

iPlant Collaborative Foundation API Tutorial



What is an API?

Application programming interface

From Wikipedia, the free encyclopedia

"API" redirects here. For other uses, see [API \(disambiguation\)](#).

An **application programming interface (API)** is a specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for [routines](#), [data structures](#), [object classes](#), and variables.

APIs provide:

- Utility
- Abstraction
- Interoperability
- Documentation

iPlant APIs are of a class known as Web APIs

- This doesn't mean you only do web things with them
- It means you interact with them via web actions



What is an API?

Web APIs

[edit]

Main article: [web service](#)

When used in the context of [web development](#), an API is typically defined as a set of Hypertext Transfer Protocol ([HTTP](#)) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language ([XML](#)) or [JavaScript Object Notation \(JSON\)](#) format. While "Web API" is virtually a synonym for [web service](#), the recent trend (so-called [Web 2.0](#)) has been moving away from Simple Object Access Protocol ([SOAP](#)) based services towards more direct [Representational State Transfer \(REST\)](#) style communications.^[5] Web APIs allow the combination of multiple services into new applications known as [mashups](#).^[6]

If the Discovery Environment is “User Interface,”
the Foundation API is “Machine Interface”



REST (REpresentational State Transfer)

1. REST uses URIs to refer to and to access resources.
2. REST is built on top of the stateless HTTP 1.1 protocol.
3. REST uses HTTP commands to define operations.

This last point is essential in REST architecture. HTTP commands have precise semantics:

1. **GET** lists or retrieves a resource at a given URI.
2. **PUT** replaces or updates a resource at a given URI.
3. **POST** creates a resources at a given URI.
4. **DELETE** removes the resources at a given URI.



A REST Example (1)

I've built a library database in which I store information about my books, DVDs, music, etc. and exposed it as a RESTful service

`http://fonner.org/library`

Sub-URLs of this `/library` endpoint represent types of media

- `http://fonner.org/library/books`
- `http://fonner.org/library/dvds`
- `http://fonner.org/library/music`

I can add, update, remove, or fetch records from these via HTTP operations.



A REST Example (2)

Let's go about storing information on a **dvd** that my 3 year old son loves:



```
{"title": "Mary Poppins",  
 "genre": "Family",  
 "release-date": "August, 1964",  
 "notes": "Carson will sing Chim Chim  
 Cher-ee for a week after watching"}
```

I encode this data into an HTML form and POST it to

<http://fonner.org/library/dvds>

And then...



A REST Example (3)

I get back the following:

- 1) An HTTP status code of 200 OK, which means that the operation succeeded
- 2) A message body (in JSON format due to my preference)

```
{"id":101,  
  "date":"May 18, 2012",  
  "uri":"http://fonner.org/library/dvds/101"}
```

Now let's say I want to retrieve that new record because I have a short memory and can't recall what I just POSTed.



A REST Example (4)

I perform an HTTP GET operation on
<http://fonner.org/library/dvds/101>

I get back:

- 1) An HTTP status code of 200 OK, which means that the operation succeeded
- 2) A message body

```
{"id":101,  
  "date":"May 18, 2012",  
  "title":"Mary Poppins",  
  "genre":"Family",  
  "release-date":"August, 1964",  
  "notes":"Carson will sing Chim Chim Cher-ee for  
  a week after watching"}
```

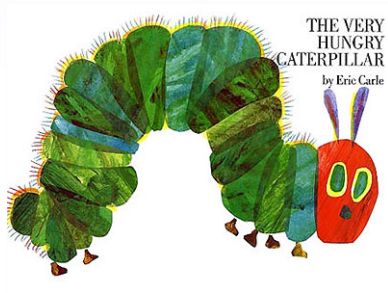


A REST Example (5)

To update the record, say to change the notes to something less silly, I send an updated version of the original data object via HTTP PUT to <http://fonner.org/library/dvds/101>

