DARIAH Storage API – A Basic Storage Service API on Bit Preservation Level

DARIAH-DE
Aufbau von Forschungsinfrastrukturen für die e-Humanities

Authors:

Stefan E. Funk, Peter Gietz, Martin Haase (DAASI International)

Patrick Harms, Andreas Aschenbrenner (SUB Göttingen)

Danah Tonne (KIT)

Jedrzej Rybicki (FZ Jülich)

Version: 1.0
File: DARIAH-Storage-API-v1.0_final
Created: 2011-10-14
Last Changes: 2012-11-26 (16:02:32)

Date	Version	Editor	Comment
2011-09-11	0.0.5	Stefan E. Funk	First version.
2011-09-12	0.0.6	Stefan E. Funk	Added more comments and new architectural image (still to discuss).
2011-09-19	0.1	Stefan E. Funk	Added chapter Use Cases, provided by Andi and Patrick.
2011-10-04	0.2	Stefan E. Funk	Translated from German (partly), added first plain HTTP API.
2011-10-07	0.3	Stefan E. Funk	Added HTTP CRUD operation descriptions.
2011-10-10	0.3.1	Stefan E. Funk	Added some comments.
2011-10-14	0.4	Stefan E. Funk	Added OPTIONS method, described storage concept.
2011-10-14	0.5	Peter Gietz	Added AAI-Concept.
2011-10-15	0.6	Stefan E. Funk	Formatting.
2011-10-21	0.7	Martin Haase, Peter Gietz	Added diagram and description of ECP, made some minor amendments in reaction to some comments.
2011-11-09	0.8	Stefan E. Funk	Adapted to recent e-mails, split up into AAI and Storage API documents.
2011-11-14	0.9	Jedrzej Rybicki	Accounted for on-going discussion, split operations into request/response. Added links to HTTP 1.1. Weak consistency definition added.
2011-11-17	0.9.1	Stefan E. Funk	Finalized for AP1.
2011-11-17	0.9.2	Stefan E. Funk	Removed left-over comments for AP1 release.
2011-12-09	0.9.3	Stefan E. Funk	Merged comments from Natasa.
2011-12-14	0.9.4	Stefan E. Funk	Commented comments from Patrick, updated Storage Service picture.
2011-12-15	0.9.5	Stefan E. Funk	Moved use-cases to appendix, accepted all the changes.
2011-12-16	0.9.6	Stefan E. Funk	Checked and added some HTTP technical things.
2011-12-19	0.9.6	Stefan E. Funk	Added PAOS header and examples as described in [DARIAH AAI].
2012-01-09	0.9.7	Stefan E. Funk	Added optional parameter PID to CREATE operations.
2012-02-09	1.0	Stefan E. Funk	Added last HTTP headers. Version set to 1.0.

Table Of Contents

1 Requirements of the Basic Storage Service API

- The API concentrates on bitstream preservation, but could be reusable as a general simple storage API.
- Based on the REST architectural style as described in [Fielding2011].
- Use of HTTPS/SSL.
- AAI functions will be covered – if possible – with HTTP authentication (SAML assertions and/or session IDs should be usable).
- Administrative metadata management will be provided.
- Since that API is likely to be extended, explicit versioning of the interface should be provided.
- Strict separation of create and update calls: An update can only be done with a HTTP PUT call on already existing resources (because you can't address a resource that doesn't yet exist!). A create call can be done using a HTTP POST on the storage resource itself. The storage implementation then will create a new ID for the resource to create, and delivers back its ID.
- Basic Storage Service will offer only a weak form of data consistency. In case of concurrent update and retrieval, it is only guaranteed that the resource representation sent back to the service client always is consistent (see Optimistic Locking).

2 Functionalities not covered by the Basic Storage Service

…but may be provided by DARIAH High Level Services.

- Community specific (descriptive) metadata management like indexing, searching, etc. (metadata can only be stored as a file).
- Technical metadata management, e.g. creating or storing JHOVE technical metadata providing detailled technical information on file format and version, validity and even more sophisticated metadata for curation usage.
- Versioning and revision management.
- Bulk ingest of many files, including transaction handling.
- Persistent identifier management (e.g. the GWDG Handle Service).
- Accounting is not part of the API.

