# A Method for Creating and Maintaining an Inventory of an Institution's Scientific Data

- The problem:
  - Scientific institutions archive massive amounts of data, much of it only known about by a single project.
- The goal:
  - Make all the archived data discoverable and accessible by everyone at the institution.

# The Parts of our Method and the Tools They Use

- DataIngest/data_inventory
  - Accesses the data archives and populates the inventory database
  - Uses: LAMP (Linux, Apache, MySQL and PHP)
- The Data Inventory restful web API
  - Provides a restful application interface into the inventory DB
  - Uses: LAMP and the “slim” rest framework
- The search CIRA Data web application
  - Uses the the data inventory web API to search the inventory for specific data files
  - Uses: PHP, jQuery(javascript) and AJAX

# The Development Tools

- Zend Studio
  - Eclipse with the PHP Development Tools plugin
  - Includes a javaScript editor, PHPunit and a PHP debugger
- Subversion for revision control
- phpMyAdmin: A web GUI for MySQL Administration

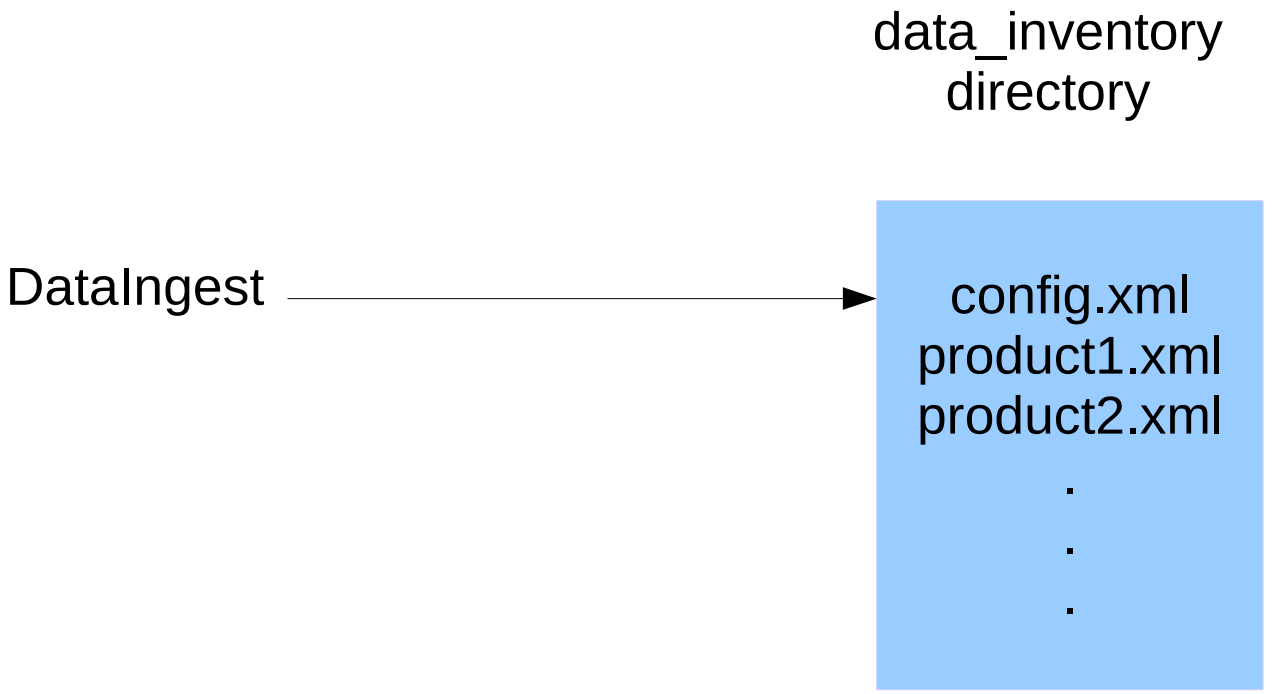
# DataIngest: the first part of DataIngest/data_inventory

- Purpose: To ingest the files from any data collection or analysis project and to add the files' metadata to any database
- Loads metadata obtained from a client project's path names into a database
- Optionally moves the files to specified directories
- Highly configurable: Uses configuration files to specify how the data files' metadata is obtained, and how to add it to a DB
- Caveat: All the metadata needed for the DB must be available from the file path names

# Data_inventory: the second part of DataIngest/data_inventory

- The DataIngest configuration environment for populating the data inventory database
- Just a directory that contains configuration files
  - It contains the database configuration file – config.xml and the product configuration files
  - The DataIngest executables run in this directory

# DataIngest/data_inventory



# The data_ingest DB Fields

collections:
- collection name
- public/private flag
- notes

products:
- product name
- description

product_versions:
- product version
  - (e.g. 006 or P2_R04)
- public/private flag
- notes

files:
- file name
- file location
- start time
- end time
- file size
- time inventoried