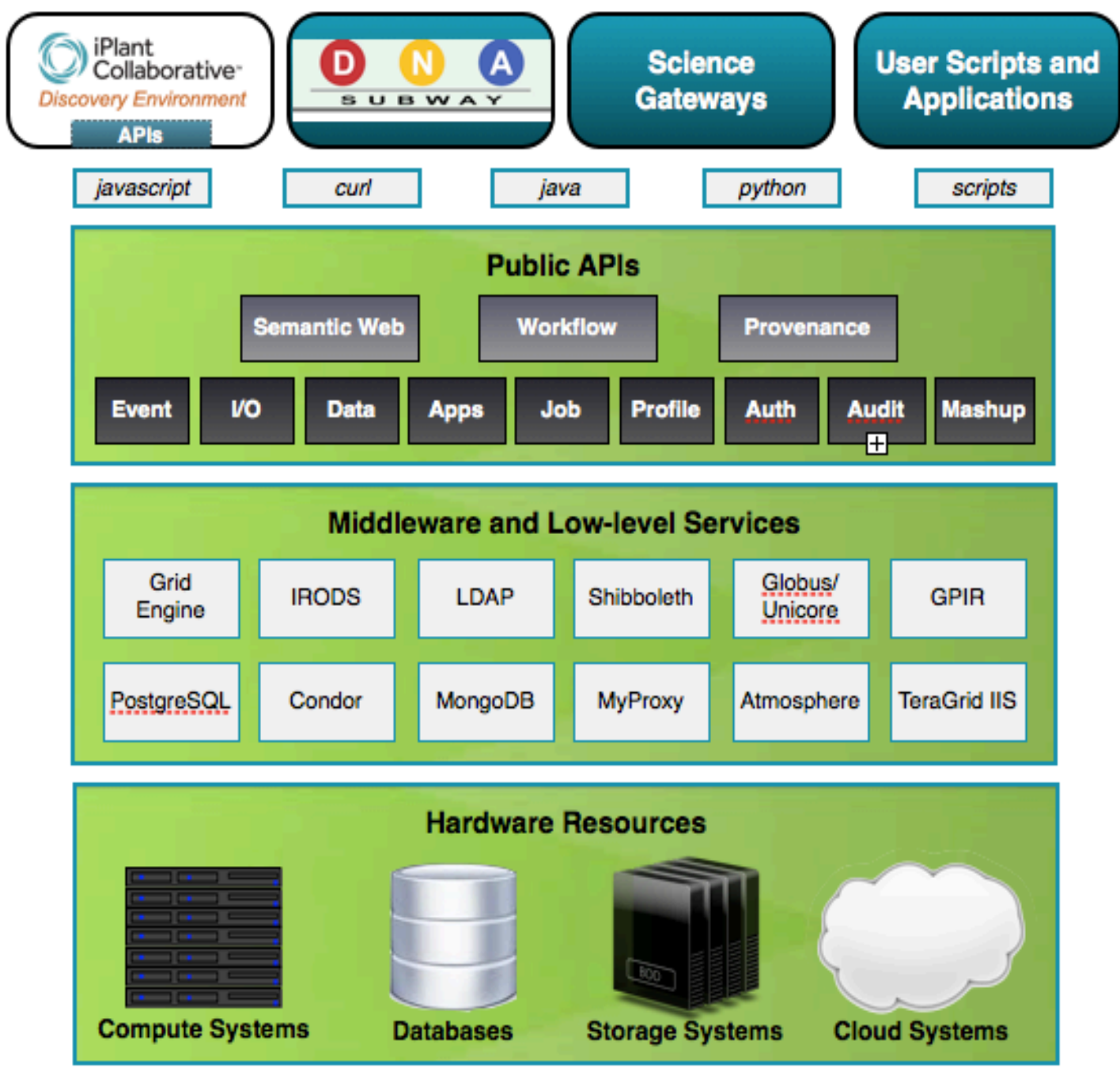
To delete the record entirely, I perform an HTTP DELETE on <http://fonner.org/library/dvds/101>

That's a nutshell example of a simple REST API



Importantly, the paradigm can be extended to accommodate more than just database records! We can move beyond documenting a children's book collection to driving a national-scale cyberinfrastructure.