3 Basic Storage Service Architecture

The Storage Service is working on representation level, so every API call is working for one representation, e.g. a file identified by an URL containing its ID. The Storage Service itself can be addressed by an URL itself, e.g. for POSTing new representations of files to

the Storage Service.

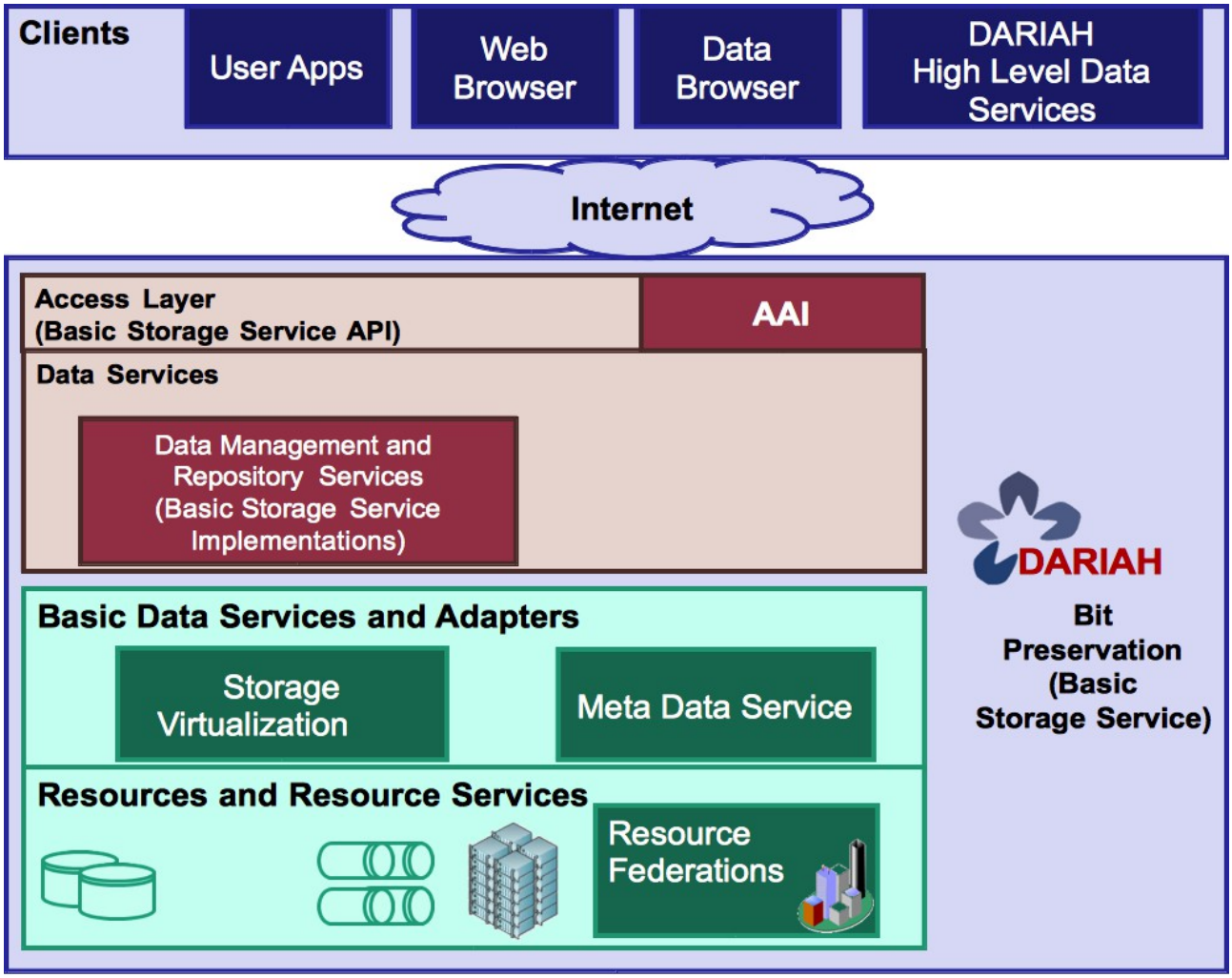

Figure 1: General DARIAH Storage Architecture

Thus this Basic Storage Service API is represented by the **Access Layer** in the picture above. The **AAI** is handled in the API (e.g. by containing authentification and authorisation information), not in the Basic Storage Service implementations, respectively the service implementation will be simply relaying that information to the AAI service, and requests access. Furthermore the **Data Management & Repository Services** represent the Basic Storage Service implementations, including metadata management for basic storage purposes (e.g. LastModified timestamps, etc.) and management of **Basic Data Services and Adapters** (e.g. the question, which repository shall be used by the user authenticated).

Higher level services like community and technical metadata management, versioning and revision management, bulk ingest, and PID management will be provided by the ***DARIAH High Level Data Services***

Each of the actions described below must be processed as an atomic action.

Asynchronous or concurrent service calls must not be lead to inconsistent data.

4 DARIAH RESTful Storage API

The DARIAH RESTful Storage API is based on the REST architectural style as described in [Fielding2011]. In principle “REST components perform actions on a resource by using a representation”. Furthermore unless stated otherwise the API implements the semantics of the HTTP method, Headers and Error as defined in [rfc2616].

Resources are addressed by URLs, and the actions to perform on a representation will be performed by using some of the basic HTTP methods to achieve basic CRUD operations on the resources (create, retrieve, update, and delete).

The general form of the RESTful request URL is:

`http://<service-base-url>/[<ID>]`

