
Overview of Data Environment Functionality

A very common mode of data analysis is the "script pipeline" model, in which:

- 1) Raw data is stored in a set of input files,
- 2) a set of scripts is applied to the raw data (usually by being run at a command line) to generate some desired computation, and the results are stored in new files, and then
- 3) another round of scripts is applied to the output files from step 2, to generate further computed data files.

By iteratively repeating this process, adding new raw data at various points along the way, and combining the results of different chains of analysis, complex data analysis script pipelines are built up. The script pipeline model is very effective because it is simple and flexible, allowing the user to combine file formats of any kind and scripts written in any command-line enabled scripting language. Script pipelines are, however, prone to disorganization, and large projects with many scripts can become unwieldy.

The Data Environment is a system for organizing, visualizing, managing, generating, and sharing script pipelines. The basic structure of the Data Environment is:

1) Dependency Notation: Users write python modules containing functions (or "operations") with simple notations that describe the data dependency links created by the operations.

2) Link Extraction: The user-written python modules are then analyzed by LinkManagement utilities that extract the links, and make them available for user exploration and for system automated updating.

3) Automated Updating: When data or functional dependencies change, automatic updating of files is implemented by Update utilities that look at the system links, and then decide which operations to call in which order.

4) MetaData: MetaData can be attached to particular files and is collected during automatic updating runs.

5) Protocols: Patterns of data analysis to apply repeatedly are implemented by protocols.

6) Graphical Browsing: The user can use a Graphical Browser to look at and explore the files, data dependencies, and metadata.

7) The DotData Format: A simple tabular data format that builds on the numpy record array class.

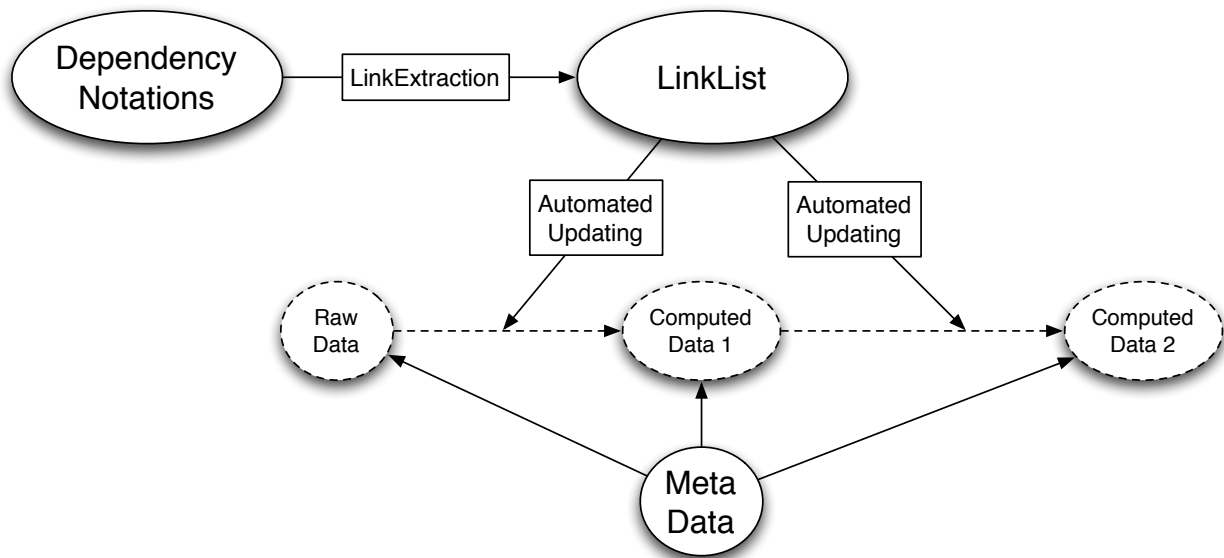


Figure 1: Data Flow Dotted lines represent data dependency links. Dotted ellipses are data files.

Below is a high-level description of each of these topics.

1. Dependency Notation:

The format is to use python keyword default variables to notate the dependencies. For each function that depends on source files to read, and creates files to write out:

- A "depends_on" keyword is used to describe read-only dependencies.
- A "creates" keyword is used to describe write-out creations.

See examples of this in, for example Operations/Dan_Operations/TestOrks/*.py

2. Link Extraction:

The main functions for this are in System/LinkManagement.py. They are:

1) **LoadLiveModules**: Makes a list of live modules in the Data Environment filesystem, to inspect for links. Determined by reading contents from user specified configuration file. (see comments in body of the code)

2) **LinksFromOperations**: Analyzes a set of python modules to find data dependency links present in the modules.

Typical Usage:

```
>> LinkList = LinksFromOperations(LoadLiveModules()) #... returns linklist in a numpy rec array format
```

3) **GetLinksBelow**: Loads the LinkList of live modules, and determines downstream data dependency links by propagating downstream through the LinkList, from paths that are in Seed.

