PersID
IV – Prototype for a Meta Resolver System / Work on Standards

Overview and studies on persistent identifier infrastructure commissioned by Knowledge Exchange and Prototype development of Meta Resolver Solution commissioned by SURFfoundation
About this publication

PersID – IV
Prototype for a Meta Resolver System / Work on Standards

A Knowledge Exchange and SURF initiative: Studies on Persistent Identifier Infrastructure and development of a URN-NBN based Global Resolution Service

SURFfoundation
PO Box 2290
NL-3500 GG Utrecht
Netherlands
T +31 30 234 66 00
F +31 30 233 29 60

Knowledge Exchange
Danish Agency for Libraries and Media
H. C. Andersens Boulevard 2
DK-1553 Copenhagen V
Denmark
T +45 3373 3315

info@surf.nl
www.surf.nl

office@knowledge-exchange.info
www.knowledge-exchange.info

Authors
Roberto Puccinelli - CNR
Edgardo Ambrosi - CNR
Maurizio Lunghi - FDR
Emanuelle Bellini - FDR
Stefano Turchi - FDR
Juha Hakala - National Library Finland

Editor
Bas Cordewener - SURFfoundation

Knowledge Exchange is a co-operative effort that supports the use and development of Information and Communications Technologies (ICT) infrastructure for higher education and research.

SURF is the collaborative organisation for higher education institutions and research institutes aimed at breakthrough innovations in ICT (www.surf.nl/en)

This publication is online available through www.knowledge-exchange.info

© Knowledge Exchange partners
April 2011
urn:nbn:nl:ui:13-wt1-6n9

This publication is published under the Creative Commons Attribution 3.0 Netherlands Licence.
Contents

1 Working Prototype .................................................................................................................. 5
  1.1 Underlying documents and guidance .............................................................................. 5

2 Work on URN-NBN related standards .................................................................................. 7
  2.1 Achievements within PersID ........................................................................................... 7
  2.2 Future perspective ........................................................................................................... 7

3 Annex IV.a – Architectural Model for a Global Resolver System ....................................... 9
  3.1 Introduction ..................................................................................................................... 9
  3.2 Component diagram ....................................................................................................... 9
  3.3 Deployment .................................................................................................................... 10
    3.3.1 Nodes ...................................................................................................................... 10
    3.3.2 External components .............................................................................................. 11
    3.3.3 Deployment diagrams ............................................................................................ 11
  3.4 Interactions ..................................................................................................................... 14
    3.4.1 Direct resolution: trusted (URN:NBN) PID to digital resource ......................... 14
    3.4.2 Direct resolution: trusted (URN:NBN) PID to descriptive metadata .............. 16
    3.4.3 Direct resolution: other PID to resource or metadata ........................................... 17
    3.4.4 Mediated resolution: trusted URN:NBN to resource ............................................ 17
    3.4.5 Mediated resolution: trusted URN:NBN to metadata ............................................ 19
    3.4.6 Mediated resolution: other PID to resource or descriptive metadata .............. 20
  3.5 Interfaces ....................................................................................................................... 21
    3.5.1 Query types ............................................................................................................ 21
    3.5.2 Response schemas .................................................................................................. 22
  3.6 Error codes ..................................................................................................................... 22
  3.7 Security .......................................................................................................................... 23
    3.8 High Availability ......................................................................................................... 23
      3.8.1 Local system redundancy ...................................................................................... 23
    3.8.2 Global redundancy ................................................................................................. 24
    3.9 Monitoring .................................................................................................................. 25

4 Annex IV.b – Meta Resolver Overview, State of the Art and Usage .................................... 26
  4.1 Introduction ..................................................................................................................... 26
  4.2 Technological overview ................................................................................................. 26
    4.2.1 Meta resolver’s Technology .................................................................................. 26
  4.3 Meta resolver’s Architecture ......................................................................................... 27
  4.4 Java packages and web components ............................................................................... 31
    4.4.1 Meta resolver’s Synchronizer ................................................................................ 31
  4.5 The Server Interface ...................................................................................................... 32
    4.5.1 Administration ........................................................................................................ 36
    4.5.2 Searching the Meta resolver’s Registry ................................................................ 38
    4.5.3 Identifier Resolution .............................................................................................. 38
  4.6 Meta resolver’s Functionality .......................................................................................... 39
    4.6.1 MUST Programming Interfaces ........................................................................... 39
    4.6.2 SHOULD HAVE Programming Interfaces .............................................................. 40
    4.6.3 COULD HAVE and WISH TO HAVE Programming Interfaces ....................... 40
  4.7 Appendix ....................................................................................................................... 40

5 Annex IV.c – Documentation Global Resolver Service Synchronizer (GRSS) ..................... 43
  5.1 GRSS package ................................................................................................................ 43
  5.2 How GRSS works .......................................................................................................... 43
  5.3 Log File and Event Log File .......................................................................................... 44
  5.4 Persistence and Persistent Actions ................................................................................ 45
  5.5 Logger and Logging Operations ...................................................................................... 47
1 Working Prototype

The prototype has been developed on the basis of the User Requirements and the Architectural Model that were defined in the PersID project. The term working prototype is used to describe the purpose of the developing: proving the feasibility and the value of cross border resolution of URN-NBNs (and potentially other PIDs) based on and maintaining an agreed level of trust between partners with regards to accessibility, long term preservation and general PID quality issues.

The prototype is an open source product and may be further developed to fulfil the full potential of a Meta Resolver Service and become a mature production service. It is also possible that new software will be developed to perform the meta resolver service. The current software is operational and can be used and/or downloaded.

The Meta resolver package is available at URL: https://scm.cedrc.cnr.it/download/jglore/jglores.zip
It includes the source code, the license and the bin-package.
A Synchroniser tool (to establish robust interaction between MRS and local resolvers) has been developed as an optional addition to the Meta Resolver (annex IV.c)

1.1 Underlying documents and guidance

- The Architectural Model for the MRS is available as Annex IV.a.
- The User Requirements for the MRS have been derived from Report III.b.
- The description of the actual prototype, its workings and usage instructions can be found in Annex IV.c. ‘Meta Resolver’ and IV.d. Synchroniser

The description of the development and resulting prototype can be found in Annex IV.b. Meta Resolver for PersID - Overview, State of the Art and Usage. A description of the synchroniser tool can be found in Annex IV.d. Synchronizer documentation.
2 Work on URN-NBN related standards

The PersID project, to successfully establish a reliable and internationally interoperable persistent identification infrastructure, aims to realise a trusting community, non-proprietary ownership of the PID system and the capability to anticipate on relevant future services.

These conditions for satisfactory international performance are best met by the URN-NBN option, because of the good experience with current national solutions, its inclusiveness (other PID systems can be resolved as well) and - being defined by IETF internet standards - its independence from vendors or technology platforms.

The current RFCs concerned with URNs and especially URN-NBNs were established in 1997. Since then the internet has changed and developed as well as function and usage of URNs. Therefore the PersID partners have an interest and in their policy agree to contribute to the updating and refinement of relevant RFCs and to comply to existing and future URN and URN-NBN related standards.

2.1 Achievements within PersID

In the context of PersID a paper was written to outline current thinking and current practice regarding persistent identifier systems. Although a general overview of existing persistent identifier systems, the importance and benefits of a standardised system are highlighted. This document, by Juha Hakala of the National Library Finland, is available in the report on Communication.

The IETF has been requested to form a working group to start the process of revising existing, outdated RFCs concerning URN-NBNs. This URNbis working group has now been established and has started to work. A first attempt has been made to redraft the RFC's #2114, #3187 and #3188. These first drafts can be found as annexes to this document.

2.2 Future perspective

The process to redraft and agree on new RFCs exceeds the running time of the PersID initiative. Partners however agreed to continue their support by contributing to the IETF coordinated work. This is the best possible way to move forward and achieve a standard based on specifications that do not depend on vendor-decisions or IT platform developments. For URN-NBN users this may be a crucial ‘must have’ to accomplish their purposes (e.g. Long Term Preservation), for other systems the standards will be an inspiring reference for further development. Overall a standard will provide clarity and will stimulate cooperation and convergence.
3  Annex IV.a – Architectural Model for a Global Resolver System

3.1  Introduction

The present document describes the architecture of the Global Resolution System that will be developed and deployed by the PersID project. This architecture complies with the principles described in the “WP4 technical proposal overview”, which can be summarized as follows:

- Unique GUI for resolving all NBNs belonging to the participating countries;
- Lightweight and non-invasive “Adapting layer” (GRS Adapter) that will allow national systems to be interoperable with the GRS; the purpose of this layer is to unify the way national systems are queried and the formats they use to return information;
- Definition of trusted NBN resolution procedures;
- Definition of the error conditions managed by the system;
- Strategies for high availability of the system;
- Definition of the security features;
- Basic administration tools and monitoring tools;
- Resolution of other non-URN PIDs via redirect.

3.2  Component diagram

Figure 1 shows the GRS components. Note that the GRS itself is represented as a macro-component including all the others.

![GRS component diagram](image)

**Figure 1:** GRS component diagram

Below are the descriptions of each component.

**Resolver**

*Description*

This component manages all the resolution tasks performed by the GRS, interacting with the registers that keep the PID-related information.
**Ports**

1. **Resolve port**: this port is used to accept and send resolution requests; it exposes an input interface for incoming requests and an output interface for outgoing requests: requests are issued as HTTP GET commands with properly formatted URLs as parameters (see section 3.5.1 “Query URL syntax”) and responses are provided as XML or HTML streams, the latter generated from the former by applying an appropriate XSLT stylesheets (see section 3.5.2 "Response schema").

**User Manager**

**Description**

This component manages the user accounts, which currently all belong to a single “Administrator” profile. Additional profiles may be added in the future if necessary.

**Synchronizer**

**Description**

This component manages synchronization between GRS master and GRS replicas. As stated in the technical proposal, there will be one GRS master and one or more replicas, hosted at different sites, in order to ensure that at least one global resolver is up and running. Each replica periodically contacts the master in order to synchronize its database with the master's one. Master and replicas feature the same software, thus the synchronizer can act both as master and as replica, based on the configuration implemented by the administrator.

**Ports:**

1. **Sync port**: this port is used to accept (when the GRS is configured as "master") or send (when it is configured as "replica") synchronization requests; it exposes input interface for incoming requests and an output interface for outgoing requests.

**Administrative Info Manager**

**Description**

This component allows the administrator to manage all the pieces of information that are exposed on the GRS portal, such as GRS policies and procedures. This allows to easily change what is published on the portal without redeploying the software.

**Register manager**

**Description**

This component allows to manage the information regarding the registers (NBN and non-NBN, trusted and not trusted) belonging to the GRS infrastructure. In particular, this is used by the administrator to insert a new register in the data base or to modify/delete an existing one. It is also used by the normal web user to browse the list of the registers and get detailed information about them.

### 3.3 Deployment

The current section describes how the different components of the PersID infrastructure are deployed and how they interact. For sake of readability, different types of interaction have been illustrated in different deployment diagrams, though the infrastructure is unique.

#### 3.3.1 Nodes

Below are the descriptions of the node types involved (i.e. servers or workstation in the present case).
**User work station**
*Description*: the computer from which a normal web user issues his/her resolution requests via web browser.

**GRS Server**
*Description*: the machine(s) where the GRS is installed.

**Trusted URN:NBN server**
*Description*: a machine hosting a trusted URN:NBN resolver.

**Other trusted PID server**
*Description*: a machine hosting a trusted PID resolver (e.g. an URN resolver which processes namespaces different from NBN).

**Other PID server**
*Description*: a machine hosting a PID resolver not belonging to the above two categories.

### 3.3.2 External components

Below are the descriptions of the external components participating to the infrastructure that are not part of the GRS.

**Web browser**
*Description*: a software for navigating the world wide web.

**Trusted URN:NBN resolver**
*Description*: a software which manages an NBN trusted resolution.

**Trusted PID resolver**
*Description*: a software which manages a trusted PID resolution (i.e. trusted resolution of non URN:NBN PIDs).

**PID Resolver**
*Description*: a software which manages a PID resolution.

### 3.3.3 Deployment diagrams

This section describes how components are deployed on the different nodes and which are their mutual dependencies. For readability reasons dependencies pertaining to different usage scenario are illustrated in different diagrams. Note that the dashed arrows indicate functional dependencies between components and not data paths, which are not shown.

**Scenario 1: direct resolution**
The first deployment diagram highlights the dependencies that are significant for the scenario where a web user contacts the GRS via web browser to issue his/her request. For sake of simplicity, in this diagram and in the following ones ports and interfaces are not shown.
A resolution request is generally issued by a web user via web browser. The GRS delegates the resolution to the appropriate trusted resolver or redirects it to the appropriate non-trusted resolver. In the former case the response is provided by the trusted resolver as an XML stream that the GRS transforms in an HTML page using an appropriate style sheet for presentation. In the latter case the web browser is redirected to the appropriate non-trusted resolver, which provides the answer in a separate browser window.

Scenario 2: mediated resolution

The second deployment diagram highlights the dependencies that are significant for the scenario where a web user contacts a trusted resolver to issue a resolution request regarding a PID that is not managed by the trusted resolver itself.
Figure 3: PersID deployment diagram (mediated access scenario)
In this case the trusted resolver forwards the request to the GRS, which acts as in the previous scenario, except that it sends the response to the trusted resolver that forwards it to the web user.

**Scenario 3: synchronization**

This diagram highlights the dependency between replicas and master in the synchronization task.

![PersID deployment diagram (synchronization scenario)](image)

**3.4 Interactions**

This section illustrates the different types of interaction that can occur between the user web browser, the GRS and the resolvers (Trusted URN:NBN resolvers, Trusted PID resolvers, Other resolvers). Each interaction is described by means of a sequence diagram. As in the previous section, the expression "direct resolution" is used to indicate that the user web browser (or an ad hoc smart client) issues the resolution request directly to the GRS, whereas "mediated resolution" is used to indicate that the user web browser (or an ad hoc smart client) issues the resolution request to a trusted resolver, which needs to forward the request to the GRS in order to get the correct answer.

"Trusted URN:NBN resolvers" and "Trusted PID" resolvers interactions follow the same patterns, thus not requiring separate descriptions. As regards "Other resolvers" interactions, resource and metadata retrieval are not described separately, being both treated by means of redirection.

**3.4.1 Direct resolution: trusted (URN:NBN) PID to digital resource**

**Description**

In this scenario, the user contacts the GRS via web browser in order to resolve a (URN:NBN) PID managed by a trusted resolver to the corresponding digital resource.

**Interaction**

Two possible solutions are presented below.
Solution A

Figure 5: direct resolution of trusted (URN:NBN) PID to resource (solution A)

Call 1): The web browser issues an HTTP POST to the GRS, providing the (URN:NBN) PID as parameter.

Call 2): The GRS sends an HTTP GET request to the appropriate trusted resolver, passing the (URN:NBN) PID as parameter.