where <`service-base-url`> represents the storage service's base URL, and <`ID`> represents the resource (e.g. a file or the storage service itself) that shall be worked with.

4.1 AAI Integration

In general every service call using AAI as described in [DARIAH AAI] needs to send the header and body data as given in the following HTTP POST example. If no HTTP body is needed for the HTTP method used, the use of multipart messages is not necessary.

4.2 HTTP POST

Via POST a new resource is being created. You have to address the Storage Service directly with it's resource name, because you do not yet have an URL to the resource to be created, because it has not been created yet.

POST			
Request			
Service	URL	Mandatory	The service's location.
Version	String	Optional	The API version.
Accept	PAOS binding	Optional	Indicate support for ECP profile, see [DARIAH AAI].
PAOS-Header	PAOS header	Optional	PAOS namespaces and version as described in [DARIAH AAI].
PAOS-Content-Type	Mime	Optional	PAOS mimetype as described in [DARIAH AAI].
PAOS-Data	Octet-Stream	Optional	PAOS XML data as described in [DARIAH AAI].
Content-Type	Mime	Mandatory	Content-Type of the provided data.
PID	URI	Optional	A persistent identifier that points to

			an aggregation the resource belongs to. It can be used to aggregate resources on Storage Service Implementation Level [UC3 CF].
Data	Octet-Stream	Mandatory	The file's content.
Response			
Code	201 Created		
Location	URL	Mandatory	Returns the URL of the resource.
Last-Modified	DateTime	Mandatory	Last modification date.
Etag	String	Optional	Opaque resource version ID.
Side effects	Creates a new resource.		
Errors	401 Unauthorized		
	405 Method Not Allowed (POST on an already existing resource)		
	503 Service unavailable		
	500 Internal Server Error		

General request:

```
UA: POST / HTTP/1.x
UA: Host: <Service>
[UA: Version: <Version>]
[UA: Accept: <PAOS-Content-Type>]
[UA: PAOS: <PAOS-Header>]
[UA: PID: urn:dariah:x53981]
[UA: Content-Type: multipart/mixed; boundary="gc0p4J"]
[UA: --gc0p4J]
[UA: Content-Type: <PAOS-Content-Type>]
[UA: <PAOS-Data>]
[UA: --gc0p4J]
UA: Content-Type: <Content-Type>
UA: <Data>
[UA: --gc0p4J--]
```

