European Persistent Identifier Consortium (EPIC) API v1

(PID: http://hdl.handle.net/11858/00-ZZZZ-0000-0001-6D1A-5 )

von

Tibor Kálmán
tibor [dot] kalman [at] gwdg [dot] de

CLARIN-D (08.09.2011)

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Am Fassberg, 37077 Göttingen

Fon: 0551 201-1510  Fax: 0551 201-2150

gwdg@gwdg.de  www.gwdg.de
Agenda

- Persistent Identifiers in the eResearch
- Consortium for PIDs
- EPIC PID-Service for the Research
  - RESTful Web Service
  - API of the PID-Service
- Examples
  - Python, Java, Shell
- Outlook
- Summary
Motivation (1)

- The amount of stored digital data grows rapidly in all areas of science
- Access and long-time preservation of the digital objects are important issues
- Standardized interface to the storage and repository systems
  - Interface is independent from the underlying infrastructure
    - HTTP, WebDAV are very common
    - It enables access to the stored digital objects with standard tools like web- or filesystem-browsers
  - Changes in the storage and repository systems are possible “at any time” and “without any problem” (at least in theory)
Standard Interfaces to Digital Objects

- Research data is accessible using standard interfaces
- Currently the “physical” address of a digital object is often used for referencing and citation

http://providerX/serverX/pathX/fileX

Middleware

Grid or Cloud technology

Standardized Interface

Storage

Middleware

Storage

Middleware

Storage

Middleware

Storage
Reference Based on Physical Location

- After changes in the location the objects has to be referenced with a new physical address

http://providerX/serviceX/pathX/fileX

http://providerZ/serverZ/pathZ/fileZ

Standardized Interface

Grid or Cloud technology

Middleware

Middleware

Middleware

Middleware

Storage

Storage

Storage

Storage
Motivation (2)

- The amount of stored scientific data grows, but how to identify them?
  - More and more relations between these data and other resources become essential for science as for instance references to scientific publications
  - URIs are currently often used for referencing and citation

- Uniform Resource Identifier (URI): the “physical” address of a digital object
  - The URI contains physical pathes and semantical contents
  - The address is often not persistent (migration, etc)
  - After changes the objects has to be referenced with a new URI
  - URIs are frequently outdated (after couple of years)

- URIs are not suitable for referencing digital objects!
Referencing with Persistent Identifier

http://pidservice/identifier1

http://providerX/serviceX/pathX/fileX

Persistent Identifier Service

PID Database (PID → URL)

Standardized Interface

Grid or Cloud technology

Middleware

Middleware

Middleware

Middleware

Storage

Storage

Storage

Storage
Persistent Identifier (PID)

Scientific institutes need a strategy for the long term preservation and long-lasting accessibility of scientific resources:

- **Persistent IDentifier (PID)**
  - Digital Objects are registered in well-kept repositories
  - With a content (reference), that is never changing (PID)
  - The underlying systems can be changed ("living organisms")
  - A migration is possible on various levels (changes in hardware, software, format, etc)

- For the allocation, management and resolution of PIDs:
  - One needs a commonly agreed process
  - A high degree of robustness and reliability in the long-term: Handle system, like Domain Name System (DNS)

It is possible to reference and cite the digital objects by PIDs!
Management and Resolution

- Allocation, Management and Resolution of a persistent identifier (a DOI example):
  - http://dx.doi.org/10.1007/s10723-009-9134-3
    - http://www.springerlink.com/content/wm45432131n4v6g8/
    - http://pubman.mpdl.mpg.de/pubman/item/escidoc:218009:2

- Identifiers
  - eDoc: 442632
    - http://edoc.mpg.de/442632
  - ISI: 000272244900005
    - http://gateway.isiknowledge.com/...
  - ISSN: 1570-7873
  - DOI: 10.1007/s10723-009-9134-3
    - http://dx.doi.org/10.1007/s10723-009-9134-3
GWDG runs a PID Service (on behalf of the Max Planck Society)

Based on the Handle System (http://www.handle.net/)

Goal: Allocation, Management and Resolution of identifiers for research data (scientific digital objects)

Together with other european partners a consortium was build to provide this services to the european research community

- European Persistent Identifier Consortium (EPIC)
- http://www.pidconsortium.eu/
Agenda