Call 2) return: the trusted resolver replies with an HTTP 303 redirect to the digital resource associated to the (URN:NBN) PID.

Call 1) return: the GRS forwards the HTTP 303 redirect to the user’s web browser.

Solution B

Figure 6: direct resolution of trusted (URN:NBN) PID to resource (solution B)

Call 1): The web browser issues an HTTP POST to the GRS, providing the (URN:NBN) PID as parameter.

Call 2): The GRS sends an HTTP GET request to the appropriate trusted resolver, passing the (URN:NBN) PID as parameter.
3.4.2 Direct resolution: trusted (URN:NBN) PID to descriptive metadata

Description
In this scenario, the user contacts the GRS via web browser in order to resolve a (URN:NBN) PID managed by a trusted resolver to the corresponding descriptive metadata.

Interaction

**Figure 7:** Direct resolution of a trusted (URN:NBN) PID to the corresponding descriptive metadata

**Call 1:** The web browser issues an HTTP POST to the GRS, providing two parameters: the (URN:NBN) PID and the request type (indicated by one of the service codes described in RFC 2483). In the present case the service code is I2C (URI to URC).

**Call 2:** The GRS sends an HTTP GET request to the appropriate trusted resolver, passing the (URN:NBN) PID and the service code as parameters.

**Call 2 return:** The trusted resolver replies with an XML stream containing the requested metadata.
Call 3): the GRS applies the appropriate XSLT style sheet to the XML stream.

Call 3) return: the GRS produces an HTML page that contains the requested metadata.

Call 1) return: the GRS forwards the HTML page containing the requested metadata to the user’s web browser.

3.4.3 Direct resolution: other PID to resource or metadata

Description
In this scenario, the user contacts the GRS via web browser in order to resolve a PID managed by a non-trusted resolver “c” to the corresponding digital resource or descriptive metadata.

Interaction

![Diagram](Diagram.png)

**Figure 8:** Direct resolution of a non-trusted PID to the corresponding resource or metadata

Call 1): The web browser issues an HTTP POST to the GRS, providing the PID as parameter and the possibly the request type in case of metadata retrieval.

Call 1) return: the GRS returns an HTTP 303 redirect to the URL to be used in order to query resolver “c”.

Call 2): The web browser sends an HTTP GET request to resolver “c”, using the URL provided by the GRS.

Call 2) return: resolver “c”, based on the request type, replies with the digital resource or the requested metadata.

3.4.4 Mediated resolution: trusted URN:NBN to resource

Description
In this scenario, the user contacts a trusted resolver “a” via web browser in order to resolve a (URN:NBN) PID, managed by a different trusted resolver “d”, to the corresponding digital resource.

Interaction
Two solutions are presented below.
**Solution A**

![Diagram](image)

**Figure 9:** Mediated resolution of trusted URN:NBN to resource (solution A)

**Call 1:** the web browser issues an HTTP POST to resolver "a", providing the PID as parameter.

**Call 2:** resolver “a” sends an HTTP GET request to the GRS, providing the PID as parameter.

**Call 3:** the GRS sends an HTTP GET request to resolver “d”, providing the PID as parameter.

**Call 3) return:** resolver “d” returns to GRS an HTTP 303 redirect to the requested resource.

**Call 2) return:** the GRS forwards to resolver “a” the HTTP 303 redirect.

**Call 1) return:** resolver “a” forwards to the web browser the HTTP 303 redirect.
**Solution B**

*Figure 10: Mediated resolution of trusted URN:NBN to resource (solution B)*

**Call 1):** The web browser issues an HTTP POST to resolver "a", providing the PID as parameter.

**Call 2):** Resolver “a” sends an HTTP GET request to the GRS, providing the PID as parameter.

**Call 3):** The GRS sends an HTTP GET request to resolver “d”, providing the PID as parameter.

**Call 3) return:** Resolver “d” returns to GRS the URL of the requested resource within an XML stream that complies with the schema presented in section 3.5.2.

**Call 2) return:** The GRS forwards to resolver “a” the XML response received from resolver “d”.

**Call 1) return:** Resolver “a” returns to the web browser an appropriate HTTP 303 redirect to the URL extracted from the XML response.

The same considerations presented in section 3.4.1 as regards the comparison of solution A and B apply in this case.

### 3.4.5 Mediated resolution: trusted URN:NBN to metadata

**Description**

In this scenario, the user contacts a trusted resolver “a” via web browser in order to resolve a (URN:NBN) PID, managed by a different trusted resolver “d”, to the corresponding descriptive metadata.
3.4.6 Mediated resolution: other PID to resource or descriptive metadata

Description
In this scenario, the user contacts a trusted resolver “a” via web browser in order to resolve a PID managed by a non-trusted resolver (“Other PID resolver”) to the corresponding digital resource or descriptive metadata.
Interaction

**Call 1**: the web browser issues an HTTP POST to resolver “a”, providing the PID and possibly the request type (e.g. “I2C”) in cases other than resource retrieval.

**Call 2**: resolver ”a” sends an HTTP GET request to the GRS, providing the PID and possibly the request type (e.g. “I2C”) in cases other than resource retrieval.

**Call 2) return**: the GRS returns to resolver ”a” an appropriate HTTP 303 redirect to resolver ”c”, providing the URL to be used for querying it.

**Call 1) return**: resolver ”a” forwards the HTTP 303 redirect to the web browser.

**Call 3**: the web browser sends an HTTP GET request to resolver ”c”, using the URL provided by resolver ”a”.

**Call 3) return**: resolver ”c”, based on the request type, provides access to the digital resource or the associated metadata.

### 3.5 Interfaces

This section describes the interfaces that the GRS exposes toward the outer world (external use agents). In particular, the GRS accepts/issues resolution requests in the form of HTTP GET commands (with properly formatted URLs as arguments) and replies with/expects XML streams as responses. The same interfaces must be exposed by the local resolvers. In the following subsections we provide information about supported query types and response XML schema. Details about query URL syntax and the actual response XML schema will be provided in a separate document (“PersID Interfaces”).

#### 3.5.1 Query types

Below is the list of the supported interrogations. Some of them must be mandatorily supported by all the participating resolvers and are marked with (M). The others are optional and are marked with (O).

1. **resource retrieval (M)**: retrieval of the actual digital resource corresponding to the specified PID;
2. **descriptive metadata retrieval (O):** retrieval of the descriptive metadata corresponding to the specified PID;

3. **administrative metadata retrieval,** which can be further divided into the following three interrogation types:
   a. **technical metadata retrieval (O):** retrieval of the information specifying the hardware and software requirements for resource usage;
   b. **preservation metadata retrieval (O):** retrieval of the preservation metadata such as changes between the identified version of the resource and the previous / subsequent versions;
   c. **rights metadata retrieval (O):** retrieval of the usage rights information corresponding to the specified resource;

4. **PID info retrieval (M):** retrieval of the information regarding the specified PID (date of creation, identifier creator, identifier owner(s));

5. **organisational commitment info retrieval (O):** retrieval of the information corresponding to the organisational commitment for preserving the identified version of the resource;

6. **URL list retrieval (M):** retrieval of the list of URLs that point to the identified resource.

7. **URN list retrieval (O):** retrieval of the list of URNs that identify other manifestations of the same work.

### 3.5.2 Response schemas

Responses to HTTP GET commands are provided by the GRS and by the local registers as XML streams. Each XML response includes one string element representing the PID and one complex element representing the information bundle, which, based on the request type, contains one of the following items:

- the primary resource URL,
- the list of all known URLs,
- the list of all known URNs related to the resource (other manifestations and the description of the work itself),
- the resource metadata (descriptive, rights, technical, preservation policy),
- the PID metadata,
- an error message.

### 3.6 Error codes

Table 1 contains the list of the managed error conditions. Each one has a unique integer identifier (code), a mnemonic name and an extended description.

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Malformed NBN</td>
<td>The specified NBN name does not comply with format specifications</td>
</tr>
<tr>
<td>2</td>
<td>Unsupported PID type</td>
<td>The PID type is not one of those the GRS can handle</td>
</tr>
<tr>
<td>3</td>
<td>Non existent NBN</td>
<td>The specified NBN is well-formed but does not correspond to any actual resource</td>
</tr>
<tr>
<td>4</td>
<td>Resolver unreachable</td>
<td>The resolver does not respond within the timeout interval</td>
</tr>
<tr>
<td>5</td>
<td>GRS internal error</td>
<td>The GRS has encountered an internal error</td>
</tr>
<tr>
<td>6</td>
<td>Malformed answer</td>
<td>The resolver has provided a malformed answer</td>
</tr>
</tbody>
</table>
# Security

Security issues regard the user authentication (only for administrators) and server authentication (how GRS and trusted resolvers mutually identify themselves). As regards user authentication, this is achieved by associating login credentials to each registered user (username and password). As regards server authentication, the both the GRS and the trusted resolver feature server certificates, which enable them to prove their identity when contacted by an external system. Resolution requests can be sent via https and include an authentication token which proves the identity of the requester (which, depending on the usage scenario, may be the GRS or a trusted resolver). This credentials cannot be "sniffed" on the network, since https is an encrypted protocol.

# High Availability

## Local system redundancy

Each GRS site will have to implement a redundant architecture, in order to ensure high availability at the local level. A possible solution, here described only by way of example, could feature two or more apache web servers acting as web front-ends and sharing a common virtual host name. Load balancing among them could be implemented at the firewall level. Requests to the advertised GRS’ URL would be evenly dispatched to the available front-ends.

Two or more JEE application servers could host the application. The web front-ends would evenly dispatch user sessions to the available application servers.

The data base could be managed by a cluster of two servers in standard hardware failover configuration (one active and the other ready to take over the load in case of failure) or in parallel cluster configuration.

Actual data could be stored in a disk array connected via fibre channel and configured in RAID 1 or 5. The connection could be “direct attach” or via Storage Area Network.
3.8.2 Global redundancy

Global redundancy will be achieved by running more instances of the GRS at different sites. There will be one master GRS and one or more replicas. National resolvers will subscribe to the master GRS and the replicas will periodically contact the master in order to receive updates regarding registered resolvers.
3.9 Monitoring

Monitoring will be performed by means of open source tools. In particular the following platforms will be leveraged:

- **Nagios**, which allows monitoring server availability and process status. Other useful parameters can be subsequently defined and monitored;
- **AWStats**, which allows to produce statistics about web accesses, request types, daily independent users, etc.

Both tools feature a web interface.
Annex IV.b – Meta Resolver Overview, State of the Art and Usage

4.1 Introduction

The present document contains relevant information regarding the PesID Meta resolver both from the technical and end-user perspective. The Meta resolver is a software able to provide a unique access point for persistent identifier (PID) resolution services managed by different and geographically distributed organizations. In particular, it acts as a mediator between the end user willing to resolve a persistent identifier and the registers/resolvers which provide the actual resolution service. The interaction between the Meta resolver and the registers/resolvers can take place in two different ways:

- via standard PesID REST API;
- via request redirection.

The first method requires the remote register/resolver to expose the REST API described in section xx of the present document. In this case the Meta resolver invokes the correct method exposed by the remote register/resolver using the PID specified by the end-user as a parameter, formats the XML response applying the appropriate style sheet and forwards the result to the end user. The second method just requires the Meta resolver to be aware of the URL formats to be used in order to invoke the remote resolution services. In this case the Meta resolver just composes the URL and performs an HTTP 303 redirect. No further mediation or response processing is performed by the Meta resolver. The Meta resolver provides a web interface, for human to machine interaction, and a REST API form machine to machine interaction. The web interface features:

- a public section, where the user can access the resolution service and relevant information about the system, the PersID initiative etc.;
- a password protected administration interface, which allows to configure the system.

The present document starts with an overview of the technologies used to implement the Meta resolver and then describes the software components which make up the system. It then provides the specification of the REST API which allow machine to machine interaction. The last sections are devoted to the Web user interface and to the installation procedure. For an extensive description of the system architecture see [1].

4.2 Technological overview

4.2.1 Meta resolver’s Technology

The Meta resolver, as specified in the architectural model [1], has been developed both for human to machine interaction (via web interface) and for machine to machine interaction (via REST API). The core architecture is based on the SURF extension of the SPRING framework. SPRING is a well established MVC framework, which allows for quick implementation of Java Enterprise applications. SURF, building on the SPRING features and capabilities, provides a way for developing web user interfaces using server-side scripts and templates. SURF makes it extremely easy to create pages, templates, components and layouts for web applications and also REST-based programming interfaces. This latter feature is particularly useful for the implementation of SPIs (Service Programming Interface).

SPRING SURF facilitates the design and implementation of complex applications where, except for business logic, everything is delegated to a framework with strong support from the developing community. As a SPRING extension, SURF also ensures access to many features of the underlying MVC framework.
The Meta resolver web interface is based on ZKOSS, an Ajax framework for the development of Rich Internet Applications, which hides to the developer the JavaScript details of the user interface implementation. In particular, one added value achieved during the development of the Meta resolver is that a tight integration between the two frameworks, SURF and ZKOSS, has been designed, developed and successfully tested. Here below is the complete list of the technologies used for the Meta resolver implementation:

- Maven Building System
- Spring Framework
- Spring Roo
- Spring WebFlow
- Spring SURF
- Zkoss
- FreeMarker Templates
- Hibernate
- Database

### 4.3 Meta resolver’s Architecture

In the following some architectural aspects are shown. The Meta resolver contains a very sophisticated control layer. Such layer is completely hidden by Hibernate, Spring, Spring SURF and ZKOSS components. As shown in figure 1, data sets are managed via Hibernate which maps Java Objects to Database tables. Hibernate provides a very high level API for interacting with data.

![Hibernate Architecture](image)

**Figure 1:** Hibernate

The Meta resolver defines some data models as shown in figure 2. The models are specified in Java language and mapped to tables via Java Annotation Mechanism. Annotations are used for those attributes that must be declared persistent.
The set of APIs provided by Hibernate is one of the building blocks which the Spring framework is built on. The figure 3 offers a view of the main Spring components. The Meta resolver is developed according to the MVC paradigm implemented by Spring, which uses a built-in DAO component to interact with Hibernate.

The Meta resolver as above mentioned is a server. As such it offers a protocol to the clients for all interactions. The protocol is implemented via Web Scripts. The Web Script mechanism is based on the REST paradigm. A Web Script is composed by:

1. the first and most important SPI layer, that are URLs with parameters. Clients must invoke such URLs via the GET or POST methods defined by the HTTP protocol.
2. a second component for the business logic. The business logic represents the complete set of operations and elaborations that the Meta resolver is required to be able to perform. Logics can be implemented by Java Script and via pure Java code.

3. a third component to render the data model, that is the answer provided to the client.