Example request (no auth):

```
UA: POST / HTTP/1.x
UA: Host: storage.dariah.eu
```

```
UA: Version: 1.0
UA: Content-Type: text/xml
UA: <xml><kindof>This is XML</kindof></xml>
```

Example response:

```
OS: HTTP/1.x 201 Created
OS: Location: http://storage.dariah.eu/urn:dariah:578x
OS: Content-Type: text/xml
OS: Transfer-Encoding: chunked
OS: Date: Thu, 09 Feb 2012 08:12:31 GMT
OS: Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
OS: Etag: "478fb2358f700"
```

4.3 HTTP GET

Via GET a resource is being *retrieved*. The API provides only a weak consistency, that is in case of concurrent GET and PUT, the service client can assume that either the old version of the resource (before a concurrent update) or the newer version of the resource (after a concorrent update) will be provided, and not a mixture of old and new.

GET is a safe method [rfc2616], i.e. the request has no side-effects and can be repeated to produce the same effect.

GET			
Request			
Service	URL	Mandatory	The service's location.
ID	Identifier	Mandatory	The ID of the file to retrieve.
Version	String	Optional	The API version.
Accept	PAOS binding	Optional	Indicate support for ECP profile, see [DARIAH AAI].
Etag	String	Optional	The Etag of the resource, given by the server.
Response			
Code	200 OK		
	304 Not Modified		
Content-Type	Mime	Mandatory	Content-Type of the provided data.
Last-Modified	DateTime	Mandatory	Last modification date.
Etag	String	Optional	Opaque resource version ID.
Data	Octet-stream	Mandatory	The resource's content.

Side effects	None.	
Errors	401	Unauthorized
	404	Not Found
	503	Service unavailable
	500	Internal Server Error

General request:

```
UA: GET <ID> HTTP/1.x
UA: Host: <Service>
[UA: Version: <Version>]
[UA: Accept: <PAOS-Content-Type>]
[UA: If-None-Match: <Etag>]
```

Example request (no auth):

```
UA: GET /urn:dariah:578x HTTP/1.x
UA: Host: storage.dariah.eu
UA: Version: 1.0
UA: If-None-Match: "478fb2358f700"
```

Example response:

```
OS: HTTP/1.x 200 OK
OS: Content-Type: text/xml
OS: Content-Length: 123
OS: Transfer-Encoding: chunked
OS: Date: Thu, 09 Feb 2012 08:12:31 GMT
OS: Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
OS: Etag: "478fb2358f700"
OS: <xml><kindof>This is XML</kindof></xml>
```

4.4 HTTP HEAD

Via HEAD a resource is being retrieved without the content itself (HTTP header only). It provides exactly the same header structure as a full-blown GET would do.

HEAD is a safe method, i.e. the request can be repeated without side-effects and produce the same result.

HEAD

Request			
Service	URL	Mandatory	The service's location.
ID	Identifier	Mandatory	The ID of the file to retrieve.
Version	String	Optional	The API version.
Accept	PAOS binding	Optional	Indicate support for ECP profile, see [DARIAH AAI].
Etag	String	Optional	The Etag of the resource, given by the server.
Response			
Code	200 OK		
	304 Not Modified		
Content-Type	Mime	Mandatory	Content-Type of the requested resource.
Last-Modified	DateTime	Mandatory	Last modification date.
Etag	String	Optional	Opaque resource version ID.
Side effects	None.		
Errors	401 Unauthorized		
	404 Not Found		
	503 Service unavailable		
	500 Internal Server Error		

General request:

```
UA: HEAD <ID> HTTP/1.x
UA: Host: <Service>
[UA: Version: <Version>]
[UA: Accept: <PAOS-Content-Type>]
[UA: If-None-Match: <Etag>]
```

Example request (no auth):

```
UA: HEAD /urn:dariah:578x HTTP/1.x
UA: Host: storage.dariah.eu
UA: Version: 1.0
UA: If-None-Match: "478fb2358f700"
```

Example response:

```
OS: HTTP/1.x 200 OK
OS: Content-Type: text/xml
```

```
OS: Content-Length: 123
OS: Date: Thu, 09 Feb 2012 08:12:31 GMT
OS: Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
OS: Etag: "478fb2358f700"
```

4.5 HTTP PUT

Via PUT a resource is being updated.