- Persistent Identifiers in the eResearch
- Consortium for PIDs
- EPIC PID-Service for the Research
  - RESTful Web Service
  - API of the PID-Service
- Examples
  - Python, Java, Shell
- Outlook
- Summary
Consortium for a PID System

- European Persistent Identifier Consortium (EPIC)
- is dedicated to providing a **Persistent IDentifier** (PID) service
- main scope is European scientific and cultural heritage communities
- is a consortium of three major European scientific computing centers
  - with solid backing of national funding authorities
  - with long experience in providing reliable, safe and secure services and technical sustainability
  - with a company-like structure
  - with the possibility to provide SLAs
  - are involved in several big eScience projects
  - have signed a MoU to provide a PID system for the scientific community
Members of EPIC: GWDG (1)

- Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG)
- GWDG is a corporate facility of the Max-Planck-Society and the Georg-August University of Göttingen
- It operates for both as a computer center (for the MPG it is furthermore IT competence center)
- GWDG was founded in 1970 as a company (operates on a non-profit principle)
- It is located in Göttingen
- 25,000 users
- 1000 scientific HPC users
- Staff: about 80 employees
Members of EPIC: GWDG (2)

- Main tasks of the GWDG:
  - high performance computing
  - high performance networking
  - infrastructure services
  - IT consulting

- Partner in several eScience & grid projects:
  - DARIAH-DE, CLARIN-D, D-Grid DGSI

- Project leader of the research projects:
  - Instant-Grid, OptiNum-Grid

- Other projects: GoeGrid, Kopal, etc
Members of EPIC: SARA

- Stichting Academisch Rekencentrum Amsterdam (SARA)
- SARA Computing and Networking Services is an advanced ICT service center
- SARA supports researchers in the Netherlands and works closely together with the academic community, government institutes and industry
- It supplies – since more than 30 years – a complete package of
  - high performance computing and
  - Visualization
  - high performance networking and
  - infrastructure services
- SARA is located in Amsterdam
Members of EPIC: CSC

- IT Center for Science Ltd (CSC)
- CSC, as part of the Finnish national research structure, develops and offers high-quality information technology services
- It provides Finland's most powerful supercomputing environment
- CSC was founded in 1970, reorganized as a company in 1993 (operates on a non-profit principle)
- Facilities in Espoo, close to Otanie campus of Helsinki University
- Staff 180
- 3000 researchers use CSC’s computing capacity
The Most Active User Communities of EPIC

- MPG, Max Planck Society
- CLARIN, Common Language Resources and Technology Infrastructure
- DARIAH-DE, Digital Research Infrastructure for the Arts and Humanities
- SUB, Niedersächsische Staats- und Universitätsbibliothek Göttingen
- DKRZ, German Climate Computing Center
Syntax of the Handles issued by the GWDG (1)

- For the resolution of PIDs one needs a commonly agreed process
  - worldwide handle (PID) framework
- GWDG uses the prefix the number 11858
  - http://handle.gwdg.de:8000/
  - This is integrated into the general and worldwide handle framework
  - The global handle service delegates all requests for resolution concerning prefix 11858 to the GWDG

Handle of this presentation:
- http://hdl.handle.net/11858/00-ZZZZ-0000-0001-6D1A-5
Syntax of the Handles issued by the GWDG (2)

- 11858/00-XXXX-0000-0000-0000-C
- prfix fg-inst-num1-num2-num3-c

The meaning of these fields above is:

- **prfix** is the handle prefix, which is fixed to 11858
- **fg** is a uppercase hexadecimal flag, that can be used for special purposes, to be defined later(derived handles etc)
- **inst** is a field with alphanumerical uppercase digits and describes the institution responsible for registration of the handle,
- **num1-num2-num3** are 12 bytes, coded in uppercase hexadecimal digits with delimiters
- **c** is a checksum to ensure plausibility of the handle string.
Interfaces for Assigning, Managing, Resolving EPIC Handles

- Assigning, Managing, Resolving of PIDs

- Interfaces:
  - (Native) Handle Interface
    - [RFC3652] Handle System Protocol
    - [RFC3651] Handle System Namespace and Service Definition
  - Webinterface:
    - REST-basiertes Web Services Interface
    - http://handle.gwdg.de:8080/pidservice/
Agenda