The Meta resolver exposes a set of Web Scripts, which provide access to the majority of the Meta resolver’s functions. Only Web Scripts related to data persistency, described above, are implemented via pure Java language. The usage of pure Java to manage the persistent layer is out of the scope of the present document. Suffice it to say that the server must implement some controllers, which directly interact with the data layer, that need to be coded in Java.

![Figure 4: Web Script Logics via Java Backed Beans](image)

As shown in figure 4 some Web Script controllers are developed via the Java Backed Bean mechanism of the Spring framework. As already explained, the Meta resolver is completely based on different complex technologies that cooperate with each other. An insight is provided below for the correlated involved components. From a conceptual point of view, the figure 3, completely defines the Meta resolver. The main component is the PersID. It has the sole purpose of providing a wrapper for the underlying layers.
The component that is directly connected with Meta resolver logics is the "Service Programming Interfaces (SPI), which is responsible for exposing the Meta resolver as a server. The SPIs are used by both web applications (like the Meta resolver portal) and other third-party applications. The SPI component is directly linked to two fundamental components that implement the logics of the server: “Web Script via Java Script” engine (WSJS) and the “Web Script via Java Backed Beans” component (WSJBB). The former is interfaced with the SPI through the “Spring SURF” (SS) layer. Both components WSJS and WSJBB are connected to data via the “Spring framework” (SF) and the “HIBERNATE layer” (H). SF and H make data management and persistence transparent to
SPIs. Finally, the SPIs component is connected to two components which render the results provided by the Meta resolver: the “Freemarker Template” component (FTL), which is a native Viewer for SS, and the new ZKOSs component, which represents a step-over development achieved by CNR through the integration of ZKOSs and SS. This add-on represents an interesting evolution of the SS framework.

4.4 Java packages and web components

The present section lists and describes the Java packages and web components developed for the Meta resolver. Further details on the meaning, syntax and use of the web components can be found in the ZKoss, SPRING and SURF documentation.

Java Components:
- `org.persid.jglore.controller`. The “controller” package implements the logic for some operations such as, initialization of the authentication system, security and login control management.
- `org.persid.jglore.files`, the “file” package implements the logic for XSD schema management. Currently the system uses one XSD to validate all XML replies received from the remote resolvers. A further enhancement could be to enable the management of multiple XSD schemas, which are selected based on the type of reply provided by the remote resolver. This, given the modular architecture of the system, could be achieved with minimal development effort.
- `org.persid.jglore.model`. The “model” package implements the persistent entities managed by Hibernate. The persistency layer makes use of the Java Annotation Technique.
- `org.persid.jglore.service`. The “service” package implements the access point to the controller of roles assigned at authentication time.
- `org.persid.jglore.service.impl`. The “impl” package implements different types of service logic, such as CRUD operations over the persistency layer, validation mechanism via XSD for National Resolver answers, trasformation via XSL for some National Resolver answers.
- `org.persid.jglore.util`. The “util” package implements some utilities such as an Http Client, functions for JSON de/serialization, functions for JSON de/en-coding.
- `org.persid.jglore.web.scripts`. The “web.scripts” package contains control logic, implemented via Java Backed Beans, for some Web Scripts. For example, the persistency layer can be accessed via Web Scripts and the control logic is implemented in the “util” package.

Main Web Components:
- “Templates” defines the Page Models used by the SURF Framework and the Pages rendered via ZKOSs.
- “Footer“, defined in the “Templates” component, implements via Web Script and Freemarker Templates the Page’s footers.
- “Header”, defined in the “Templates” component, implements via Web Script and Freemarker Templates the Page’s headers.
- “Navigation”, defined in the “Templates” component, implements via Web Script and Freemarker Templates the SURF Components. For example, “Components” defines the different menus (vertical, horizontal, left and right) available in the application web interface.
- “Pages” implements via Web Script the functional components of the Meta resolver. For example, “Pages” defines different Views related to the functions required by [1].
- zk pages provides the bridge between SURF and ZKOSs.
- zk view, provides the Views rendered via ZKOSs.

4.4.1 Meta resolver’s Synchronizer

In order to ensure high availability of the Meta resolver service, the Architectural Model [1] specifies that there should be one Master Meta resolver and one or more replicas, which can be queried in case the Master is down. Synchronization between Master and replica configurations is
provided by a separate component, the Synchronizer. This component must be installed on all Meta resolver servers and requires the system administrator to specify whether the server is acting as a Master or as a replica. This allows Meta resolver replicas to periodically contact the Master to update their configurations. For more details on this component, please refer to the document [4].

4.5 The Server Interface

The Meta resolver consists of a server and a set of web interfaces. The server will be discussed below. In this section the interfaces are briefly described. The implemented Meta resolver’s web components adhere to the architectural and functional requirements prescribed in [1] and [2]. Below the Meta resolver’s features exposed to the users are briefly listed. More details are provided in the next subsection.

- Persistent Identifier resolution as shown in figure 6. This interface allows to retrieve a digital resource or the related metadata by specifying the assigned persistent identifier. The user just need to type the persistent identifier in the text box, select the desired type of search and press the “Resolve” button.

Figure 6: PID Resolution

- Browse list of resolvers. This section allows the user to browse the list of the resolvers that are registered in the Meta resolver data base. The user can type a search string in the input box and/or set values for some filters, which allow to narrow the scope of his/her query (figure 7).
Management of Meta resolver configuration. The password protected “Global Administration” section of the Meta resolver web interface allows the system administrator to manage the configuration of the Meta resolver. In particular, the following items can be managed:

- National Resolver details as shown in figure 8. The buttons on the top of the table allow to load/refresh the list of resolvers (in order to ensure visualization of the last modifications), add a new resolver, update the information about one selected resolver, delete one or more selected resolvers. Resolvers can be selected by checking the boxes on the left of the respective table rows.
Partner details as shown in figure 9. The buttons on the top of the table allow to load/refresh the list of partners (in order to ensure visualization of the last modifications), add a new partner, update the information about one selected partner, delete one or more selected partners. Partners can be selected by checking the boxes on the left of the respective table rows. The information managed in this section is used to fill the partner’s list published in the top right box of the Meta resolver home page.

![Figure 9: Partner Administration](image)

Registered User details as shown in figure 10. The buttons on the top of the table allow to load/refresh the list of users (in order to ensure visualization of the last modifications), add a new user, update the information about one selected user, delete one or more selected users. Users can be selected by checking the boxes on the left of the respective table rows.

![Figure 10: User Administration](image)
- Server content management as shown in figure 11, and
- Community content management as shown in figure 11. All textual contents published on the Meta resolver portal can be managed in this section. In order to edit one item (e.g. MR description), press the “Edit” button on the right of the respective text area, modify the content and then press the “Save” button. The new content is immediately available in the appropriate area of the Meta resolver portal.

**Figure 11:** Content Administration

XSD schema management. This section allows to manage the XSD schemas used to validate XML messages sent and received by the Meta resolver. In particular, it is possible to upload a new XSD schema, validate an XML file using the current XSD schema and check for XSD schema existence. Figure 12.

**Figure 12:** Validation Schema Administration
It is worth pointing out that the information entered via the ContentManagement section, is exposed in different parts of the Meta resolver portal. For example, the content handled by the Policy management section is made immediately and permanently available in the “Policy” section. Other managed contents regard contact persons (“Contact us” portal section), procedures to join the initiative (“How to Join Us”), information about the initiative (“About Us”), information about the Meta resolver (“About MR”), and finally information about the NBN standard and related aspects (“About NBN standard”). Table 5 shows the sections containing the mentioned contents.

Moreover, a content summary related to users, partners and National Resolvers is shown in a collapsible box, which is placed at the top right of the appropriate sections of the Meta resolver portal. For example, in the home page of the Meta resolver portal it is possible to expand the list of partners by clicking on the “Partners” label.

### Table 1: Example of Content Context

#### 4.5.1 Administration

**User Details**
The User-detail management allows to perform the CRUD (Create, Read, Update, Delete) operations on the “User” DB table. The managed User information is the following:
- UserName,
- Password,
- FirstName,
- MiddleName,
- LastName
Username and password are required to login the server. The Meta resolver has been equipped with a profiling system. The profiler assigns a role to users who are allowed to login. Roles are intercepted at various points of the Spring framework, such as in Components, Pages and Web Scripts. According to the assigned role, the server allows or denies access to resources.

Currently, only the administrator and guest profiles are defined. Guest is the default role assigned to users that are not logged in. Currently, a logged in user gets the role of administrator. In the future additional roles will be defined to map the responsibilities of different types of users.

**National Resolver Details**
The National Resolver-detail management allows to perform the CRUD (Create, Read, Update, Delete) operations on the Registers DB table. The managed information for National Resolvers is the following:
- Name,
- Url,
- Type,
- Trusted,
- Policy,
- Organization,
- Business Model,
- Web Site,
- Contact,
- Standard,
- Digital Resource,
- Metadata Resource.
Part of this information is used by the Meta resolver for the resolution process and part is exposed to the user in the register browsing section of the portal.

**Partner Details**
The Partner-detail management allows to perform the CRUD (Create, Read, Update, Delete) operations on the Partners DB table (list of members of the PersID community). The managed information is the following:
- Name,
- Url
The URL points to the official partner’s web site.

**Content Management Details**
The information items regarding the Meta resolver that can be managed from the Meta resolver administration interface are the following:
- MR Description,
- People,
- About MR

The information items regarding the PersID initiative that can be managed from the Meta resolver administration interface are the following:
- Policy,
- How to join the Infrastructure,
- Other Info,
- NBN Description
Such contents are made available in different parts of the web interface.

**Validating Schema Details**
The Meta resolver resolution mechanism requires the participation of the National Resolvers. In particular, the Meta resolver can forward the resolution request to the appropriate National Resolver in two ways: if the National Resolver exposes the PersID SPI, the Meta resolver invokes
the appropriate REST service and the result is returned as an XML stream, whose content is then formatted by the Meta resolver using an appropriate style sheet and presented to the final user; if the National Resolver does not expose the PersID SPI, the request is simply redirected to the appropriate National Resolver using its specific query URL syntax. No further processing is performed by the Meta resolver. The XML streams must be validated before being used for composing the answers for the final users. The XSD schemas are the standard tool for XML validation. The PersID community has agreed on a specific XSD schema, which defines a basic syntax. The Meta resolver is equipped with a module for XSD management. Three operations can be performed:
- Upload XSD
- Validation Response
- Check XSD

These functions allow to permanently store the XSD schemas. Storing multiple schemas will be crucial in the future in order to be able to manage multiple response models. Currently, only one schema is used (the first of the stored list). The “Upload XSD” function allows to upload a new XSD schema. The validation function enables the administrator to manually check a response received from a national registry. The “Check XSD” function allows to check the availability of at least one XSD schema.

4.5.2 Searching the Meta resolver’s Registry

The Meta resolver’s registry of local resolvers can be searched via web interface. The user can specify a search string and/or set the following filters:
- Type, allows to specify the type of register (NBN, NON-NBN),
- Country, allows to specify the country, which is responsible for the register (when this applies),
- Trusted, allows to specify whether the register adheres to the PersID policy.

4.5.3 Identifier Resolution

The main function provided by the server, both via REST SPI and web interface, is the identifier resolution. The user just needs to type the identifier in the text box at the centre of the Meta resolver home page, select the search option and press the “Resolve” button. At the moment, the identifier is parsed just to identify the National Resolver to be queried for resolution but no syntax checking is performed. The search function features two modes, the “Easy” and the “Advanced” one. The easy search has two options:
- “All Metadata”, retrieval of all the associated metadata.

The advanced search has more options:
- “Metadata”, retrieval of resource metadata,
- “Info about Identifier”, retrieval of PID metadata (information about the PID itself),
- “Primary URL”, the primary URL of the digital resource,
- “URL List”, the list of known URLs pointing to the digital resource.

For more information about the Meta resolver’s resolution behavior, please read the architectural document [1].

As previously stated, the Meta resolver is made up by a server and a web interface. Many server’s functions are made available both via web interface and via SPI (Service Programming Interface). The SPI adheres to the REST specification and is implemented via SPRING SURF.

REST and SPRING SURF allow to leverage the Meta resolver as a platform for building new third-party applications or interfaces towards legacy applications. They also provide high decoupling
between the Meta resolver and the external components that interact with it. Adding new functions is thus quite easy and implies a very low impact on the rest of the source code. This implementation choice makes the system very easily extendable.

4.6 Meta resolver’s Functionality

The Meta resolver offers SPIs based on the REST protocol. Such SPIs (Programming Interface) are accessible via http protocol. The main service is the PID resolution. Basically the Meta resolver can perform PID resolution in two ways: Delegation and Forwarding as defined in [2]. The Delegation requires that the National Resolver exposes the PersID standard SPI. The Forwarding is used when the National Resolver does not expose the PersID standard SPI. In this latter case the National Resolver is responsible for result formatting an presentation. More details are provided in [2]. The user requirement document [2] specifies that the Meta resolver exposes some essential interfaces. It must be remembered that all necessary data are managed by a software layer that makes such data permanent. The persistence layer can be accessed via the web. The Meta resolver's Web Interfaces and SPIs are classified below according to the MoSCoW paradigm adopted in the user requirement document [2]. This helps understanding where at the moment the development activities can be positioned in an ideal roadmap.

4.6.1 MUST Programming Interfaces

The MUST requirements are a set of features that needed to be implemented in order for the Meta resolver to be considered acceptable.

- Set and Give information on how to refer to a PID in the PersID infrastructure. Such content can be set via web administration and is shown in the Home page [http://persid-home/](http://persid-home/). It is a dynamic content. Its modification does not require the server to be restarted. The previous statements are valid for all server’s content mentioned in the following. By default, the home page provides the PID resolution service and information about a suitable usage of the resolution engine.
- Set and Give information on the Meta resolver itself. The “About Us!” section ([http://persid-home/aboutUs](http://persid-home/aboutUs)) shows the content related to the server and to the different National Resolvers, such as NBN TRUSTED, OTHER TRUSTED, OTHER RESOLVERS.
- Searching the Meta resolver’s registry. The section [http://persid-home/BrowseRegistry](http://persid-home/BrowseRegistry) provides the interface for retrieving National Resolvers.
- The sections
  - [http://persid-home/resolversCrud](http://persid-home/resolversCrud)
  - [http://persid-home/partnersCrud](http://persid-home/partnersCrud)
  - [http://persid-home/usersCrud](http://persid-home/usersCrud)
  - [http://persid-home/schemaCrud](http://persid-home/schemaCrud)
  - [http://persid-home/infosCrud](http://persid-home/infosCrud)
make available the CRUD operations for National Resolver, PersID partners, users, validation schemas, and various contents. The managed data are also presented in side menus and boxes placed in the appropriate parts of the portal. The CRUD mechanisms are also used for querying the Meta resolver DB.
- Resolution of PID and retrieval of digital resources. The resolution is available via web and via SPIs. The following SPIs are available:
The Meta resolver allows to resolve PIDs, both via web and SPIs. It first parses the PID in order to identify the National Resolver that can resolve it. Once the National Resolver is identified, the Meta resolver queries it in order to retrieve the resource or the associated metadata.