# A Product Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>

<product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="product.xsd">
   <!-- TMI.3B.GPR10.V1.20130101.BIN.gz -->
   <codedname><![CDATA[TMI.3B.GPR10.<c*(version)>.<0i6><0d>.BIN.gz]]></codedname>
   <full_ingest_day_of_month>
      23
   </full_ingest_day_of_month>
   <input_dirs>
      <from><![CDATA[ftp://rain.atmos.colostate.edu/RAINMAP10v2/data/tmi/<Y>/<0i2><0M>]]></from>
   </input_dirs>
   <product_fields> <!-- valid for this product, same for every file -->
      <field name="public">1</field> <!-- Yes, public -->
      <field name="cnotes">TRMM does not have collections in the way CloudSat does, so one is created for all of TRMM.</field>
      <field name="vnotes">See http://rain.atmos.colostate.edu/RAINMAP10</field>
      <field name="collection">TRMM</field>
      <field name="prodname">TMI Rainfall</field>
      <field name="proddescr">TMI Rainfall</field>
   </product_fields>
   <file_fields> <!-- different for every file. Product specific - include constant values for the product -->
      <field name="prodversion">[version]</field>
      <field name="end_datetime" modify="add 86400">[Y]-[0M]-[0d] [0H]:[0m]:[0s]</field>
   </file_fields>
</product>
```

# A Coded File Name

```
<product>
   <!-- TMI.3B.GPR10.V1.20130101.BIN.gz -->
   <codedname>
      <![CDATA[TMI.3B.GPR10.<c*(version)>.<0i6><0d>.BIN.gz]]>
   </codedname>
   .
   .
   .
</product>
```

# A Coded Input Directory

```
<product>
   .
   .
   .
   <input_dirs>
      <from>
         <!-- ftp://rain.atmos.colostate.edu/RAINMAP10v2/data/tmi/2013/1309 -->
         <![CDATA[ftp://rain.atmos.colostate.edu/RAINMAP10v2/data/tmi/<Y>/<0i2><0M>]]>
      </from>
   </input_dirs>
   .
   .
   .
</product>
```

# The Product Fields

```
<product>
   .
   .
   .
   <product_fields> <!-- valid for this product, same for every file -->
      <field name="public">1</field> <!-- Yes, public -->
      <field name="collection">TRMM</field>
      <field name="prodname">TMI Rainfall</field>
      <field name="proddescr">TMI Rainfall</field>
   </product_fields>
   .
   .
   .
</product>
```

# The File Fields

```
<product>
    .
    .
    .
    <file_fields> <!-- different for every file. -->
        <field name="prodversion">[version]</field>
        <field name="end_datetime" modify="add 86400">
            [Y]-[0M]-[0d] [0H]:[0m]:[0s]
        </field>
    </file_fields>
</product>
```

# The Database Configuration File

- The database name
- The database credentials
- The list of common DB field names
  - Specific to the target DB and are the same for every file
- The list of file fields
  - DB specific and different for every file
- The DB querys that specify, in sql, how metadata are added to the DB

# The DB File Fields

```
<config>
   <database name="data_inventory">
      .
      .
      .
      <file_fields> <!-- different for every file. DB specific only.
                           The same for every product -->
         <field name="productid" modify="sql" noresult="insert_product">
            select productid from products where products.prodname='{prodname}'
         </field>
         <field name="filename">{filename}</field>
         <field name="start_datetime">[Y]-[0M]-[0d] [0H]:[0m]:[0s]</field>
      </file_fields>
      .
      .
      .
   </database>
</config>
```

# The DB Query Fields

```
.
.
.
<queries>
   <query tag="insert_product">
      <sql result="productid">
         insert into products set prodname='{prodname}', proddescr='{proddescr}'
      </sql>
   </query>
   <query tag="insertfile">
      <sql result="fileid">
         insert into files set filename='{filename}',
           product_versionid='{product_versionid}', filelocation='{filelocation}',
           start_time='{start_datetime}', end_time='{end_datetime}',
           volume='{volume}', creation_time='{creation_time}'
      </sql>
   </query>
</queries>
.
.
.
```

# The Data Inventory Restful WEB API

- Provides a restful application interface into the data_inventory DB
- Written in PHP
- Uses the “slim” framework
  - Slim is a PHP micro framework that helps you quickly write simple yet powerful web applications and APIs - http://www.slimframework.com
- Based on Brian Mulloy's web book
  - http://info.apigee.com/Portals/62317/docs/web%20api.pdf

# Data Inventory API Examples

- Products
  - List all: GET /products
  - Get a product: GET /products/{productId}
- Files
  - List the files for a product:
    - GET /product/{productId}/files
  - Get file details:
    - GET /files/{fileId}

# The search CIRA Data web application

- Uses the the data inventory web API to search the inventory for specific data files
- Dynamically displays hierarchal lists of the collections, products and files
- Allows the user to select a subset of the items at each level
  - Uses: PHP, jQuery(javascript) and AJAX

# The search CIRA Data app tools

- PHP for the server side initialization
- jQuery(javascript) for most of the UI
- AJAX to make the data_inventory API calls

# Summary

- Successfully inventoried, and continuing to inventory, a significant amount of the satellite data archived at CIRA
  - Will inventory more data sets and more types of data
- Implemented a partial restful API for accessing the inventory
  - Will implement additional resources and search parameters
  - Will make the API accessible to all of CIRA
- Implemented a prototype web app that uses the API and allows a user to search the inventory
  - We might add features and make it more usable
  - We will provide documentation and training to help other CIRA developers write their own applications