Language/OS-independent application development

Complexity is abstracted behind API layer

Reuse and integration of Open Source Middleware

National-scale physical resources



Foundation API Overview

Endpoint	Activities
IO	File data storage/retrieval/management. Database interoperability. File metadata management
DATA	Transparent file-format conversion
PROFILE	User information discovery and management
APPS	Registration and discovery of public or private HPC applications
JOB	Submission and management of computing jobs on XSEDE systems
AUTH	Token-based highly secure authentication with proxy capability
SYSTEMS	Returns availability and other information about XSEDE JOB hosts
POSTIT	A URL –shortener and limited-used URL generator



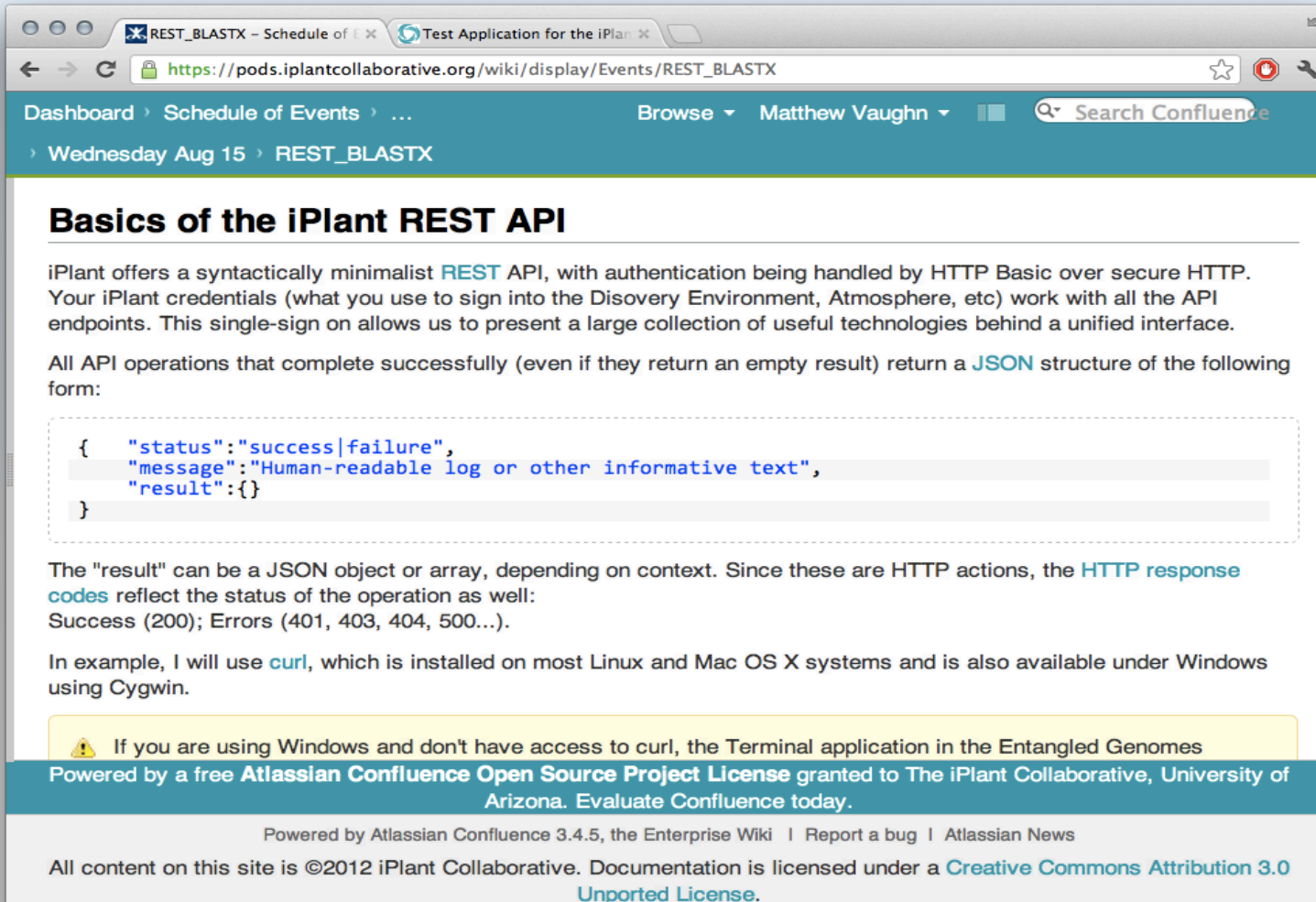
Foundation API Overview

Endpoint	Activities
IO	File data storage/retrieval/management. Database interoperability. File metadata management
DATA	Transparent file-format conversion
PROFILE	User information discovery and management
APPS	Registration and discovery of public or private HPC applications
JOB	Submission and management of computing jobs on XSEDE systems
AUTH	Token-based highly secure authentication with proxy capability
SYSTEMS	Returns availability and other information about XSEDE JOB hosts
POSTIT	A URL –shortener and limited-used URL generator

We'll focus on using these four today.