4.6.2 SHOULD HAVE Programming Interfaces

SHOULD requirements are considered important but not critical for the project. Nevertheless, all of them have been implemented. The “Contact Us” section (http://persid-home/contactUs) presents information about
• people responsible for the Meta resolver,
• how to register a resolver and become partner of PersID,
• and other information.

The persistence layer and the management module allows to dynamically change such content without the need to restart the server.

4.6.3 COULD HAVE and WISH TO HAVE Programming Interfaces

As requested in the [2], the Meta resolver has been developed as a meta resolution service for any type of PID (URN, ISBN, etc. ...). It is designed to provide a registration system for different National Resolvers and a resolution mechanism for several resource identifiers. Thus both the COULD HAVE and the WISH TO HAVE requirements have been implemented (see [2]).

References
[1] PersID Architectural Model
[2] PersID User Requirements
[4] Synchronizer

4.7 Appendix

A - Download
The Meta resolver software of the PersID project is based on the concept of Maven artifact. In a few words, we can say that Maven simplifies the process of software building and packaging. It provides a set of specialized tools for managing the dependencies of a Java project. It has a mechanism for a rapid construction and deployment of a software. Thus, every partner of the PersID project will find extremely easy to connect to the SVN managed by CNR and download the Meta resolver code. Maven features can also be leveraged by those partners willing to extend their resolvers with the PersID standard SPI. The stable release of the Meta resolver can be found at web URL https://scm.cedrc.cnr.it/nbn/tags/jglores. The credentials to download the source code are:
• account:persid members
• password:persid all456.

The command to checkout the project is
svn co https://scm.cedrc.cnr.it/nbn/tags/jglores

B - Configuration
Once downloaded the project, few simple commands must be issued to download the dependencies from external Maven repositories, build the project and deploy it.
• Enter the folder jglore.
• Run the Maven command to satisfy the declared dependencies (mvn update:dependence).
• Run the Maven command to build the project (mvn build).
Run the Maven command to deploy the WAR package (mvn deploy).

The project can be deployed on any JEE web container (e.g. Tomcat). Once the application server has been started and the Meta resolver application has been booted, the administrator can login with his/her credentials (username: admin, password: p3r21d)

The first thing the administrator should do is changing the password in the “Global Administration” section of the portal.

**B.1 - Data Persistence**

Access to the persistent data layer can be configured and customized by the administrator. It is strongly recommended to refer to the Hibernate project documentation [3]. A minimal standard configuration is provided within the Meta resolver. Such configuration is defined in the file jglore/src/main/resources/META-INF/spring/defaultrepository.properties. The following properties are set up and can be changed:

- jglore.connection.driver class=org.hsqldb.jdbcDriver
- jglore.connection.url=jdbc:/hsqldb:/file:/jgloredb
- jglore.connection.username=sa
- jglore.connection.password=
- jglore.hibernate.dialect=org.hibernate.dialect.HSQLDialect

Such configuration represents a basic connection to HSQLDB via JDBC driver. A file on the server’s file system is set as the default data storage. The credentials to access the DB are specified here as well as the SQL dialect that Hibernate must use to communicate with HSQLDB.

**C - Installing and Booting**

The Maven tool builds the Meta resolver as a WAR (Web Archives Repository). The deploying phase strictly depends on the Application Server adopted by the organization. The booting phase completely depends on the Application Server. For details on Deploying, Booting and Monitoring a Web App please read the specific documentation.
5    Annex IV.c – Documentation Global Resolver Service Synchronizer (GRSS)

The present document is about the Global Resolver Service Synchronizer, an application developed within the PersID project This documentation is intended to be the GRSS user manual, containing explanations about the GRSS functioning and instructions for its integration.

The GRSSynchronizer is a Spring Web Application designed with the intent to enable the synchronization between different nodes within a network. The use case scenario is the following: a network layer consists of \( n \) nodes divided in two sets. A set of replica nodes having a cardinality of \( n-1 \), and a set of master nodes containing a single element. Every node has a database which represents its own persistent state. A replica node must periodically synchronize itself with the master node through an automatic procedure, so that the whole system is kept coherent Since the system of interest is about a network layer where the nodes are related as stated above, from now on the following definitions will be taken:

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>It a system of peer nodes where one of them is a Master and the rest are Slaves.</td>
</tr>
<tr>
<td>Node</td>
<td>It is an element of the Network</td>
</tr>
<tr>
<td>Master Node (or Master)</td>
<td>It is the reference Node for the synchronization procedure</td>
</tr>
<tr>
<td>Slave Node (or Slave)</td>
<td>It is a Node which must be kept synchronized with the Master</td>
</tr>
<tr>
<td>Synchronize</td>
<td>It is the operation responsible of keeping the Slaves aligned with the Master</td>
</tr>
</tbody>
</table>

Table 1: Definitions

5.1    GRSS package

The GRSS package is given with the following software elements:
- The synchronization system
- The persistence layer
- The interface to interact with the persistence layer via web controller
- The event log system
- An embedded Quartz scheduler to perform custom periodic synchronizations

5.2    How GRSS works

Let us suppose the Network is set up and the Slave knows the identity of the Master (i.e. its IP address). Every Node owns a Log File (see further) which contains the database tables version numbers and represents the Node persistence state. Depending on the temporal parameters of its scheduler, the Slave periodically contacts the Master to verify its alignment; once expired the waiting time, the Slave sends a request to the Master asking for the database tables versions to confront them with the ones it owns. If the versions check is successful (i.e. no misalignment is detected) the Slave will end the procedure and it will close the connection. If the versions check detects a misalignment, the Slave will request the Master to send a JSON serialized message containing the misaligned tables. Please note that the version check is about every single table, so that it is possible to perform table-level updates saving bandwidth and resources. When the Slave receives the data, it can update the corresponding tables aligning its database. Finally, it refreshes the Log File setting its own table versions to the Master’ s ones.
Figure 1: A simplified view of GRSS System

In Figure 1 is represented a simplified view of the whole system. On the centre are shown the three main elements responsible for the synchronization process: the Synchronizer which acts as a client-side component, the SynchService which acts as a server-side component and the Logger which is the component managing the current Node persistence state representation.

5.3 Log File and Event Log File

To understand the GRSS behaviour it is important to grasp the difference between Log File and Event Log File. Usually, we can refer to a "log" as a document containing information concerning every action performed by (or on) a system. According to such definition we can expect that a log consists of a huge amount of lines describing every event happened since a certain time. Within the GRSS context this kind of document corresponds to the Event Log File. Event Log File is not involved in the synchronization process at all, and it exists only for diagnostic and monitoring purposes. On the other hand, Log File is at the heart of the synchronization process and it is quite different from an ordinary log. It aims to log (or represent) a state, not an history and it consists of just one line regardless of the system lifetime.

Log File is stored locally and enables a memory-less tracing of every operation performed on a database table which changes the table state (e.g. a "write" operation will be logged while a "read" operation will not). Every time a persistent operation is performed the Log File is refreshed to reflect the new state of the Node. To clear the things up, it is important to figure out that the Log File intent is about labelling the current database state to satisfy the following:
To clear the things up, it is important to figure out that the Log File intent is about labelling the current database state to satisfy the following:

- If a Node is misaligned the system must be able to recognize its state via Log File analysis and run the alignment procedure.
- If a Node is aligned the system must be able recognize its state via Log File analysis, avoiding unnecessary alignment execution.

Sometimes\(^1\), it can be tolerated a Node performing the alignment procedure while being aligned, but it can’t be tolerated a Node not performing the alignment procedure while being misaligned.

**Statement 1:** coherence preserving main assumption

We make a final exemplification. Let a Replica have such Log File:

```
[Fri Oec 03 08:42:34 CET 2010Jon_synchronization:<A_VERSION=157/><B VERSION=98/>
```

For now, let us focus on the items between angled brackets. The only information that really matters is that such Node has an "A" table in its own database which is aligned with an "A" table with the "157" version number on a different Node dwelling somewhere on the Network (and, of course, the same reasoning can be done for the "B" table). It doesn’t necessarily mean that "A" table has actually encountered 157 modifications since its creation. In fact it is worth reminding that Log File is about the current state of the database, not its history.

To enable a fine grained interaction it keeps such information via a version number associated to a single table.

An example of log file is the following:

```
[Fri Oec 03 08:42:34 CET 2010J
on_synchronization:
<INFO_VERSION=15633/><PARTNER_VERSION=98733332/><RESOLVER_VERSION=2435672/><RESOLVE RXMLRESPONSE VERSION=2627292/>
```

Here it is possible to see three different pieces of information:

- **[Fri Oec 03 08: 42: 34 CET 2010]** is a time-stamp which acts as operation temporal reference.
- **on_synchronization** is a tag representing the operation which caused the log creation.
- **<x_VERSION=Y>**... are the database tables version number (X is the table name)

The possible operation tag are:

- **on_synchronization:** the log file is created as a synchronization procedure consequence.
- **on_update:** the log file is created as a direct update consequence (i.e. the database is modified through a non-synchronization procedure).
- **on_initialization:** the log file is created as an initialization consequence.
- **file_fixed:** the log file is generated as a recovery action consequence.

### 5.4 Persistence and Persistent Actions

In this section we focus on the database information storing and deletion system, along with the tracing of the a Node persistence state modification. By design\(^2\), within the Network context, the

\(^1\)See “Logger and Logging Operations” section

\(^2\)See PersID Architectural Model
Master Node is the main access point to the Global Resolver Service and so it should be the only Node to be direct accessed for a persistence operation. In other words, it seems sensible that a data insertion, deletion, update performed by a user should be executed on a Master rather than a Slave. If something happens to the Master, then will be chosen a Slave in the Network boundaries to take its place. Persistent operations could be performed directly on such Node, but after the state modification.

Persistent operations are performed through the PersistentService class. This abstraction maps the GRS persistent operations on one or more Hibernate operations. Since the entire set of persistent operations must be performed on every database table (represented through a class) the PersistenceService is designed as a template. As mentioned above, to preserve the system coherence only writing operations must be traced. The database modification awareness is achieved by the means of a mechanism able to intercept their execution. Aspect Oriented Paradigm can help with this task.

![Diagram of a Logging System view](image_url)

**Figure 2:** A Logging System view

Briefly, the PersistenceService class is wrapped by a proxy capable of intercepting the persistence methods calls implemented by the Spring Framework class ProxyFactoryBean. When it happens, the LogAfterReturningAdvice class is asked to trace the object, the subject and the action representing the persistent method invocation. LogAfterReturningAdvice passes these pieces of information to the Logger class which is responsible of freezing the Node state into the Log File.
In Figure 3 is shown a sequence diagram of a persistence operation execution. We can summarize the steps as follows:

1. **the PersistenceService** is requested to perform a persistence operation through the add () method invocation (please note, it is an operation which modify the database table state)
2. **the ProxyFactoryBean** intercepts the method invocation and alerts `LogAfterReturningAdvice` which processes the information extracting four key elements:
   a. the target object
   b. the method invoked
   c. the add method arguments
   d. the add method returned value
3. **the LogAfterReturningAdvice** passes the control to the Logger class, which processes the information freezing the Node modifications into the Log File.

Let us suppose there exists a database table called "resolvers" which is represented by the Resolver class. At a certain time a user decides to register a new resolver through the web interface provided with the GRSS package. She fills the form fields and hits the submit button sending the request to the responsible controller. After few calls the control is given to the `PersistenceServiceImpl` which gathers the necessary information to add the requested record to the "resolvers" database table. While the `PersistenceServiceImpl` class is requested to complete its work, the `proxyFactoryBean` awakes and alerts the `LogAfterReturningAdvice` class that a persistent operation is executing. The `afterReturning()` method is then invoked with the following arguments:

1. **the target class**: `PersistenceServiceImpl`
2. **the target class method called**: add ()
3. **the target class method arguments**: `Resolver`
4. **the target class method returned value**: `null`

Finally, the control is given to the `Logger` to represent the system state modification refreshing the Log File and to trace the persistence action on the Event Log File.

### 5.5 Logger and Logging Operations
The Logger class is the GRSS element responsible for the correct execution of logging operations. It must accomplish the following tasks:

- to parse the messages sent by the LogAfterReturningAdvice class to be aware of what is happening on the system database
- to manage the Log File:
  - writing the information representing the current system state
  - reading the information representing the current system state
  - keeping the file in a formally correct state
  - being able to detect a file corruption
  - regenerating the file if it is compromised
- to delegate the event logging operation to the EventLogger class

We will now cover all the Logger conceptual functionalities through use case scenario exemplifications.

Let us suppose the previous resolver adding operation is submitted by user request. Once the persistence operation is performed, LogAfterReturningAdvice invokes the Logger writeLog() method passing the String to be logged (containing the information discussed above) as an argument. After the method is called, the Logger verifies the Log File existence and updates the Resolver version number by unit incrementation. If the Log File does not exist it will be created and the tables version number will be initialized to a random value between 1 and 1,000,000,000. After every writing operation a check is performed on the Log File to verify its formal integrity. If the integrity check fails the file will be recovered through fixing or rebuilding procedures.

Let us suppose the reference Node is a Master which is handling an incoming synchronization request by a Slave. In the "Synchronization Procedure" section will be shown that the Slave synchronization request is responsible for a Master database tables version retrieving to decide the Slave alignment status. The function extractDBVersionFromLog() is designed to satisfy such requirement. In this example the Slave is found misaligned and the synchronization routine is run to re-establish the system coherence. Once the Slave database tables are updated, the Log File is refreshed to the Master database version via the replaceLog() Logger method.

Finally, the fileCheck() and fixLogFile() methods are dedicated to the Log File integrity preservation. Since such file is at the heart of the synchronization system it must be kept coherent as much as possible. Log File consistency is preserved through different security approaches and the following points represent the security actions applied to keep the Network consistent. Please note that the security actions are designed with the intent to satisfy the assumption declared in the Statement 1.

1. the Log File is initialized with random tables version number between 1 and 1,000,000,000
2. after a writing, the Log File is checked for correctness
3. if the Log File is missing, it is automatically regenerated via initialization
4. if the Log File is corrupted, it is automatically regenerated
   1. this case involves the accidental modification of the Log File (e.g. an user opens the file and deletes some text)

The most important thing is that the worst case is about a useless re-synchronization execution which is a very endurable drawback considering the expected database size.

**Log File Failures**

Even though the logging system is designed to handle critical issues, it is always possible to encounter a synchronization failure. At the moment, two critical events are known.

1. The Log File is initialized/rebuilt and the random number selection extracts a value present on a different Node Log File. If this event occurs, the table related to such version number will
produce a wrong successful alignment check. It is worth noting that the most probable event related to this failure (i.e. the failure of a single table) has a probability of one on a billion (the interval has been selected to keep this probability reasonably low). Moreover, within a normal system, the initialization/rebuild operations are rather uncommon events and so this can be considered a minor failure.