- Persistent Identifiers in the eResearch
- Consortium for PIDs
- EPIC PID-Service for the Research
  - RESTful Web Service
  - API of the PID-Service
- Examples
  - Python, Java, Shell
- Outlook
- Summary
The EPIC PID-Service

- Access control, right management:
  - **Searching, resolution** of PIDs (=Handles): *free access*
  - **Allocation** of new PIDs and **management** of existing PIDs: *Authentication and Authorisation needed*
    - Basic auth: username + password (like Web)

- User database:
  - User information
  - Roles
RESTful Web Services

- Representational State Transfer (REST) was first introduced by Roy Fielding
  - co-founder of the Apache HTTP Server project, was the chair of the Apache Software Foundation, member of the interim OpenSolaris Boards, involved in the development of HTML and Uniform Resource Identifiers

  - describes REST as a key architectural principle of the World Wide Web
  - analyzes a set of software architecture principles that use the Web as a platform for distributed computing

- REST-Example: document updates by email?
REST Design Principles

Basic design principles of REST:

1) Use standard HTTP methods
2) Be stateless
3) URIs should be intuitive
4) Client chooses the data format (XML, JSON)
The basic functions of persistent storage:
create, read, update, delete (CRUD)

- To create a PID (=resource) on the server
  - use POST

- To resolve a PID (=retrieve or read a resource)
  - use GET

- To modify a PID (=change the state of a resource or to update it)
  - use PUT

- To remove a PID (=delete the resource)
  - use DELETE
Design: (1) Standard HTTP Methods

EPIC PID-Service at the GWDG:

- To create a PID (=resource) on the server
  - use POST
- To resolve a PID (=retrieve a resource)
  - use GET
- To modify a PID (=change the state of a resource or to update it)
  - use PUT
- To remove a PID (=delete the resource)
  - use DELETE

PID is persistent: cannot be removed!
Design: (2) Be Stateless

- Services need to scale to meet increasing performance demands
- To decrease the overall response time of a Web service
  - Special topology of servers: server clusters or infrastructures for load-balancing and failover
  - Requests can be forwarded from one server to the other
    - no state or context held locally → simplified design and implementation
  - Web service clients have to send complete, independent requests
    - requests must include all data (parameters, context) needed by the server-side component to generate a response
    - within the HTTP headers and body of a request
Design: (2) Be Stateless

Request to the PID-Service must include all data!

Source: www.ibm.com
Design: (3) URIs should be intuitive

- URI: addressing resources
  - describes the location of something anywhere in the world from anywhere in the world
- URI should require little, if any, explanation to understand what it points to
- URI should be straightforward, predictable, and easily understood
- One way to achieve this usability is to define URIs like a directory structure
Design: (3) URIs should be intuitive

- **World wide Handle system:**
  - http://hdl.handle.net/11858/00-ZZZZ-0000-0001-6D1A-5

- **PID-Service root**
  - http://handle.gwdg.de:8080/pidservice/

- **URIs like a directory structure:**
  - http://handle.gwdg.de:8080/pidservice/write/create
  - http://handle.gwdg.de:8080/pidservice/write/modify
  - http://handle.gwdg.de:8080/pidservice/read/view
  - http://.../pidservice/read/view/pid/11858%2F00-ZZZZ-0000-0001-6D1A-5

- Instead of this, use parameter 'pid=':
  - http://handle.gwdg.de:8080/pidservice/read/view?pid=11858%2F00-ZZZZ-0000-0001-6D1A-5
Design: (4) Client Chooses the Data Format

- This allows a variety of clients written in different languages running on different platforms and devices.
- Using MIME types and the HTTP Accept header (content negotiation) lets clients choose which data format is right for them.
- Minimizes data coupling between the service and the applications that use the service.
- XML or JavaScript Object Notation (JSON) or HTML.
Design: (4) Client Chooses the Data Format

PID-Service:
- Request: with content negotiation, Response: HTML or XML
- Parameter in HTTP request: `encoding='xml'`

```xml
<pidserviceresponse>
  <action>update</action>
  <message>TITLE modified. AUTHORS modified. </message>
  <Handle>
    <pid>11858/00-ZZZZ-0000-0000-0229-F</pid>
    <url>
      http://www.swe.informatik.uni-goettingen.de/edu/notes/index.php?vorl_nr=78
    </url>
    <title>
      Praktikum: Anwendung und Programmierung im Grid (SS 2011)
    </title>
    <authors>
      Jens Grabowski, Arnulf Quadt, Oswald Haan, Heike Neuroth, Thomas Rings, Jörg Meyer, Tibor Kalman, Patrick Harms
    </authors>
  </Handle>
</pidserviceresponse>
```
The EPIC Pid-Service Implementation