PUT is an idempotent method [rfc2616], that is the side-effect of a series of identical PUT requests is exactly equivalent to the side-effect of a single request.

PUT			
Request			
Service	URL	Mandatory	The service's location.
ID	Identifier	Mandatory	The ID of the file to store.
Version	String	Optional	The API version.
Content-Type	Mime	Mandatory	Content-Type of the provided data.
Accept	PAOS binding	Optional	Indicate support for ECP profile, see [DARIAH AAI].
PID	URI	Optional	A persistent identifier that points to an aggregation the resource belongs to.It can be used to aggregate resources on Storage Service Implementation Level [UC3 CF].
Data	Octet-Stream	Mandatory	The file's content.
Response			
Code	201 Created		
Last-Modified	DateTime	Mandatory	Last modification date.
Etag	String	Optional	Opaque resource version ID.
Side effects	The resource is updated.		
Errors	401 Unauthorized		
	404 Not Found		
	409 Conflict (resource has not not been updated due to a concurrent API call, see optimistic locking)		
	503 Service unavailable		
	500 Internal Server Error		

General request:

```
UA: PUT <ID> HTTP/1.x
UA: Host: <Service>
[UA: Version: <Version>]
[UA: Accept: <PAOS-Content-Type>]
[UA: PID: urn:dariah:x53981]
UA: Content-Type: <Content-Type>
UA: <Data>
```

Example request (no auth):

```
UA: PUT /urn:dariah:578x HTTP/1.x
UA: Host: storage.dariah.eu
UA: Version: 1.0
UA: Content-Type: text/xml
UA: <xml><kindof>This is XML</kindof></xml>
```

Example response :

```
OS: HTTP/1.x 201 Created
OS: Location: http://storage.dariah.eu/urn:dariah:578x
OS: Content-Type: text/xml
OS: Transfer-Encoding: chunked
OS: Date: Thu, 09 Feb 2012 08:12:31 GMT
OS: Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
OS: Etag: "478fb2358f700"
```

4.6 HTTP DELETE

Via DELETE a resource is being deleted.

DELETE is an idempotent method, that is the side-effect of a series of identical DELETE requests is exactly equivalent to the side-effect of a single request.

DELETE			
Request			
Service	URL	Mandatory	The service's location.
Version	Identifier	Optional	The API version.
ID	Identifier	Mandatory	The ID of the file to store.

Accept	PAOS binding	Optional	Indicate support for ECP profile, see [DARIAH AAI].
Response			
Code	204 No Content		
Last-Modified	DateTime	Mandatory	Last modification date.
Side effects	Deletes the resource.		
Errors	401 Unauthorized		
	404 Not Found		
	409 Conflict (resource has not been deleted due to a concurrent API call, see optimistic locking)		
	503 Service unavailable		
	500 Internal Server Error		

General request:

```
UA: DELETE <ID> HTTP/1.x
UA: Host: <Service>
[UA: Version: <Version>]
[UA: Accept: <PAOS-Content-Type>]
```

Example request (no auth):

```
UA: DELETE /urn:dariah:578x HTTP/1.x
UA: Host: storage.dariah.eu
UA: Version: 1.0
```

Example response

```
OS: HTTP/1.x 204 No Content
OS: Date: Thu, 09 Feb 2012 08:12:31 GMT
```

4.7 HTTP OPTIONS

Via OPTIONS the client requests information about the HTTP methods the server implements. Maybe there are service implementations that do e.g. read only.

OPTIONS			
Service	URL	Mandatory	The service's location.
Version	Identifier	Optional	The API version.