Please note that a random initialization/rebuild avoids several critical scenarios. For instance, let us suppose a different rule where a Node is initialized/rebuilt with version numbers = 0. Under this hypothesis it is possible the occurrence of the following critical scenario:

a. A Slave remains offline for a long time. Its Log File has the following configuration:
   
   `<A VERSION=1/>/<B VERSION=0/>/<C VERSION=0/>/<D VERSION=0/>`

b. At a time T the Master Log File has the configuration:
   
   `<A VERSION=32/>/<B VERSION=4/>/<C VERSION=5/>/<D VERSION=1/>`

c. At the time T+1 the Master encounters a problem and it is forced to rebuild its Log File. Now its Log File is in the starting configuration:
   
   `<A VERSION=0/>/<B VERSION=0/>/<C VERSION=0/>/<D VERSION=0/>`

d. At the time T+1 the Slave returns online and contact the Master for a synchronization request. The procedure outputs a wrong successful alignment for the B, C and D tables producing a system coherence failure.

2. The Master and the Slave database are modified at the same time. If this event occurs, the databases can be (probably are) misaligned, but the table version number have been incremented by one, resulting the same. If the system prescriptions are respected this event should never occur. When a Node is in the Slave state it should be updated only via synchronization request and not directly by user interaction.

5.6 Synchronization Procedure

In this section will be shown the synchronization procedure, with all the classes responsible to keep the system coherent.
First we must distinguish the client side application from the server side application. Since a Node must be able to change its own state (i.e., to become a Slave or a Master) a basic requirement is that the same software must run on a Node regardless of its role. The GRSS Synchronizer meets this requirement implementing both client and server side application. The project guidelines demand the Slave to act as a client and the Master to act as a server, so that the Slave will contact the Master for a synchronization request. The main class to rule the client side application is named Synchronizer. As shown in the sequence diagram in Figure 4, the Synchronizer class is responsible for managing many core functionalities to successfully achieve the synchronization process. We will explain a detailed step by step synchronization attempt performed by a Slave in the Network.

1. The client schedule time expires and the synchronization procedure starts
2. The client checks its state, if it is in a Master state the synchronization routine will be aborted, otherwise it will continue
3. The client (from now on the Slave) contacts the MasterVersionController via a HTTP GET request asking for database tables versions
4. If the server node is in the Master state it retrieves its own table versions, runs a serialization routine and sends them to the Slave as a JSON payload.
5. The Slave stores the Master table versions and it performs a check against its tables versions.
6. If a misalignment is detected the Slave sends via HTTP POST an alignment request to the Master's SynchronizeController asking for the misaligned tables.

Figure 4: Synchronization Sequence Diagram
7. The SynchronizeController interacts with the PersistenceService to get the requested tables and performs a JSON serialization before sending it as a response.

8. Once the response is received by the Slave, it drops its tables and rebuilds them with the data obtained in the previous exchange. When the process ends the Slave is aligned with the Master.

Since the task to accomplish is to synchronize two Nodes within a Network it is mandatory that the Nodes have the same database schema, at least for the tables to be synchronized. If this requirement is not met the system is able to detect the schema incompatibility anyway.

5.7 Configuring GRSS Package

There are two access points to configure the GRSS package.

1. the dispatcher-servlet.xml file
2. the GRSNode class

The dispatcher-servlet file is the natural configuration point in the Spring Web Applications and of course almost everything which can be configured can be set up from here. To quick start the GRSS application it is essential to set up the elements represented in Table 2.

<table>
<thead>
<tr>
<th>Bean ID</th>
<th>Attributes</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronizer</td>
<td>• synchronizeURL</td>
<td>The only value to be changed is the IP address and the port. The path to the controller should be left the same</td>
</tr>
<tr>
<td></td>
<td>• getMasterversionURL</td>
<td></td>
</tr>
<tr>
<td>dataSource</td>
<td>• url</td>
<td>They are the database connection parameters and depend on the local system</td>
</tr>
<tr>
<td></td>
<td>• username</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• password</td>
<td></td>
</tr>
<tr>
<td>logger</td>
<td>• logPath</td>
<td>It must be set to an existing file system directory. Remember to end it with the \ character (or /, depending on the OS).</td>
</tr>
<tr>
<td>eventlogger</td>
<td>• sourceLogPath</td>
<td>The sourceLogPath must be set to the logger bean logPath while the destinationLogPath can be an arbitrary existing file system directory. Remember to end it with the \ character (or /, depending on the OS).</td>
</tr>
<tr>
<td></td>
<td>• destinationLogPath</td>
<td></td>
</tr>
<tr>
<td>UpdaterOnScheduleJob</td>
<td></td>
<td>It must be a valid cron expression³</td>
</tr>
</tbody>
</table>

Table 2: GRSS Dispatcher-Servlet Configuration Parameters

The GRSNode class contains a single configuration point within the init () method. It is about the Node state setup during the initialization process. We recommend to keep a Slave default configuration to avoid the existence of multiple Master on the Network even for a limited time. Even though a user interface for the Node state modification is not present (because it goes beyond the GRSS intents) the GRSNode class exposes the setMaster () and setSlave () public methods to serve the purpose.

³For details consult http://www.quartz-scheduler.org/docs/tutoriaIIITutorialLesson06.html or http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html
5.8 GRSS Integration Details

The GRSS package is designed to fit the Global Resolver Service project and it is given with four database tables integrated:

1. Info
2. Partner
3. Resolver
4. ResolverXMLResponse

Of course, it is always possible to remove them or to add new tables. Here the instructions to add a new table are shown (the remove operation is alike). To configure the GRSS to work with a custom database table it is necessary:

- to design a class to represent the table to add (domain package), using the existing classes as model.
- to edit the PersistentObjectType (aop package) enumeration adding the class name (in uppercase).
- to edit the SynchService (service package) class declaring a new PersistentService<new table> as an attribute. Then the correspondent giveSynch () method must be written, using the existing methods as model.
- to edit the Synchronizer class declaring a new PersistentService<new_table> as an attribute, and modifying the synchronize () method following the pattern used for the existing persistent classes.
- To edit the synchronizeController class modifying the handleRequestInternal () method following the pattern used for the existing persistent classes.

It is necessary to configure the servlet-dispatcher.xml file to reflect such modifications as shown in Table 3.

<table>
<thead>
<tr>
<th>Bean ID</th>
<th>Attributes</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>session Factory</td>
<td>annotatedClasses</td>
<td>The new class must be added to the list</td>
</tr>
<tr>
<td>synchservice</td>
<td>PersistenceService_*</td>
<td>The new persistence service must be referenced</td>
</tr>
<tr>
<td>synchronizer</td>
<td>persistenceService_*</td>
<td>The new persistence service must be referenced</td>
</tr>
</tbody>
</table>

Table 3: GRSS Dispatcher-Servlet Modifications to Add a Table

If a user interface for the new table persistency management is required, it should be created, but it isn't mandatory for synchronization purposes.

Minor Details

During the implementation the following details have been noticed:

1. The Postgres Hibernate mapping encounters some problems when the annotated classes representing the tables have attributes names starting with an upper case. Since the GRS application is requested to be as much general as possible it should be considered to fix this problem (e.g. the "HowToJoin" field of the Info class has been changed in "howToJoin").
2. In the GRS application existed a misspelled table named "ResolverXMLResponce". This name has been changed in "ResolverXMLResponse".


6    Annex IV.d – URN Syntax Drafts

6.1    Draft-ah-rfc2141bis-urn-02

Abstract
Uniform Resource Names (URNs) are intended to serve as persistent, location-independent, resource identifiers. This document serves as the foundation of the 'urn' URI Scheme according to RFC 3986 and sets forward the canonical syntax for URNs, which subdivides URNs into "namespaces". A discussion of both existing legacy and new namespaces and requirements for URN presentation and transmission are presented. Finally, there is a discussion of URN equivalence and how to determine it. This document supersedes RFC 2141.

The requirements and procedures for URN Namespace registration documents are currently set forth in RFC 3406, which is also expected to be updated by an independent, revised specification.

Discussion
This draft version has been obtained by importing the text from RFC 2141 into modern tools and making a first round of updating steps. It is intended to serve as one of the starting points for an effort to bring URN RFCs in alignment with STD 63, STD 68, BCP 26, and the requirements from emerging distributed national and international URN resolution systems, and advance them on the IETF Standards Track. Comments are welcome on the urn@ietf.org mailing list (or sent to the document editor).

Status of This Memo
This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at anytime. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 2, 2010.

Copyright Notice
Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License. This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.
6.1.1 Introduction

‘urn’ is a particular URI Scheme (according to STD 63, RFC 3986 [RFC3986] and BCP 35, RFC 4395 [RFC4395]) that is dedicated to forming a hierarchical framework for persistent identifiers.

Uniform Resource Names (URNs) are intended to serve as persistent, location-independent, resource identifiers and are designed to make it easy to map other namespaces (that share the properties of URNs) into URI-space. Therefore, the URN syntax provides a means to encode character data in a form that can be sent in existing protocols, transcribed on most keyboards, etc.

The first level of hierarchy is given by the classification of URIs into "URI Schemes", and for URNs, the second level is organized into "URN Namespaces".

Historical Perspective and Motivation

For the intended audience of this RFC, which is expected to include groups interested in persistent identifiers in general and not in continuous contact with the IETF and the RFC series, this section gives a brief outline of the evolution of the matter over time. Appendix A gives hints on how to obtain RFCs and related information.

Attempts to define generally applicable identifiers for network resources go back to the mid-1970 years. Among the applicable RFCs is RFC 615 [RFC0615], which subsequently has been obsoleted by RFC 645 [RFC0645].

The seminal document in the RFC series regarding URIs (Uniform Resource Identifiers) for use with the World Wide Web (WWW) has been RFC 1630 [RFC1630], published in 1994. In the same year, the general concept or Uniform Resource Names has been laid down in RFC 1737 [RFC1737], and that of Uniform Resource Locators in RFC 1736 [RFC1736].

The original formal specification of URN Syntax, RFC 2141 [RFC2141] has been adopted in 1997. That document was based on the original specification of URLs (Uniform Resource Locators) in RFC 1738 [RFC1738] and RFC 1808 [RFC1808], which later on, in 1998, has been generalized and consolidated in the Generic URI specification, RFC 2396 [RFC2396]. Most parts of these URI/URL documents have been superseded in 2005 by STD 63, RFC 3986 [RFC3986]. Notably, RFC 2141 makes -- essentially normative -- reference to a draft version of RFC 2396.

Over time, the terms "URI", "URL", and "URN" have been refined and slightly shifted according to emerging insight and use. This has been clarified in a joint effort of the IETF and the World Wide Web Council, published 2002 for the IETF in RFC 3305 [RFC3305].

The wealth of URI Schemes and URN Namespaces needs to be organized in a persistent way, in order to guide application developers and users to the standardized top level branches and the related specifications. These registries are maintained by the Internet Assigned Numbers Authority (IANA) [IANA] at [IANA-URI] and [IANA-URN], respectively. Registration procedures for URI Schemes originally had been laid down in RFC 2717 [RFC2717] and guidelines for the related specification documents were given in RFC 2718 [RFC2718]. These documents have been obsoleted and consolidated into BCP 35, RFC 4395 [RFC4395], which is based on, and aligned with, RFC 3986.

Note that RFC 2141 predates RFC 2717 and, although the ‘urn’ URI scheme is listed in [IANA-URI] with a pointer to RFC 2141, this registration has never been performed formally.

Similarly, the URN Namespace definition and registration mechanisms originally have been specified in RFC 2611 [RFC2611], which has been obsoleted by BCP 66, RFC 3406 [RFC3406]. Guidelines for documents prescribing IANA procedures have been revised as well over the years, and
at the time of this writing, BCP 26, RFC 5226 [RFC5226] is the normative document. Neither RFC 4395 nor RFC 3406 conform with RFC 5226.

Early documents specifying URI and URN syntax, including RFC 2141, made use of an ad-hoc variant of the original Backus-Naur Form (BNF) that never has been formally specified.

Over the years, the IETF has shifted to the use of a predominant formal language used to define the syntax of textual protocol elements, dubbed "Augmented Backus-Naur Form" (ABNF). The specification of ABNF also has evolved, and now STD 68, RFC 5234 [RFC5234] is the normative document for it (that also will be used in this RFC).

**Objective of this RFC**

RFC 2141 does not seamlessly match current Internet Standards. The primary objective of this document is the alignment with the URI Standard [RFC3986] and guidelines [RFC4395], the ABNF Standard [RFC5234] and the current IANA Guidelines [RFC5226] in general. Further, experience from emerging international efforts to establish a general, distributed, stable URN resolution service are expected to be taken into account during the draft stage of this document.

For advancing the URN specification on the Internet Standards-Track, it needs to be based on documents of comparable maturity. Therefore, to further advancements of the formal maturity level of this RFC, it deliberately makes normative references only to documents at Full Standard or Best Current Practice level.

Thus, this replacement document for RFC 2141 should make it possible to advance the URN framework step by step on the Internet Standard maturity ladder. All other related documents depend on it; therefore this is the first step to undertake.

Out of scope for this document is a revision of the URN Namespace Definition Mechanisms document, BCP 66 [RFC3406].

**Requirement Language**
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

### 6.1.2 URN Syntax

This document defines the URI Scheme ‘urn’. Hence, URNs are specific URIs as specified in RFC 3986 [RFC3986]. The formal syntax definitions below are given in ABNF according to RFC 5234 [RFC5234] and make use of some "Core Rules" specified in Appendix B of that Standard and several generic rules defined in Appendix A of RFC 3986.

The syntax definitions below do, and syntax definitions in dependent documents MUST, conform to the URI syntax specified in RFC 3986, in the sense that additional syntax rules must only constrain the general rules from RFC 3986. In other words: a general URI parser based on RFC 3986 MUST be able to parse any legal URN, and specific semantics can be obtained from URN-specific parsing.

**NOTE:** The remainder of this section still requires MUCH work!

URNs conform to the <path-rootless> variant of the general URI syntax specified in Section 3 of [RFC3986]:

```plaintext
URI = scheme "::" path-rootless [ "?" query ] [ "#" fragment ]
path-rootless = segment-nz *( "/" segment )
```
segment-nz = 1*pchar
segment = *pchar

pchar = unreserved / pct-encoded / sub-delims / ":" / "@"

with

scheme = "urn"

and the following additional syntax rule superimposed on <pathrootless> to establish a level of hierarchy called "Namespace":

urn-path = NID ":" NSS

Here "urn" is the URI scheme name, <NID> is the Namespace Identifier, and <NSS> is the Namespace Specific String. The colons are REQUIRED separator characters.

Per RFC 3986, the URN Scheme name (here "urn") is case-insensitive.