- Implemented in Java
- Runs in a Tomcat service container
- Uses a PostgreSQL database

- Screenshots, parameters on the next slides
View (read) a PID

GET PIDSERVICE/read/view:
- to view a Handle
  - locally,
  - via a Handle System web proxy,
  - or to jump to its URL
- Parameter: pid
- Optional:
  - proxyview=yes
  - redirect=yes
  - metadata=yes,
  - showmenu=yes (adds buttons),
  - encoding=xml
Find a PID

**GET** PIDSERVICE/read/search:
- to find all Handles where \( ? = ? \)
- Parameter: at least one of:
  - pid, checksum, size, title, authors, pubdate, expdate, metadata_url, creator
- Optional:
  - showmenu=yes
  - encoding=xml
- Returns HTTP status code and HTML or XML content
View (read) a PID (1)

View Handle

Contents of ZZZZ Handle '11858/00-ZZZZ-0000-0000-000C-7'

Handle 11858/00-ZZZZ-0000-0000-000C-7
Location http://www.gwdg.de/aktuell/index4.html
Size 0
Checksum N/A
Published N/A
Title GWDG
Author(s) N/A
Metadata N/A
PID Creator demo2
PID Expiry N/A

Contact PID Creator at: Email: jon@doe.xx

User details

No active user!

[Back to input form]
View (read) a PID (2)

PID-Service:
- Request: XML, Response: XML
- Parameter in HTTP request: encoding='xml'

XML response:
```xml
<pidserviceresponse>
  <action>search</action>
  <matches>1</matches>
  <message>1 found</message>
  <Handle>... see ...</Handle>
  <User>
    <uid ...>epicuser</uid>
    <institute>ABCD</institute>
    <contact>Email</contact>
  </User>
</pidserviceresponse>
```

<Handle> Tag in the XML response:
```xml
<Handle>
  <pid>11858/...</pid>
  <url>http://www.gwdg.de/</url>
  <size>1234</size>
  <checksum>md5:123abc...</checksum>
  <pubdate>2011-09-08</pubdate>
  <title>GWDG homepage</title>
  <authors>...</authors>
  <metadata_url>http://...</metadata_url>
  <creator>epicuser</creator>
  <expdate>9999</expdate>
</Handle>
```
Creating a PID

**POST PIDSERVICE/write/create:**
- Minimal PID only needs:
  - url
- Optional:
  - encoding=xml

**POST PIDSERVICE/write/create:**
- Parameters:
  - url,
  - title,
  - authors,
  - size (in bytes),
  - checksum (md5:...),
  - pubdate (yyyy[-mm-dd [hh:mm:ss]]),
  - expdate (same),
  - metadata_url
- Optional:
  - suffix,
  - encoding=xml
- Minimal PID only needs: url
- authors, pubdate and title must be either all set or all not set;
- checksum can only be set once
POST PIDSERVICE/write/modify:

- Parameters:
  - pid
  - oldtitle (to validate correct PID)
  - and one or more of "create"
  - parameters to change fields
- Optional:
  - encoding=xml
- Returns:
  - HTTP status code
  - HTML or XML content
User management

User management, only for GWDG (EPIC) admins:

- GET user/view
- POST user/create
- POST user/modify

Force login and display user properties:

- GET write/whoami

Display user properties:

- GET read/whoami
  - user could be anonymous
HTTP Status Codes

- HTTP headers of responses contain HTTP status codes

Overview of HTTP status codes
- 200 OK request succeeded
- 201 CREATED a POST write/create succeeded
- 400 BAD REQUEST bad parameters or command
- 401 AUTH REQUIRED you have to log in first
- 403 FORBIDDEN user has no access permissions
- 404 NOT FOUND no search results, or attempt to view Handle which does not exist

- Note: GET read/view can also return HTTP redirects
Agenda

- Persistent Identifiers in the eResearch
- Consortium for PIDs
- EPIC PID-Service for the Research
  - RESTful Web Service
  - API of the PID-Service
- Examples
  - Python, Java, Shell
- Outlook
- Summary
#!/usr/bin/env python
import httplib2, pprint, urllib