Real World Example: BLASTX



The screenshot shows a web browser window displaying a Confluence wiki page. The browser's address bar shows the URL https://pods.iplantcollaborative.org/wiki/display/Events/REST_BLASTX. The page title is "Basics of the iPlant REST API". The content explains that iPlant offers a syntactically minimalist REST API with authentication handled by HTTP Basic over secure HTTP. It notes that all API operations return a JSON structure. A code block shows a JSON object with fields for status, message, and result. The text also mentions that the result can be a JSON object or array and that HTTP response codes reflect the status of the operation. A note at the bottom of the page suggests using curl on Linux and Mac OS X, and provides a warning for Windows users. The footer of the page includes information about the Atlassian Confluence license and copyright.

Dashboard > Schedule of Events > ... Browse > Matthew Vaughn > Search Confluence

> Wednesday Aug 15 > REST_BLASTX

Basics of the iPlant REST API

iPlant offers a syntactically minimalist **REST** API, with authentication being handled by HTTP Basic over secure HTTP. Your iPlant credentials (what you use to sign into the Discovery Environment, Atmosphere, etc) work with all the API endpoints. This single-sign on allows us to present a large collection of useful technologies behind a unified interface.

All API operations that complete successfully (even if they return an empty result) return a **JSON** structure of the following form:

```
{  "status": "success|failure",  "message": "Human-readable log or other informative text",  "result": {}}
```

The "result" can be a JSON object or array, depending on context. Since these are HTTP actions, the **HTTP response codes** reflect the status of the operation as well: Success (200); Errors (401, 403, 404, 500...).

In example, I will use **curl**, which is installed on most Linux and Mac OS X systems and is also available under Windows using Cygwin.

⚠️ If you are using Windows and don't have access to curl, the Terminal application in the Entangled Genomes

Powered by a free **Atlassian Confluence Open Source Project License** granted to The iPlant Collaborative, University of Arizona. Evaluate Confluence today.

Powered by Atlassian Confluence 3.4.5, the Enterprise Wiki | Report a bug | Atlassian News

All content on this site is ©2012 iPlant Collaborative. Documentation is licensed under a **Creative Commons Attribution 3.0 Unported License**.

https://pods.iplantcollaborative.org/wiki/display/Events/REST_BLASTX



The apigee API console

The screenshot shows a web browser window with the URL <https://foundation.iplantc.org>. The page header features the iPlant Collaborative logo and the tagline "Empowering A New Plant Biology". Below the header, the main title reads "Developer Console for the iPlant Foundation API".

The interface includes a "Query URL" input field with a lock icon on the left and a "GET" button on the right. A sidebar on the left lists various API resources: Apps, Auth, Data, IO, Jobs, PostIt, and Profile. The main content area is divided into "Request" and "Response" tabs. A central tip box provides instructions for making an API request:

- TIP** To make an **API Request**, try the following:
 - Select a method from the resource list *and* click the verb button.
 - Enter an API request in the text field *and* click the verb button.

The footer contains copyright information: ©2010 iPlant Collaborative, links to Privacy Statement and RSS Feed, and a funding acknowledgment: The iPlant Collaborative is funded by a grant from the National Science Foundation Plant Cyberinfrastructure Program (#EF-0735191).