The Namespace ID (also a case-insensitive string) determines the syntactic structure and the semantic interpretation of the Namespace Specific String. Generic details on NID syntax can be found below in Section 2.1 and the NSS syntax is elaborated upon in Section 2.2.

Each particular namespace is based on a specific document that must normatively describe (among other things) the details of the <NSS> values allowed in conjunction with the respective <NID>. The specification requirements and registration procedures for URN namespaces are the subject of a dedicated document, currently RFC 3406 [RFC3406] -- to be updated for conformance with BCP 26 and alignment with implementation experience.

Note (to be discussed):

RFC 2141 has deferred the decision on whether <query> and <fragment> components are applicable to URNs and reserved the use of bare (unencoded) question mark ("?") and hash ("#") characters in URNs.

There is evidence of desire to be able to use these components (which are split off by the high-level parsing rules of RFC 3986), or at least the <fragment> component, in URNs belonging to selected namespaces. Thus, this draft version tentatively aims at allowing these components in the general syntax. These components however shall only be allowed if and only if the specification document for a particular URN namespace specifically does say so and discusses the ramifications of this addition.

Question mark and hash sign remain reserved for this purpose and cannot appear unencoded in an NSS. This way, backwards compatibility with existing URN namespaces is guaranteed and compatibility with general URI parsers is improved.

**Namespace Identifier Syntax**

The following is the syntax for the Namespace Identifier. To (a) be consistent with all potential resolution schemes and (b) not put any undue constraints on any potential resolution scheme, Namespace Identifiers are ASCII strings with the syntax:

NID = ( ALPHA / DIGIT ) 0*31 ( ALPHA / DIGIT / ":" )

Namespace Identifiers are case-insensitive, so that for instance "ISBN" and "isbn" refer to the same namespace.
To avoid confusion with the URI Scheme name "urn", the NID "urn" is permanently reserved by this RFC and MUST NOT be used or registered.

**Namespace Specific String Syntax**

Note: In order to make visible the migration path from RFC 2141 and the influence of the evolution of URI syntax from RFC 2396 to RFC 3986 on it, at this draft stage, the subsequent syntax description is highly annotated and expanded. After discussion, a substantial consolidation is expected.

As already required by RFC 1737, there is a single canonical representation of the NSS portion of an URN.

Note: If the DISCUSSes above and below can be affirmed (allowing optional <question> and <fragment> components as well as "&" and "~" in the path), the syntax could be simplified very much to:

NSS = 1*pchar ; or equivalent: NSS = segment-nz

The format of this single canonical form follows:

NSS = 1*URN-char
URN-char = trans / pct-encoded
trans = ALPHA / DIGIT / u-other
u-other = ";" / "@"
; From RFC 3986 <gen-delims>
; specifically allowed in <pchar>.
; From RFC 3986:
; gen-delims = ";" / "/" / "?" / "#" / "[" / "]" / "@"
/ "!" / "$" / "~" / "('" / ")"/ "*" / "+" / "," / ";" / "="
; this is RFC 3986 <sub-delims> except ";&".

; From RFC 3986:
; sub-delims = ";!" / "$" / ";&" / "=" / "('" / ")"
/ "*" / "+" / "," / ";" / "==

; Issue: can/should ";&" be allowed ?
; If we allow <question> and <fragment> according to the
generic URI syntax, there seems to be no more need to exclude ";&".

NSS = 1*URN-char
URN-char = trans / pct-encoded
trans = ALPHA / DIGIT / u-other
u-other = ";" / "@"
; From RFC 3986 <gen-delims>
; specifically allowed in <pchar>.
; From RFC 3986:
; gen-delims = ";" / "/" / "?" / "#" / "[" / "]" / "@"
/ "!" / "$" / "~" / "('" / ")"/ "*" / "+" / "," / ";" / "="
; this is RFC 3986 <sub-delims> except ";&".

; From RFC 3986:
; sub-delims = ";!" / "$" / ";&" / "=" / "('" / ")"
/ "*" / "+" / "," / ";" / "==

; Issue: can/should "~" be allowed ?
; If we allow <question> and <fragment> according to the
generic URI syntax, there seems to be no more need to exclude "~".

NSS = 1*URN-char
URN-char = trans / pct-encoded
trans = ALPHA / DIGIT / u-other
u-other = ";" / "@"
; From RFC 3986 <gen-delims>
; specifically allowed in <pchar>.
; From RFC 3986:
; gen-delims = ";" / "/" / "?" / "#" / "[" / "]" / "@"
/ "!" / "$" / "~" / "('" / ")"/ "*" / "+" / "," / ";" / "="
; this is RFC 3986 <sub-delims> except ";&".

; From RFC 3986:
; sub-delims = ";!" / "$" / ";&" / "=" / "('" / ")"
/ "*" / "+" / "," / ";" / "==

; Issue: can/should "~" be allowed ?
; If we allow "&" and "~", <trans> becomes <pchar>,
greatly simplifying the syntax rules and parsers!

NSS = 1*URN-char
URN-char = trans / pct-encoded
trans = ALPHA / DIGIT / u-other
u-other = ";" / "@"
; From RFC 3986 <gen-delims>
; specifically allowed in <pchar>.
; From RFC 3986:
; gen-delims = ";" / "/" / "?" / "#" / "[" / "]" / "@"
/ "!" / "$" / "~" / "('" / ")"/ "*" / "+" / "," / ";" / "="
; this is RFC 3986 <sub-delims> except ";&".

; From RFC 3986:
; sub-delims = ";!" / "$" / ";&" / "=" / "('" / ")"
/ "*" / "+" / "," / ";" / "==

; Issue: can/should "~" be allowed ?
; If we allow "&" and "~", <trans> becomes <pchar>,
greatly simplifying the syntax rules and parsers!

NSS = 1*URN-char
URN-char = trans / pct-encoded
trans = ALPHA / DIGIT / u-other
u-other = ";" / "@"
; From RFC 3986 <gen-delims>
; specifically allowed in <pchar>.
; From RFC 3986:
; gen-delims = ";" / "/" / "?" / "#" / "[" / "]" / "@"
/ "!" / "$" / "~" / "('" / ")"/ "*" / "+" / "," / ";" / "="
; this is RFC 3986 <sub-delims> except ";&".

; From RFC 3986:
; sub-delims = ";!" / "$" / ";&" / "=" / "('" / ")"
/ "*" / "+" / "," / ";" / "==

; Issue: can/should "~" be allowed ?
; If we allow "&" and "~", <trans> becomes <pchar>,
greatly simplifying the syntax rules and parsers!
Depending on the rules governing a namespace, valid identifiers in a namespace might contain characters that are not members of the URN character set above (<URN-char>). Such strings MUST be translated into canonical NSS format before using them as protocol elements or otherwise passing them on to other applications. Translation is done by encoding each character outside the URN character set as a sequence of octets using UTF-8 encoding [RFC3629], and the "percent-encoding" of each of those octets as "%" followed by two <HEXDIG> characters. The two characters give the hexadecimal representation of that octet.

**Special and Reserved Characters**

The remaining printable characters left to be discussed above comprise the generic delimiters and the reserved characters, which are restricted for special use only. These characters are discussed below, giving the specifics of why each character is special or reserved.

**Delimiter Characters**

RFC 3986 [RFC3986] defines the general delimiter characters used in URIs:

```
gen-delims = ":" / "/" / ":?" / ":#" / ":[" / ":]" / ":@"
```

From among the <gen-delims>, ":" and ":@" are also included in <pchar> and hence allowed in the path components of URIs.

The at-character (":@") in generic URIs only has a specific meaning when contained in the <authority> part, which is absent in URNs. Hence, ":@" is available in the <NSS> part of URNs.

With URNs, the colon (":") is used as a delimiter character not only between the scheme name (":urn") and the <NID>, but also between the latter and the <NSS>, and many existing URN namespaces additionally use ":" to further subdivide a single RFC 3986 path segment in the <NSS> in a hierarchical manner.

**Note:** Using ":" as a sub-delimiter in the path in favor of "/" is attractive because it avoids possible complications that could arise from the inappropriate use of relative URI references [RFC3986] for URNs.

The characters "/", ":?", and ":#" separate path components and the <question> and <fragment> parts in the generic URI syntax; they are restricted to this role in URNs as well, although the <path> in URNs only admits a single <segment> and hence "/" is not allowed. Therefore, these characters MUST NOT appear in the <NSS> part of a URN in unencoded form. Namespaces that need these characters MUST employ in their URNs the appropriate percent-encoding for each character.

The square brackets ("[" and "]") also play a particular role when contained in the <authority> part, which is absent in URNs. However, for conformance with the generic URI syntax, they are not allowed literally in the <NSS> component of URNs. If a specific URN namespace reflects semantics that require these characters, they MUST be percent-encoded in the respective URNs.

**The '%' character**

The "%" character is reserved in the URN syntax for introducing the escape sequence for an octet that is either not a printable ASCII character or reserved for special purposes, as described in this section. Literal use of the "%" character in an underlying namespace must be encoded as "%25" in URNs for that namespace. The presence of a "%" character in a URN MUST always be followed by two <HEXDIG> characters, which three together semantically form an abstract <pctencoded> octet.

Namespaces MAY designate one or more characters from the URN character set as having special meaning for that namespace. If the namespace also uses that character in a literal sense as well, the character used in a literal sense MUST be encoded with "%" followed by the hexadecimal
representation of that octet. Further, a character MUST NOT be percent-encoded if the character is not a reserved character. Therefore, the process of registering a namespace identifier shall include publication of a definition of which characters have a special meaning to that namespace.

Other Excluded Characters
The following list is included only for the sake of completeness. It includes the characters discussed in Sections 2.3.1 and 2.3.2. Any octets/characters on this list are explicitly NOT part of the URN <NSS> character set, and if used in an URN, MUST be percent-encoded.

```
excluded = CTL / SP           ; control characters and space
         / DQUOTE               ; "
         / "#"                  ; from <gen-delims>
         / "%"                 ; see above
         ; DISCUSS!
         / "&"                  ; DISCUSS -- see above!
         / "/"                   ; from <gen-delims>
         / "<<" / ">"            ; DISCUSS -- see above!
         / "?"                  ; from <gen-delims>
         / "["                   ; from <gen-delims>
         / "\"                   ; from <gen-delims>
         / "]"                   ; from <gen-delims>
         / "%^"                  ; from <gen-delims>
         ; DISCUSS!
         / "\"                   ; DISCUSS -- see above!
         / %x7F                  ; DEL (control character)
         / %x80-FF               ; non-ASCII
```

In addition, the NUL octet (0 hex) SHOULD never be used, in either unencoded or percent-encoded form.

In textual context, a URN ends when an octet/character from the excluded character set (<excluded>) is encountered. The character from the excluded character set is NOT part of the URN. [ Does that still make sense? -- it collides with possible question / fragment! ]

6.1.3 Support of Existing Legacy Naming Systems and New Naming Systems

Any namespace (existing or newly devised) that is proposed as a URN namespace and fulfills the criteria of URN namespaces MUST be expressed in this syntax. If names in these namespaces contain characters other than those defined for the URN character set, they MUST be translated into canonical form as discussed in Section 2.2.

6.1.4 URN Presentation and Transport

The URN syntax defines the canonical format for URNs and all URN transport and interchanges MUST take place in this format. Further, all URN-aware applications MUST offer the option of displaying URNs in this canonical form to allow for direct transcription (for example by cut-and-paste techniques). Such applications MAY support display of URNs in a more human-friendly form and may use a character set that includes characters that aren’t permitted in URN syntax as defined in this RFC (that is, they may replace %-notation by characters in some extended character set in display to humans).

6.1.5 Lexical Equivalence in URNs
For various purposes such as caching, it is often desirable to determine whether two URNs are the same without resolving them. The general purpose means of doing so is by testing for "lexical equivalence" as defined below.

Two URNs are lexically equivalent if they are octet-by-octet equal after the following preprocessing:

1. normalize the case of the leading "urn" scheme;
2. normalize the case of the NID;
3. normalize the case of any percent-encoding.

Note that percent-encoding MUST NOT be removed.

Some namespaces may define additional lexical equivalences, such as case-insensitivity of the NSS (or parts thereof). Additional lexical equivalences MUST be documented as part of namespace registration, MUST always have the effect of eliminating some of the false negatives obtained by the procedure above, and MUST NEVER say that two URNs are not equivalent if the procedure above says they are equivalent.

**Examples of Lexical Equivalence**
The following URN comparisons highlight the lexical equivalence definitions:

1. URN:foo:a123,456
2. urn:foo:a123,456
3. urn:FOO:a123,456
4. urn:foo:A123,456
5. urn:foo:a123%2C456
6. URN:FOO:a123%2c456

URNs 1, 2, and 3 are all lexically equivalent. URN 4 is not lexically equivalent to any of the other URNs of the above set. URNs 5 and 6 are only lexically equivalent to each other.

6.1.6 **Functional Equivalence in URNs**

Functional equivalence is determined by practice within a given namespace and managed by resolvers for that namespace. Thus, it is beyond the scope of this document. Namespace registrations must include guidance on how to determine functional equivalence for that namespace, i.e. when two URNs are identical within a namespace.

6.1.7 **The ‘urn’ URI Scheme**

At the time of publication of RFC 2141, no formal registration procedure for URI Schemes had been established yet, and so IANA only informally has registered the ‘urn’ URI Scheme with a reference to [RFC2141].

The section below contains the URI scheme registration template for the ‘urn’ scheme, in accordance with RFC 4395 [RFC4395].

Note: In order to be useable as a standalone text (after being extracted from this RFC), the template below does not contain formal anchors to the references listed in section 11, but instead gives to common RFC designations in prose. However, for compliance with editorial policy, it needs to be noted:

This registration template refers to RFCs 2196, 2276, 2608, 3401 through 3404, and 3406 [RFC2169] [RFC2276] [RFC2608] [RFC3401][RFC3402] [RFC3403] [RFC3404] [RFC3406].
Registration of URI Scheme ‘urn’
[ RFC Editor: please replace "XXXX" in all instances of "RFC XXXX" below by the RFC number assigned to this document. ]

URI scheme name: urn

Status: permanent

URI scheme syntax:

See Section 2 of RFC XXXX.

URI scheme semantics:

‘urn’ URIs, known as Universal Resource Names (URNs), serve as persistent, location-independent, resource identifiers for concrete and abstract objects that have network accessible instances and/or metadata.

URNs are structured hierarchically into URN Namespaces, the management of which is delegated to namespace-specific authorities. Each such URN namespace is founded in an independent specification and registered with IANA, following the guidelines and procedures of BCP 66 (at the time of this registration: RFC 3406).

Encoding considerations:

All URNs are ASCII strings conforming to the general URI syntax from STD 66. As described in Sections 2.2 and 2.3.2 of RFC XXXX, characters needed by the URN namespace specific semantics but not contained in the US-ASCII charset MUST be encoded in UTF-8 according to STD 63; any octets outside the allowed character set MUST then be percent-encoded.