# Service URL
url = 'http://handle.gwdg.de:8080/pidservice/write/create'

# Creates a new HTTP Object
http = httplib2.Http()

# User login
delicious_user = 'epicuser'
delicious_pass = 'epicpasswd'
http.add_credentials(delicious_user, delicious_pass)

# PID parameters
PID_url = "http://handle.gwdg.de/javadocs/"
PID_encoding = "xml"

# PID to URL
params = urllib.urlencode({
    'url': PID_url,
    'encoding': PID_encoding
})
print "---Request---"
print 'Encoded parameters:', params
response, content = http.request(url, 'POST', params,
    headers={'Content-type': 'application/x-www-form-urlencoded'}
)

# Print response to screen
print "---Content---"
pprint.pprint(content)
print "---Response---"
pprint.pprint(response)
Python: View a PID

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

import httplib2, pprint, urllib

# Creates a new HTTP object
http = httplib2.Http()

# User login
delicious_user = 'epicuser'
delicious_pass = 'epicpasswd'
http.add_credentials(delicious_user, delicious_pass)

# Handle URL for "CREATE" service
url = 'http://handle.gwdg.de:8080/pidservice/read/view'

# PID Infos
PID_encoding = "xml"

# PID to URL (pid=11858%2F60-ZZZZ-0600-0001-4743-4?showmenu=yes)
params3 = urllib.urlencode({
    'pid': '11858/00-ZZZZ-0600-0001-4743-4",
    'showmenu': 'yes",
    'encoding': PID_encoding,
    'proxyview': 'yes'  # shows the content of the "HANDLE" via Handle proxy site
    'redirect': 'yes'  # shows the original content (redirected to the url)
})

# Connect to the Service
print "---Request URL---"
print "Service URL: %s, url" % url
print "---Request parameters---"
print 'Encoded parameters:', params3
response, content = http.request(url+'?'+params3, method="GET")

print "---Status---"
if response['status'] == "200":
    try:
        print "All O.K. ...."
        print "PID Location: ", response['location']
    except:
        print "(Error) No PID Redirect"

if response['status'] == "403":
    print "(Error 403) Forbidden (Maybe there exists already a PID for this object)"

if response['status'] == "404":
    print "(Error 404) Not Found"

if response['status'] == "405":
    print "(Error 405) Not Allowed"
```
```python
#!/usr/bin/env python
import httplib2, pprint, urllib

url = 'http://handle.gwdg.de:8080/pidservice/write/modify'

# Creates a new HTPP Object
http = httplib2.Http()

# User login
delicious_user = 'epicuser'
delicious_pass = 'epicpasswd'
http.add_credentials(delicious_user, delicious_pass)

# PID parameters
new_PID_url = 'http://handle.gwdg.de/javadocs/
old_PID_title = "PIDService Documentation page"
new_PID_title = "PIDService Documentation"
PID_encoding = "xml"

# PID to URL
params = urllib.urlencode({
    'pid':"11858/00-ZZZZ-0000-0001-4743-4",
    'url': new_PID_url,
    'oldtitle': old_PID_title,
    'newtitle': new_PID_title,
    'encoding': PID_encoding
})
print "---Request---"
print 'Encoded parameters:', params

response, content = http.request(url, 'POST', params,
    headers={['Content-type': 'application/x-www-form-urlencoded']
}

# Print response to screen
print "---Content---"
pprint.pprint(content)
print "---Response---"
pprint.pprint(response)
```
Java: View a PID

```java
/**
 * Client for the REST API of the Persistent Identifier Service
 * For details see: http://www.pidconsortium.eu/index.php?page=process
 */
package de.gwdg.pidservice;

/**
 * @author Tibor [dot] Kalman [at] gwdg [dot] de
 */
public class PidClient {

    public static void main(String[] args) throws Exception {
        String pid = "11858/00-ZZZZ-0000-0001-4743-4";
        searchPid(pid);
        modifyPid(pid);
    }

    public static void searchPid(String pid) throws IOException {
        String serviceUrl = "http://handle.gwdg.de:8080/pidservice/read/search";
        String serviceParam = URLEncoder.encode(pid, "UTF-8");

        URL url = new URL(serviceUrl + "?" + "pid=" + serviceParam);
        URLConnection connection = url.openConnection();
        BufferedReader in = new BufferedReader(new InputStreamReader(
            connection.getInputStream()));