General request:

```
UA: OPTIONS * HTTP/1.x
UA: Host: <Service>
[UA: Version: <Version>]
```

Example request:

```
UA: OPTIONS * HTTP/1.x
UA: Host: storage.dariah.eu
UA: Version: 1.0
```

Example response

```
OS: HTTP/1.x 200 OK
OS: Date: Thu, 09 Feb 2012 08:12:31 GMT
OS: Allow: OPTIONS, GET, HEAD, POST, PUT, DELETE
OS: Content-Length: 0
```

5 Optimistic Locking

The Basic Storage Service uses optimistic locking for ensuring data consistency. It locking pertains to update operations for all resources of the DARIAH Basic Storage Service. It is automatically enforced by the service without any user interaction. The main goal is to avoid conflicts between two concurrent update operations on the same resource. Optimistic locking acts on the assumption that conflicts occur rarely. Typical locking strategies necessitate the locking of a resource for retrieve (read) operations as well, which can impair many users, especially in scenarios with a high read-to-write ratio.

Optimistic locking always allows retrieving resources. However, it cannot avoid that a concurrent user overwrites a resource while another user is still modifying it locally; it just can detect the conflict and notify users. The typical flow of actions for an update operation with optimistic locking is as follows:

- The user retrieves a resource. Its representation contains a last-modification-date attribute automatically set by the storage service.
- The user modifies the resource representation locally.
- The user stores the modified resource back to the originating service by invoking the update() method.
- The service compares the last-modification-date attribute in the representation with the last-modification-date of the latest version of the object stored in the repository. If the latter date is newer, the originally retrieved version has been meanwhile overwritten by a concurrent user. To avoid a conflict, the service will not execute the update, but raise an exception to inform the user about the conflict.

Optimistic locking is only relevant for update operations. Create operations are by definition not affected by concurrency. Delete operations have the ultimate goal to remove

a resource, so an intermediate update of the resource by a concurrent user should not block the deletion. Retrieve operations are always possible.

6 Performance Issues

Current version of the API covers only very basic usages of the Basic Storage Service. It is likely that the API would be extended in the future. In particular the performance issues will be addressed when occurred. It is expected that the future version of the API support following basic extensions on that filed:

- Conditional operations: for GET and PUT, HTTP headers If-Modified-Since: Date time, If-Match: Etag, will be used. So that the service client has a possibility to request a resource only when it was updated since the last access time. Support of these HTTP features will allow for content caching on local sites.
- In order to deal with larger resources efficiently, partial GET might be supported. This might be realized via HTTP range header and status code 206 (Partial Content). Client will be then able to request only a subset of bytes for a given resource.
- Since the process of ingesting data might take more time (e.g. due to a sophisticated content validation or integrity checks), API might provide asynchronous uploads. In such cases server will response to a POST request with 202 Accepted and URI for the resource which will be created later.

7 References

[Fielding2000]	Fielding, Roy Thomas: „Architectural Styles and the Design of Network-based Software Architectures“, 2000
[handle]	http://www.handle.net/
[rfc2616]	HTTP 1.1 Standard (RFC)
[DARIAH AAI]	DARIAH Authorization and Authentication Infrastructure
[UC3 CF]	UC3 Curation Foundations, Rev. 0.13 (2010-03-25)

8 Appendix

8.1 Use Cases

The following use-cases not only describe the requirements for the Basic Storage Service (including bit preservation), but additionally the functionalities of higher level services, that are not covered by the Basic Storage Service API. The Basic Storage Service will most of the time be only accessed indirectly using graphical user interfaces and/or higher level services that do access the Basic Storage Service via the Basic Storage Service API.

8.1.1 Storage vs Bit Preservation

$Scholar at a German center for digital humanities ($GCDH) has finalised her work on a DFG-funded research project (German Research Foundation). As part of the contractual requirements, $Scholar committed herself to retaining the results of the research project for at least 10 years. However, $Scholar worked on a temporary contract and after the research project has completed, she moved on to another DH-centre in India. The research data would have been lost.
Fortunately, $GCDH has established policies and technical mechanisms to store all the data of its employees in a preservation vault (storage). Although that vault does not yet validate the data and does not ensure the availability of research metadata, the data are forwarded to the DARIAH Basic Storage Service for long-term availibility.[1] The contract between $GCDH and the service provider ensures that the data will be preserved without accidental changes to the original bit-stream for a 10-year period. After 10 years, the service provider informs the $GCDH about the pending end of the retention period. The $GCDH may then decide to either extend the retention period or dispose of the data.