Applications/protocols that use this URI scheme:

URNs that serve to identify abstract resources for protocol purposes are expected to be recognized directly by the implementations of these protocols.

In general, resolution systems for URNs are specified on a per-namespace basis. If appropriate for the namespace, these systems resolve URNs to (possibly multiple) URIs that allow the network access to the identified object or metadata on it.

"Architectural Principles of Uniform Resource Name Resolution" (RFC 2276) explains the basic concepts. Some resolution systems laid down in IETF specifications are:

* Trivial HTTP-based URN Resolution (RFC 2169)
* Dynamic Delegation Discovery System (DDDS, RFCs 3401-3404)
* Service Location Protocol (SLPv2, RFC 2608)

Interoperability Considerations:

Persistence and stability of URNs require appropriate resolution systems.

Security Considerations:
See Section 8 of RFC XXXX.

Contact:

Provisionally: the authors of this draft.
This registration will be discussed on the following IETF lists: uri-review and urn.
It is expected that a "URNbis" WG be formed in the IETF and take over control of this
document, and that subsequently the Chairs and mailing list of that WG serve as the
primary contact.

Author / Change controller:

The authors of this draft.
Change control is with the IESG.

References:

RFC XXXX.

Procedures for the specification and registration of URN namespaces are detailed in BCP 66
(at the time of this writing: RFC 3406; a rfc3406-bis document is expected as a deliverable
of the proposed "URNbis" WG).

6.1.8 Security Considerations

This document specifies the syntax and general requirements for URNs, which are the specific URIs
that use the 'urn' URI scheme. As such, the general security considerations of STD 66 [RFC3986]
apply. However, each URN namespace will have specific security considerations, according to the
semantics and usage of the underlying namespace. While some namespaces may assign special
meaning to certain of the characters of the Namespace Specific String, any security considerations
resulting from such assignment are outside the scope of this document. It is REQUIRED by BCP 66
[RFC3406] that the process of registering a namespace identifier include any such considerations.

6.1.9 IANA Considerations

IANA is asked to update the existing informal registration of the 'urn' URI Scheme by the template
in Section 7.1 above and list this RFC as the current normative reference in [IANA-URI].

IANA is asked to add a note to [IANA-URN] that 'urn' is a permanently reserved formal namespace
identifier string that cannot be registered, in order to avoid confusion with the 'urn' URI scheme.

6.1.10 Acknowledgements

This document is heavily based on RFC 2141, the author of which has laid the foundation for this
work, and which contained the following Acknowledgements:

Thanks to various members of the URN working group for comments on earlier drafts of
this document. This document is partially supported by the National Science Foundation,
Cooperative Agreement NCR-9218179.

This document also heavily relies on and acknowledges the work done for STD 66 [RFC3986].

Your name could go here ...
6.1.11 References

Normative References


Informative References


6.1.12 Appendix A - How to Locate IETF Documents (Informative)

Request For Comments (RFCs) are available from the RFC Editor site using the canonical URIs http://www.rfc-editor.org/rfc/rfcNNNN.txt or <ftp://ftp.rfc-editor.org/in-notes/rfcNNNN.txt> (where 'NNNN' is the serial number of the RFC), and from numerous mirror sites.

Additional metadata for any RFC, including possible Errata, are available from <http://www.rfc-editor.org/info/rfcNNNN> (where 'NNNN' again is the serial number of the RFC). A HTML-ized version and a PDF facsimile of each RFC are available from the IETF Tools site at <http://tools.ietf.org/http/rfcNNNN> and <http://tools.ietf.org/pdf/rfcNNNN>, respectively.

Current Internet Draft documents are available via the search engines at <http://www.ietf.org/id-info/> and <http://www.rfc-editor.org/idsearch.html>; archival copies of older IETF documents can be found at <http://tools.ietf.org/id/>. 
6.1.13 Appendix B - Handling of URNs by URL Resolvers/Browsers

The URN syntax has been defined so that URNs can be used in places where URLs are expected. A resolver that conforms to the current URI syntax specification [RFC3986] will extract a scheme value of "urn" rather than a scheme value of "urn:<nid>".

An URN MUST be considered an opaque URI by URL resolvers and passed (with the "urn:" tag) to an URN resolver for resolution. The URN resolver can either be an external resolver that the URL resolver knows of, or it can be functionality built into the URL resolver.

To avoid confusion of users, an URL browser SHOULD display the complete URN (including the "urn:" tag) to ensure that there is no confusion between URN namespace identifiers and URI scheme names.

6.1.14 Appendix C - Collected ABNF (Informative)

As a service to implementers specifically interested in URN syntax, after consolidation of Section 2, the complete ABNF for URNs will be collected here, including the referenced rules from [RFC5234] and [RFC3986]. In case of (unexpected) inconsistencies, these documents remain normative for the respective productions.

T.B.D.

...

6.1.15 Appendix D - Changes since RFC 2141 (Informative)

D.1. Essential Changes from RFC 2141

[D.1 headline and all subsequent subsections starting with Appendix D.2.]

T.B.D. (after consolidation of this memo)

D.2. Changes from RFC 2141 to draft -00

Abstract amended: URI scheme, replacement for 2141, point to 3406. Use contemporary boilerplate. Added transient "Discussion" section.

s1: added new 1st para (URI scheme) and 3rd para (hierarchy).

s1.1 (Historical Perspective) added for background & motivation.

s1.2 (Objective) added.

s1.3 (2119 keywords) added -- used now throughout normative text.

s2 (URN Syntax): Shifted from BNF to ABNF; explain relationship to 3986 and gaps, how the gaps could be bridged, distinguish between URI generics and URN specifics; got rid of references to immature documents (1630, 1737).

s2.1 (NID syntax): Use ABNF and RFC 5234 terminals (core rules); removed reference to an old draft of 2396; clarified prohibition to use "urn" as NID.

s2.2 (NSS syntax): Shifted from BNF to ABNF; made ABNF consistent with subsequent textual description; exposition much expanded, showing relationship with 3986 and resulting incompatibilities; proposed how to bridge gaps, to make parsing more uniform among URIs; updated i18n considerations and pointer to UTF-8 specification.
s.2.3, s2.3.*: reworked and much expanded, along the grouping of delimiter characters from 3986 in new s2.3.1 (including old s.2.3.2); made text fully consistent with ABNF in s2.2; consistent usage of term "percent-encoded"; old s.2.3.1 became s2.3.2; old s3.4 became s3.3.3, providing complete, annotated list of excluded characters, ordered by ascending code point; and restating design decisions needed to be made to close gaps to 3986.

s3 through s6: only minor editorial changes.

s7: formal registration of ‘urn’ URI scheme added, using 4395 template.

s8: Security Cons. slightly amended.

s9: new: IANA Cons. added wrt s7.1 and prohibition of NID "urn".

s10: Acknowledgments amended.

s11: References split into Normative and Informative; updated refs and added many; only FS and BCP allowed as Normative Refs to further promotion of document.

Added Appendices A through D.

D.3. Changes from draft-00 to draft-02

Updated "Discussion" on front page to point to dedicated urn list.

Numerous editorial improvements and additions for clarification, in particular in the Introduction. No technical changes.

More Informative References; missing details supplied in D.1.

Author’s Address
Alfred Hoenes (editor)
TR-Sys
Gerlinger Str. 12
Ditzingen D-71254
Germany
EMail: ah@TR-Sys.de

6.2 Draft-hakala-rfc3187bis-isbn-urn-00

Abstract
The International Standard Book Number, ISBN, is a widely used identifier for monographic publications. Since 2001, there has been a URN (Uniform Resource Names) namespace for ISBns. The namespace registration was performed in RFC 3187 and applies to the ISBN as specified in the original ISO Standard 2108-1992. To allow for further growth in use, the successor ISO Standard, ISO 2108-2005, has defined an expanded format for the ISBN, known as "ISBN-13". This document replaces RFC 3187 and defines how both the old and new ISBN standard can be supported within the URN framework and the syntax for URNs defined in RFC 2141. An updated namespace registration is included, which describes how both the old and the new ISBN format can share the same namespace.
Discussion
This draft version is the outcome of work started in 2008 and brought to the IETF as a contribution to a much larger effort to revise the basic URN RFCs, in order to bring them in alignment with the current URI Standard (STD 63, RFC 3986), ABNF, and IANA guidelines, and to establish a modern URN resolution system for bibliographic identifiers.

Until a more specific mailing list is established, comments are welcome on the urn-nid@ietf.org mailing list (or sent to the authors).

Status of This Memo
This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 23, 2010.

Copyright Notice
Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

6.2.1 Introduction
One of the basic permanent URI schemes (cf. RFC 3986 [RFC3986], [IANA-URI]) is 'URN' (Uniform Resource Name) as defined in RFC 2141 [RFC2141]. Any identifier, when used within the URN system, needs its own namespace. As of this writing, there are 40 registered URN namespaces (see [IANA-URN]), one of which belongs to ISBN, International Standard Book Number, as specified 2001 in RFC 3187 [RFC3187].
Since 2007, there have been two variants of ISBN in use; an outdated one based on ISO 2108 1992 [ISO1] and a new one defined in ISO 2108-2005 [ISO2]. These versions shall subsequently be called "ISBN-10" and "ISBN-13", respectively, in this document. For the time being, both ISBNs may still be printed on a book, but the ISBN-13 is the actual identifier. If what is said in this document applies to both ISBN versions, the term "ISBN" is used.

As part of the validation process for the development of URNs, the IETF URN working group agreed that it is important to demonstrate that a URN syntax proposal can accommodate existing identifiers from well established namespaces. One such infrastructure for assigning and managing names comes from the bibliographic community. Bibliographic identifiers function as names for objects that exist both in print and, increasingly, in electronic formats. RFC 2288 [RFC2288] investigated the feasibility of using three identifiers (ISBN, ISSN and SICI) as URNs, with positive results, however it did not formally register corresponding URN namespaces. This was in part due to the still evolving process to formalize criteria for namespace definition documents and registration, consolidated later in the IETF into RFC 3406 [RFC3406].

URN Namespaces have subsequently been registered for both ISBN and ISSN in RFCs 3187 [RFC3187] and 3044 [RFC3044], but not for SICI, due to both the identifier's limited popularity and its complicated URN resolution process.

Guidelines for using ISBN-10s (based on ISO 2108-1992) as URNs and the original namespace registration have been published in RFC 3187 [RFC3187]. The RFC at hand replaces RFC 3187; sections related to ISBN-13 have been added, all ISBN-10 information has been updated and the namespace registration revised to make it compliant with both ISBN versions and stipulations of RFC 3406 [RFC3406], which has replaced RFC 2611 [RFC2611] applied in the initial registration.

6.2.2 Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].


6.2.3 Identification and Resolution

As a rule, ISBNs identify finite, manageably-sized objects, but these objects may still be large enough that resolution into a hierarchical system is appropriate. The materials identified by an ISBN may exist only in printed or other physical form, not electronically. In such a case, the URN:ISBN resolver should nevertheless be able to supply bibliographic data, possibly including information about where the physical resource is stored in the owning institution's holdings. There may be other resolution services supplying a wide variety of information resources or services related to the identified books.

National libraries shall be among the organizations providing persistent URN resolution services for monographic publications, independent of their form.
6.2.4 **International Standard Book Numbers**

**Overview**

**ISBN-10 Structure**
The ISBN-10 is defined by the ISO Standard 2108-1992 [ISO1]. It is a ten-digit number (the last digit can be the letter “X” as well) which is divided into four variable length parts usually separated by hyphens when printed. The parts are as follows (in this order):
- a group identifier which specifies a group of publishers, based on national, geographic, or some other criteria;
- the publisher identifier;
- the title identifier; and
- a modulo 11 check digit, using X instead of 10; the details of the calculation are specified in [ISO1].


**ISBN-13 Structure**
ISBN-13 is defined by the ISO Standard 2108-2005 [ISO2]. The ISBN-13 is a thirteen-digit number which is divided into five parts usually separated by hyphens when printed. The first and the last part have a fixed length, but the other parts have variable length. These parts are as follows (in this order):
- a prefix element of ISBN-13 is a 3 digit prefix (at the time of this writing, legal values were 978 and 979; future versions of the standard may define additional values) specified by the International ISBN Agency;
- a registration group element that specifies the registration group; it identifies the national, geographic, language, or other such grouping within which one or more ISBN Agencies operate;
- the registrant element;
- the publication element; and
- a modulo 10 check digit; the details of the calculation are specified in [ISO2].

The structural differences between the ISBN-10 and ISBN-13 are the prefix element (which did not exist in the old ISBN) and the check digit calculation algorithm, which was modulo 11 in ISBN-10 and is now modulo 10.

Terminology in ISBN-10 differs substantially from the terminology applied in ISBN-13. In this document, ISBN-13 terminology shall be used from now on; for a reader used to ISBN-10 terminology, the following mapping may be useful:
- the group identifier = the registration group element;
- the publisher identifier = the registrant element;
- the title identifier = the publication element.

Encoding Considerations
Embedding ISBNs within the URN framework does not present encoding problems, since all of the characters that can appear in an ISBN are valid in the namespace-specific string (NSS) part of the URN. %-encoding, as described in RFC 2141 [RFC2141], is never needed.


Resolution of ISBN-based URNs

General
For URN resolution purposes, all elements except the check digit (now 0-9, previously 0-9 or X) must be taken into account. The registration group and registrant element assignments are managed in such a way that the hyphens are not needed to parse the ISBN unambiguously into its constituent parts. However, the ISBN is normally transmitted and displayed with hyphens to make it easy for humans to recognize these elements without having to make reference to or have knowledge of the number assignments for registration group and registrant elements. In ISBN-10, registration group element codes such as 91 for Sweden were unique. In ISBN-13 only the combination of prefix and registration group element is guaranteed to be unique. 978-951 and 978-952 both mean Finland, but 979-951 and 979-952 almost certainly will not; registration group element(s) for Finland are not yet known for ISBNs starting with 979.

The Finnish URN registry is maintained by the national library. The service is capable of resolving ISBN-based URNs. URNs starting with URN:ISBN:978-951 or URN:ISBN:978-952 are mapped into appropriate URL addresses in a table maintained within the registry. Applications, such as the national bibliography or the open archive of a university, can use the URN as the address of the resource. There is just one place (the registry) where the location information must be kept up to date.

ISBN-13 prefix / registration group element combinations (and the corresponding ISBN-10 registration group identifiers, if any) usually designate a country, but occasionally a single combination / ISBN-10 group identifier is used to indicate a language area. For instance, “978-3” (or “3” in ISBN-10) is utilised in Germany, Austria, and the German speaking parts of Switzerland. As of this writing, there are two regional registration groups: “978-976” is used in the Caribbean community and “978-982” in the South Pacific (see [PREFIX]).

Note that the prefix and registration group element combination “979-3” has not yet been assigned. There is no intention to allocate the registration group elements in the same way as was done with ISBN-10.

