the corresponding parallel calculation. ATK will simply flip the density matrix components for atoms with a negative initial spin.

Often the anti-parallel calculation is hard to converge, but using a "flipped version" of the parallel calculation as a starting guess greatly improves the convergence rate!

In addition, you will note that there is no setting for the initial spin of the electrode atoms. Instead, the spin of the electrodes is automatically set to match the initial spin configuration of the "electrode copies" in the central region (cf. [the discussion on the two-probe geometry](#)).

This is demonstrated in practice in the [mini-tutorial on two-probe spin calculations](#).

NEW LICENSE SYSTEM

The new ATK platform uses the **LM-X License Manager** from [X-Formation](#). All users must upgrade their licenses in order to be able to run the new version.

Important

The new licenses cannot be used to run ATK 2008.10, but all customers may continue using their 2008.10 as long as they prefer (or as long as they are valid).

NO MASTER LICENSE CONSUMED FOR BASIC PYTHON TASKS

One major exception to the rule of `atkpython` being a regular Python is obviously the requirement for a license to run it. However, unlike previous versions of ATK, the software does not

consume a "master" license just for starting up. The master (and slave) licenses are only checked out when you run a self-consistent calculation, or perform some analysis. For all other tasks, both NanoLanguage and regular Python, the only thing required is a **NLPython** license, and we provide those in abundance (1000 by default). Therefore, you can perform basic tasks in ATK even if all master licenses are occupied by calculations that are running.

WHAT'S NEW AND WHAT'S MISSING

Above we mainly discussed what has changed compared to 2008.10. There are a large number of truly new features too, and these are listed on the [website](#).

We have tried our hardest not to leave any features from 2008.10 out, in order to facilitate a smooth transition to the new version. Except for the way some tools operate in VNL, the only specific feature which is no longer available is the **gated method**. In a sense this simplistic method to simulate the influence of a gate electrode is not needed now that we can place real metallic gates and dielectric regions in the device geometry. However, it is admittedly a nice, simple method to estimate the influence of a gate, without any added cost or complexity, so it might make a comeback, if many users request it!

WHAT'S NEXT

Where does ATK go from here, what's next? Well, partly that is up to you, our users. Your input and feedback is very important to us! So, if you have any comments, or ideas for future improvements and new features, please let us know. The [Forum](#) is a good place for discussions, and you can of course always [contact us directly](#) as well.

Naturally, we also have a lot of ideas ourselves - ATK is in strong active development! In general, we plan to have two major releases every year and to maintain two versions concurrently:

- A **stable branch**, with frozen functionality but where we fix issues that are discovered in the latest major release.
- A **development branch**, where we introduce new functionality. These will be released in preview versions (alphas and betas) up until the point they are considered final, in which case a new major version is released to replace the previous stable branch.

It will be possible to use both branches simultaneously, to e.g. run the self-consistent calculation with the stable branch, and then use the beta release for special analysis only available in the new version.