8.1.2 HTTP Read Rate, Open Access Data

$Scholar from the Digital Humanities Centre in Urbana, US, aims to access an image – stored in DARIAH Basic Storage Service – via her web browser. $Scholar determined the URL of the image via a dedicated web catalogue. The image is Open Access, so she is able to view the image with adequate HTTP read rate for web viewing.

8.1.3 Rights Management, Possibility to Delete Data

$Institute has been working on the literary remains of the author Thomas Bernhard since 1991. For bitstream preservation purposes they stored the data in the DARIAH Basic Storage Service. Through his last will and testament, some of the works of Thomas Bernhard are closed for public for several more decades. Even though $Institute carefully rights-managed the data (including authorisation mechanisms provided by Basic Storage Service), the literary executor sued them

1 Other aspects of bit preservation have been defined by WissGrid: http://www.wissgrid.de/workgroups/ap3/2011-03-08--bitstream-preservation.pdf

and they are now required by law to delete all traces of Bernhard texts from the Basic Storage Service.

8.1.4 Update, Delete and Versions

$Student is in the key phase of her PhD thesis, and she works heavily on the text. Since she does not trust the safety of her laptop in her shared flat, she prefers to store the data online in a trusted storage. She prefers a service that internally uses the DARIAH Basic Storage Service for bitstream preservation. Every day she uploads numerous updates of the Microsoft Word source files. These updates are transient and do not need to be re-accessed (overwriting the source). Only after finalising a chapter (which is roughly every week), she uploads a new version with a new file name that receives persistent identification and can be re-accessed later.

8.1.5 Establishing Context of the Data Through References in the PIDs

$Scholar is searching for a specific bit of evidence for her work on the 1st world war. She happens to find a digitised letter written by a Serbian soldier, that contradicts her theory. Before refuting her work of the last 3 years, she carefully scrutinises the letter to establish its trustworthiness.
The trustworthiness of research data is inherently tied to the research context they have been created in. However, the DARIAH Basic Storage Service offers storage facility “as a black box”, without knowing about the nature of the data or whether it contains resp. links to research data.
Instead the context of the data may be established through the Persistent Identifier (PID) system (in our case often: Handle [handle]). Thereby, the PID references both the data (in the Basic Storage Service) as well as the metadata (stored in a separate file in the Basic Storage Service or in another system altogether). On top of that, the PID may include checksums (for either data and metadata, or for both) to ensure the integrity of the object over time. It is recommended that the metadata is exposed as an OAI-ORE description, that references all (potentially distributed) data components as well as any "behaviours"[2] of the objects.

8.1.6 "Behaviours" of Objects Attached to the Basic Storage Service

$Scholar from the Digital Humanities Centre in Urbana, US, aims to access an image (stored in DARIAH Basic Storage Service, available as Open Access) via her web browser. The image is more than 150 MB, but the service provides her with an image that was scaled and compressed on-the-fly to only 1 MB for adequate read rate and suitable web viewing.
→ http://digilib.berlios.de/
→ http://djatoka.sourceforge.net/

2 For a definition of "behaviours" or "disseminators" see e.g. http://www.cs.cornell.edu/payette/papers/ecdl98/fedora.html

8.1.7 Bulk Ingest and Access

$Research-Network is working on the literary remains of painter Markus Prachensky. They are in the process of digitising his key works (mainly paintings, only few hand sketches) and aim to link them to the works of Mondrian and other abstract artists. The digitisations are often huge (several 100 MBs) and are ingested in bulk to the DARIAH Basic Storage Service (with access restricted to only key members of the $Research-Network) from their Apple server environment. After a year of work, $Research-Network initiates a cooperative research project with the Mondrian research network, who bulk-download all the images to analyse and compare the color schemes and shading through their image mining mechanisms.