Typical Usage: called by `System.Update.FindOutWhatWillUpdate` and `System.Update.FullUpdate`

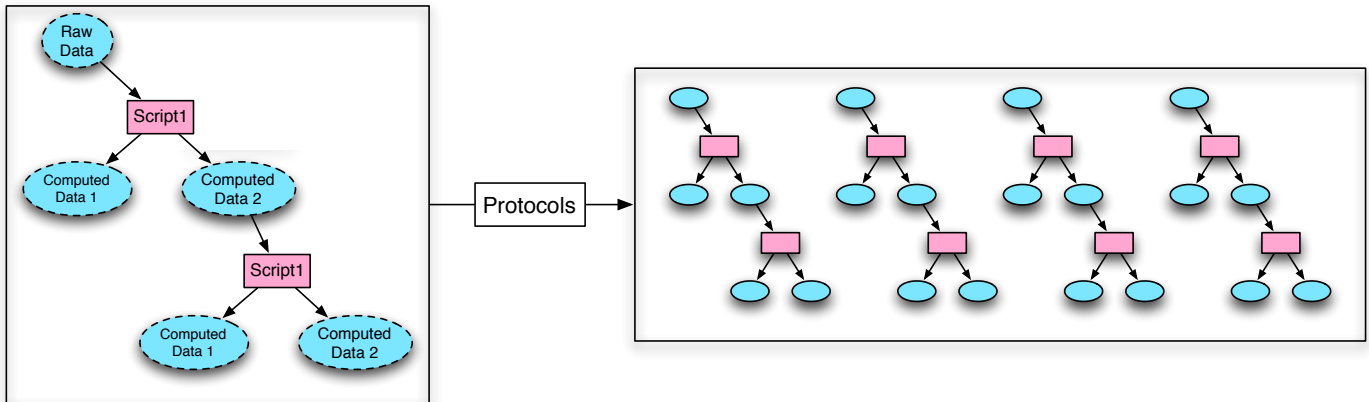


Figure 2: Protocols

3. Automated Updating:

The main functions for this are in System/Update.py. They are:

1) **FindOutWhatWillUpdate**: Determine and print out readable report indicating which files downstream of a seed will update (without making the actual update).

Typical Usage:

```
>> FindOutWhatWhatUpdate() #... then read screen print
```

2) **FullUpdate**: Actually makes updates, implements the downstream updating style.

Typical Usage:

```
>> FullUpdate() #... then see as as things get updated
```

3) **MakeUpdated**: Actually makes updates, implements the upstream updating style

Typical Usage:

```
>> MakeUpdated(Targets) #.... the see as things upstream of Targets get updated
```

4. MetaData:

The main functions for this are in System/MetaData.py. They are:

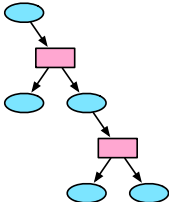
1) **AttachMetaData**: Attach metadata to a file a given path.

Typical Usage:

```
>> AttachMetaData(FileName = '../Data/Dan_Data/Bork.txt')
```

	DotData Format	

Figure 3: a. DotData Format

Graphical Browser															
		<table><tr><th>Col1</th><th>Col2</th><th>Col3</th></tr><tr><td>3</td><td>4.4</td><td>AZT</td></tr><tr><td>3</td><td>4.4</td><td>AZT</td></tr><tr><td>3</td><td>4.4</td><td>AZT</td></tr></table>		Col1	Col2	Col3	3	4.4	AZT	3	4.4	AZT	3	4.4	AZT
		Col1	Col2	Col3											
		3	4.4	AZT											
		3	4.4	AZT											
3	4.4	AZT													

b. Graphical Browser

2) `MakeRuntimeMetaData`: This is an internal usage function that attaches the results of metadata generated during the running of a system update.

Typical Usage: Used by `System.Update.UpdateLinks` during update runs.

5. Protocols

The main function for this is in `System/Protocols.py`. It is:

1) `ApplyOperations2`: Function which implements the basic protocol concept. Given an 'OpThing', which is a set of descriptions of operations and corresponding arguments to be passed to the operations, this writes out a python module with functions making the calls.

6. Graphical Browning:

In a web browser, navigate to http://DataEnvironment/System/CGI-Executables/main_gui.cgi to go the main page of the browser.

The main functions implementing this are in:

- `System/CGI-Executables/*`: Directory containing cgi scripts that implement the graphical browser,
- `System/SystemGraphOperations.py`: Functions that read information from `LinkLists` (as produced by `LinksFromOperations`) and creating graphical representation of data dependency links.

7. DotData:

A simple tabular data format that builds on the numpy record array class

The Main class is `Classes.DotData.DotData`. The usage is:

`X = DotData(Path = '../data/path')` to load from disk

`X = DotData(Columns =, Records = ...)` to load from live python objects

More detailed information can be found in several places, including:

System/Docs/DocumentationHtml/index.html

System/Docs/DocumentationPdf/api.pdf --

System/Docs/Install.txt -- a description of how to install the system

System/Docs/quickstart.txt -- a simple quick-start guide