        String decodedString;
        while ((decodedString = in.readLine()) != null) {
            System.out.println(decodedString);
        }
        in.close();
    }
}
```
public static void modifyPid(String pid) throws IOException {
    String serviceUrl = "http://handle.gwdg.de:8088/pidservice/write/modify";
    String modifiedUrl = "http://handle.gwdg.de/javadoocs/";
    String oldTitle = "PIDService Documentation";
    String modifiedTitle = "PIDService Documentation";
    String serviceParam1 = URLEncoder.encode(pid, "UTF-8");
    String serviceParam2 = URLEncoder.encode(modifiedUrl, "UTF-8");
    String serviceParam3 = URLEncoder.encode(oldTitle, "UTF-8");
    String serviceParam4 = URLEncoder.encode(modifiedTitle, "UTF-8");

    URL url = new URL(serviceUrl);
    HttpURLConnection urlConnection = null;
    BufferedReader in = null;
    try {
        urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.setDoInput(true);
        urlConnection.setDoOutput(true);
        urlConnection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
        urlConnection.setRequestMethod("POST");

        String authCred = serviceUser + ":" + servicePwd;
        String encodedAuthCred = new sun.misc.BASE64Encoder().encode(authCred.getBytes());
        urlConnection.setRequestProperty("Authorization", "Basic " + encodedAuthCred);
        /*
        * Do not use sun.misc.*, see: "Sun proprietary API"
        * instead, use "Commons Codec Library" for Base64 Encoder
        * (http://commons.apache.org/codec/)
        * import org.apache.commons.codec.DecoderException;
        * import org.apache.commons.codec.binary.Base64;
        */

        OutputStreamWriter out = new OutputStreamWriter( urlConnection.getOutputStream());
        out.write("pid=" + serviceParam1);
        out.write("oldUrl=" + serviceParam2);
        out.write("oldTitle=" + serviceParam3);
        /* Modify the title here:
        * out.write("newUrl=" + serviceParam4);
        */
        out.write("encoding="xml");
        out.close();

        in = new BufferedReader(new InputStreamReader(urlConnection.getInputStream()));

        String decodedString;
        while ((decodedString = in.readLine()) != null) {
            System.out.println(decodedString);
        }
        in.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
Shell: View a PID and Create a PID

Use a simple `wget` (Unix) or `wget.exe` (Win) command for a GET request. For example, view the PID of this talk:

```
wget "http://handle.gwdg.de:8080/pidservice/read/view?pid=11858%2F00­ZZZZ­0000­0001­6D1A­5&showmenu=no"
```

For creating or modifying a handle, a POST request has to be executed. For instance a `curl` (Unix) or `curl.exe` (Win) command to create a PID for the website of this event:

```
```
Agenda

- Persistent Identifiers in the eResearch
- Consortium for PIDs
- EPIC PID-Service for the Research
  - RESTful Web Service
  - API of the PID-Service
- Examples
  - Python, Java, Shell
- Outlook
- Summary
Outlook

- The EPIC API v1 is very simple
- New features are required
  - New community requests, for example:
    - 1 PID with multiple URLs
    - 1 create request for multiple PIDs
    - and so on... :)
  - Also the Handle system evolves
- EPIC API v2 is coming soon :)
  - EPIC API v2 will be a more generic API
  - Designed, developed and implemented by the EPIC consortium
Agenda

- Persistent Identifiers in the eResearch
- Consortium for PIDs
- EPIC PID-Service for the Research
  - RESTful Web Service
  - API of the PID-Service
- Examples
  - Python, Java, Shell
- Outlook
- Summary
Summary

- Persistent Identifiers in the eResearch are needed and should part of the strategy for the long term preservation and long-lasting accessibility of scientific resources
- EPIC is a consortium which provides a PID-Service to the European research community
- EPIC PID-Service
  - EPIC API v1 is provided by a RESTful Web Service
- Python, Java, Shell examples are available
- EPIC API v2 will be a more generic API
Thank you!
Backup Slides
Referencing with Persistent Identifier

http://pidservice/identifier1

http://providerX/serverX/pathX/fileX

PID Database

Standardized Interface

Grid or Cloud technology

Middleware

Middleware

Middleware

Middleware

Storage

Storage

Storage

Storage

Database