Powered by **apigee**



Working with cURL

The screenshot displays a Mac desktop environment. In the background, a terminal window shows the following command and output:

```
tolva:~ mwvaughn$ curl -X GET -sku "vaughn:60321bcfc47b1fde424d8b986c1eed54" https://foundation.iplantc.org/io-v1/io/vaughn/analyses | python -mjson.tool
{
  "message": "Directory downloads not supported",
  "result": null,
  "status": "error"
}
tolva:~ mwvaughn$ curl -X GET -sku "vaughn:60321bcfc47b1fde424d8b986c1eed54" https://foundation.iplantc.org/io-v1/io/list/vaughn/analyses | python -mjson.tool
{
  "message": "",
  "result": [
    {
      "format": "folder",
      "lastModified": 1327499821000,
      "length": 0,
      "mimeType": "",
      "name": "..",
      "owner": "vaughn",
      "path": "/vaughn/analyses",
      "permissions": "OWN",
      "type": "dir"
    },
    {
```

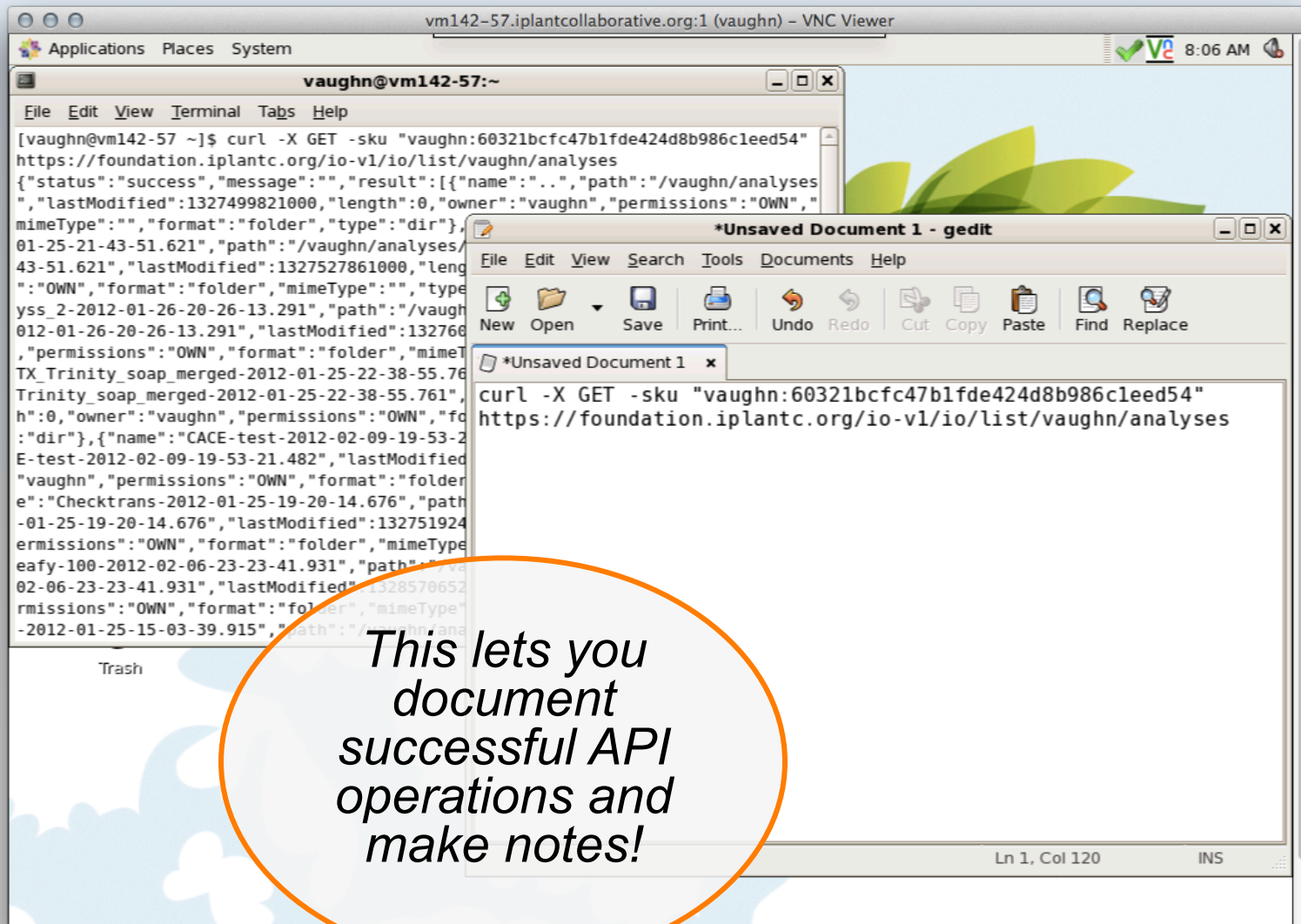
In the foreground, a text editor window titled "Untitled" contains the following command:

```
curl -X GET -sku "vaughn:60321bcfc47b1fde424d8b986c1eed54" https://foundation.iplantc.org/io-v1/io/list/vaughn/analyses | python -mjson.tool|
```

An orange oval highlights the text in the text editor with the instruction: *Type up command line into a text editor, then copy and paste into the terminal emulator*



Working with cURL



The screenshot shows a VNC viewer window titled "vm142-57.iplantcollaborative.org:1 (vaughn) - VNC Viewer". Inside, there is a terminal window and a gedit editor window. The terminal window shows a successful cURL command and its output. The gedit window has the same command pasted into it. An orange circle highlights the gedit window with the text "This lets you document successful API operations and make notes!".

```
[vaughn@vm142-57 ~]$ curl -X GET -sku "vaughn:60321bcfc47b1fde424d8b986c1eed54" https://foundation.iplantc.org/io-v1/io/list/vaughn/analyses
```

```
{ "status": "success", "message": "", "result": [{"name": "..", "path": "/vaughn/analyses", "lastModified": 1327499821000, "length": 0, "owner": "vaughn", "permissions": "OWN", "mimeType": "", "format": "folder", "type": "dir"}, {"name": "01-25-21-43-51.621", "path": "/vaughn/analyses/01-25-21-43-51.621", "lastModified": 1327527861000, "length": 0, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "yys_2-2012-01-26-20-26-13.291", "path": "/vaughn/analyses/yys_2-2012-01-26-20-26-13.291", "lastModified": 132760012-01-26-20-26-13.291, "lastModified": 132760012, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "TX_Trinity_soap_merged-2012-01-25-22-38-55.761", "path": "/vaughn/analyses/TX_Trinity_soap_merged-2012-01-25-22-38-55.761", "lastModified": 132759711-01-25-22-38-55.761, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "Trinity_soap_merged-2012-01-25-22-38-55.761", "path": "/vaughn/analyses/Trinity_soap_merged-2012-01-25-22-38-55.761", "lastModified": 132759711-01-25-22-38-55.761, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "CACE-test-2012-02-09-19-53-21.482", "path": "/vaughn/analyses/CACE-test-2012-02-09-19-53-21.482", "lastModified": 132759711-02-09-19-53-21.482, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "E-test-2012-02-09-19-53-21.482", "path": "/vaughn/analyses/E-test-2012-02-09-19-53-21.482", "lastModified": 132759711-02-09-19-53-21.482, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "Checktrans-2012-01-25-19-20-14.676", "path": "/vaughn/analyses/Checktrans-2012-01-25-19-20-14.676", "lastModified": 132759711-01-25-19-20-14.676, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "01-25-19-20-14.676", "path": "/vaughn/analyses/01-25-19-20-14.676", "lastModified": 132759711-01-25-19-20-14.676, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "ermissions", "path": "/vaughn/analyses/ermissions", "lastModified": 132759711-01-25-19-20-14.676, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "eafy-100-2012-02-06-23-23-41.931", "path": "/vaughn/analyses/eafy-100-2012-02-06-23-23-41.931", "lastModified": 132759711-02-06-23-23-41.931, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "02-06-23-23-41.931", "path": "/vaughn/analyses/02-06-23-23-41.931", "lastModified": 132759711-02-06-23-23-41.931, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "rmissions", "path": "/vaughn/analyses/rmissions", "lastModified": 132759711-02-06-23-23-41.931, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}, {"name": "-2012-01-25-15-03-39.915", "path": "/vaughn/analyses/-2012-01-25-15-03-39.915", "lastModified": 132759711-01-25-15-03-39.915, "lastModified": 132759711, "owner": "vaughn", "permissions": "OWN", "format": "folder", "mimeType": "", "type": "dir"}]}
```

This lets you document successful API operations and make notes!



Interactive Session



Additional topics

- Querying and interpreting the APPS catalog
- Callback URLs
- Using POSTIT effectively
- Deploying your own application in the APPS catalog
- Writing a simple HTML/JS application to interact with the iPlant API