The registrant element may or may not be used for resolution purposes, depending on whether individual publishers have set up their resolution services.

The publication element shall enable targeting the individual publication.

Practical Aspects
Due to the lack of URN support in, e.g., web browsers, the URNs are usually expressed as URLs when embedded in documents. The Finnish URN registry is located at http://urn.fi, and URNs are therefore expressed in the form http://urn.fi/<URN>. For example, the URN http://urn.fi/URN:ISBN:978-952-10-3937-9 identifies Sami Nurmi’s doctoral dissertation “Aspects of Inflationary Models at Low Energy Scales”.

The Finnish URN registry can not resolve URN:ISBNs with non-Finnish registration group element values until other countries establish their registries, and all these services become aware of each
other and their respective registration group responsibility domains and are able to communicate with each other. Thus the Finnish registry can deal with URN:ISBN instances with registration group element value 91 (indicating Sweden) if and only if the Swedish registry exists, its address is known to the Finnish peer and the Swedish service is capable of receiving and processing requests from other registries.

If a registration group element does not identify a single country but a language area, there are at least two means for locating the correct national bibliography. First, it is possible to define a cascade of URN registries - for instance, German, Austrian and Swiss national registries, in this order - which should collectively be aware of resolution services such as national bibliographies for ISBN-13s starting with "978-3". If the German registry is not able to find an authoritative resolution service, the request could be passed to the Austrian one, and if there are still no hits, finally to the Swiss service.

Second, the registrant element ranges assigned to the publishers in Germany, Austria and Switzerland by the ISBN Agencies could be defined directly into the national registries. This method would be more efficient than cascading, since the correct resolution service would be known immediately. The choice between these two and possible other options should be made when the establishment of the European network of URN registries reaches this level of maturity.

In some exceptional cases -- notably in the US and in the UK, where international companies do a significant portion of publishing -- the information provided by the group identifier may not always be fully reliable. For instance, some monographs published in New York by international publishing companies may get an ISBN with the registration group element "3". This is technically appropriate when the headquarters or one of the offices of the publisher is located in Germany.

Information about such a book may not always be available in the German national bibliography, but via the Library of Congress systems. Unfortunately, the German/Austrian/Swiss URN registries that should in this case be contacted may not be aware of the appropriate resolution service.

However, the problem posed by the international publishers may well be less severe than it looks. Some international publishers (Springer, for example) give the whole production to the national library of their home country as legal deposit, no matter which country the book was published. Thus everything published by Springer in New York with registration group element "3" should be resolvable via the German national bibliography. On the other hand, when these companies give their home base also as a place of publication, the "home" national library requires the legal deposit.

A large union catalogue, such as WorldCat maintained by OCLC [OCLC-WC] could be used to complement the resolution services provided in the national level, or as the default service, if no national services exist or are known to the registry from which the query originates.

Due to the semantic structure of ISBN-13, even the registrant element number of URN resolution services maintained by different kinds of organizations. For instance, "978-951-0" is the unique ISBN registrant element of the largest publisher in Finland, Sanoma-WSOY. Resolution requests for ISBNs starting with "978-951-0" can be passed to and dealt with the publisher's server, if and when it is made URNaware. In such a case, resolving the same URN in multiple locations may provide different services; the national bibliography may be able to provide bibliographic information only, while the publisher can also provide the book itself, on its own terms. Different resolution services may co-exist and complement one another. Same ISBN may be resolved both as URN and as a Digital Object Identifier (DOI) [DOIHOME]. URN-based services hosted by, e.g., a national library, might provide only bibliographic data, whereas a service based on the DOI system provided by the publisher might deliver the book, parts of the book or various services related to the work.
Persistence of resolution services is largely dependent on persistence of organizations providing them. Thus some services, independent on base technology chosen, may disappear or their content may change much sooner than some peer solutions.

**Additional Considerations**

The basic guidelines for assigning ISBNs to electronic resources are the following:

- Format/means of delivery is irrelevant to the decision whether a product needs an ISBN or not. If the content meets the requirement, it gets an ISBN, no matter what the format of the delivery system.

- Each format of a digital publication should have a separate ISBN. The definition of a new edition is normally based on one of the two criteria:
  - A change in the kind of packaging involved: the hard cover edition, the paperback edition and the library-binding edition would each get a separate ISBN. The same applies to different formats of digital files.
  - A change in the text, excluding packaging or minor changes such as correcting a spelling error. Again, this criterion applies regardless of whether the publication is in printed or in digital form.

Although these rules seem clear, their interpretation may vary. As RFC 2288 [RFC2288] points out.

The choice of whether to assign a new ISBN or to reuse an existing one when publishing a revised printing of an existing edition of a work or even a revised edition of a work is somewhat subjective. Practice varies from publisher to publisher (indeed, the distinction between a revised printing and a new edition is itself somewhat subjective). The use of ISBNs within the URN framework simply reflects these existing practices. Note that it is likely that an ISBN URN may resolve to many instances of the work (many URLs).

These instances may be fully identical, or there may be some minor differences between them. Publishers have also in some occasions reused the same ISBN for another book. This reasonably rare kind of human error does not threaten or undermine the value of the ISBN system as a whole. Neither do they pose a serious threat to the URN resolution service based on ISBNs. An error should only lead into the retrieval of two or more bibliographic records describing two different monographic publications. Based on the information in the records, a user can choose the correct record from the result set.

Most national bibliographies and especially the Books in Print correct ISBN mistakes. The systems then provide cross references "incorrect ISBN -> correct ISBN"). This should be taken into account in the URN resolution process. Further details on the process of assigning ISBNs can be found in section 5 (Namespace registration) below.

6.2.5 **URN Namespace Registration and Use**

The formal URN Namespace Identifier Registration for the pre-2005 version of the International Standard Book Number (ISBN) was done in RFC 3187 [RFC3187].

The new ISBN standard does not require a new namespace, but the registration is renewed here, as the registrant organization has moved from Staatsbibliothek zu Berlin - Preussischer Kulturbesitz to The International ISBN Agency, London, U.K, and the syntax and resolution details are amended.

**URN Namespace ID Registration for the International Standard Book Number (ISBN)**

This registration describes how International Standard Book Numbers (ISBN) can be supported within the URN framework.
Namespace ID: ISBN

This Namespace ID has already been assigned to the International Standard Book Number in January 2001 when the namespace was registered for the first time.

Registration Information:

Version: 2
Date: 2010-03-22

Declared registrant of the namespace:

Name: Mr. Brian Green
Affiliation: Director, The International ISBN Agency
Email: brian@isbn-international.org
Affiliation: EDItEUR, 39-41 North Road, London, N7 9DP, U.K.
Web URL: http://www.isbn-international.org/

Declaration of syntactic structure:

The namespace-specific string of ‘ISBN’ URNs is either an ISBN-13 (see Section 4.1.2 of RFC XXXX) or an ISBN-10 (see Section 4.1.1 of RFC XXXX); the former is preferred.


Relevant ancillary documentation:

The ISBN (International Standard Book Number) is a unique machine-readable identification number, which marks any edition of a book unambiguously. This number is defined in ISO Standard 2108. The number has been in use now for 30 years and has revolutionised the international book-trade. 170 countries and territories are officially ISBN members, and more of them are joining the system.

The administration of the ISBN system is carried out on three levels:

International agency,
Group agencies,
Publisher levels.

The International ISBN agency is located in London. The main functions of the Agency are:

- To promote, co-ordinate and supervise the world-wide use of the ISBN system.
- To approve the definition and structure of group agencies.
- To allocate group identifiers to group agencies.
- To advise on the establishment and functioning of group agencies.
- To advise group agencies on the allocation of international publisher identifiers.
- To publish the assigned group numbers and publisher prefixes in up-to-date form.

Identifier uniqueness an persistence considerations:

ISBN is a unique and persistent identifier. An ISBN, once it has been assigned, must never be re-used for another book. Moreover, a single manifestation of a book must never get a new ISBN.

There may be multiple manifestations of a single literary work such as a novel. In such case each manifestation shall receive a different ISBN. ISO has developed a new standard, ISTC (International Standard Text Code, ISO 21047-2009) which enables identification of textual works. See http://www.istc-international.org/ for more information. In the standard itself, annex E describes the relations between ISBN and other publication identifiers and ISTC.

Process of identifier assignment:

Assignment of ISBNs is controlled. There are three levels of control: the international agency, group agencies which typically operate in the national level and finally each publisher is responsible of using the ISBN system correctly. Small publishers may demand ISBN numbers one at a time by contacting the ISBN group agency. Large publishers receive ISBN blocks from which they allocate ISBNs to the books according to the ISBN assignment rules.

Process for identifier resolution:

See Section 4.3 of RFC XXXX.

Rules for lexical equivalence:

ISBN numbers are usually printed with the letters 'ISBN' and a single blank preceding them (for instance: ISBN 951-746-795-8). The data preceding the actual number must be removed before the ISBNs are analysed. The ISBN serves directly as the namespacespecific string (NSS) of 'ISBN' URNs.

Prior to comparing the NSS of two ISBN-based URNs for equivalence, all hyphens should be removed and letter 'X' capitalized. Prior to comparing a URN based on ISBN-10 with a URN based on ISBN-13, the ISBN-10 must be converted to the ISBN-13 form. This step is necessary since the ISBN-10s may or may not be already converted to the new form; as a rule, libraries shall keep the old ISBN since it is the one printed in books published prior to 2007, while publishers may convert the old identifiers originally assigned in ISBN-10 form and use the equivalent ISBN-13s in unchanged reprints of the books, which according to the ISBN assignment rules should not receive a new ISBN.

The URNs are equivalent if the normalized forms obtained this way compare equal.

Conformance with URN syntax:


Validation mechanism:

The check digit helps to assure the correctness of an ISBN number assigned for a book when it has been entered or processed by a human. Applications processing bibliographic data such as integrated library systems typically can check the correctness of both ISBN-10
and ISBN-13 (and make conversions between the two). If the number is wrong due to, e.g., a typing error made by a publisher, a correct ISBN is usually assigned afterwards. Although the book shall only contain the wrong number, national bibliography and system used by the book trade often to contain both the wrong and new, correct ISBN number.

Scope:

ISBN is a global identifier system used for identification of monographic publications. It is very widely used and supported by the publishing industry.

6.2.6 Security Considerations

This document proposes means of encoding ISBNS within the URN framework. An ISBN-based URN resolution service is depicted here both for ISBN-10 and ISBN-13, but only in a fairly generic level; thus questions of secure or authenticated resolution mechanisms are excluded. It does not deal with means of validating the integrity or authenticating the source or provenance of URNs that contain ISBNS. Issues regarding intellectual property rights associated with objects identified by the ISBNS are also beyond the scope of this document, as are questions about rights to the databases that might be used to construct resolvers.

6.2.7 IANA Considerations

IANA is asked to update the existing registration of the Formal URN Namespace 'ISBN' using the template given above in Section 5.1.

6.2.8 Acknowledgements

This draft version is the outcome of work started in 2008 and brought to the IETF in 2010 to launch a much larger effort to revise the basic URN RFCs as a part of project PersID (http://www.persid.org). PersID is developing tools for establishing an European network of URN resolvers concentrating on bibliographic identifiers. The aim in the IETF is to bring these RFCs in alignment with the current URI Standard (STD 63, RFC 3986), ABNF, and IANA guidelines. The discussion in PersID has contributed significantly to this work.

Your name could go here ...

6.2.9 References

Normative References


Informative References


Authors’ Addresses

Maarit Huttunen
The National Library of Finland
P.O. Box 26
Helsinki, Helsinki University FIN-00014
Finland
EMail: maarit.huttunen@helsinki.fi

Juha Hakala
The National Library of Finland
P.O. Box 15
Helsinki, Helsinki University FIN-00014
Finland
EMail: juha.hakala@helsinki.fi

Alfred Hoenes (editor)
TR-Sys
Gerlinger Str. 12
Ditzingen D-71254
Germany
EMail: ah@TR-Sys.de
6.3 Draft-hakala-rfc3188bis-nbn-urn-00

Abstract
National Bibliography Numbers, NBNs, are widely used by the national libraries and other organizations in order to identify various resources such as monographs pre-dating the emergence of the ISBN system or still images. As a rule, NBNs are applied to all kinds of resources that do not have an established identifier system of their own.

Since 2001, there has been a URN (Uniform Resource Names) namespace for NBNs, and during 2001-2009 millions of URN-based unique and persistent NBNs have been assigned. The namespace registration was performed in RFC 3188 and applied to the NBNs known at that point. No URN:NBN resolution services existed at the time when the RFC was written. Since then several countries have started using URN:NBNs to identify electronic resources and to provide persistent links to them. To this end, many countries have established URN:NBN resolution services that supply URN - URL linking.

This document replaces RFC 3188 and defines how NBNs can be supported within the URN framework. An updated namespace registration (version 4) is included.

Discussion
This document is an outcome of work performed in 2009-2010 as a part of the project PersID (http://www.persid.org) revising the basic URN RFCs, in order to bring them in alignment with the current URI Standard (STD 63, RFC 3986), ABNF, and IANA guidelines, and to establish a modern URN resolution system for bibliographic identifiers. This work is being brought to the IETF, and the intent is to establish a "URNbis" (or similar) working group in the Applications Area as soon as possible, which will then target this and related work.

Comments are welcome on the urn@ietf.org or the urn-nid@ietf.org mailing list (or sent to the authors). Interested parties are also invited to contribute to the WG formation process being discussed on the former list.

Status of This Memo
This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 20, 2010.

Copyright Notice
Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe the rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

6.3.1 Introduction

One of the basic permanent URI schemes (cf. RFC 3986 [RFC3986], [IANA-URI]) is ‘URN’ (Uniform Resource Name) as defined in RFC 2141 [RFC2141]. Currently, there is work in progress to revise that document [I-D.ah-rfc2141bis-urn]. Any identifier, when used within the URN system, needs its own namespace. As of this writing, there are 40 registered URN namespaces (see [IANA-URN]), one of which belongs to NBN, National Bibliography Number, as specified 2001 in RFC 3188 [RFC3188].

Currently URN:NBNs are in production use in several European countries including Finland, Germany, Italy, the Netherlands, Norway, and Sweden. Several other countries in Europe and elsewhere are considering usage of them. URN:NBNs have been applied in, e.g., Web archives, large collections of digitized materials, and collections of scientific articles.

As part of the validation process for the development of URNs back in late 90s, the IETF URN working group agreed that it was important to demonstrate that a URN syntax proposal can accommodate existing identifiers from well-established namespaces.

One such infrastructure for assigning and managing names comes from the bibliographic community. Bibliographic identifiers function as names for objects that exist both in print and, increasingly, in electronic formats. RFC 2288 [RFC2288] investigated the feasibility of using three identifiers (ISBN, ISSN and SICI) as URNs, with positive results, however it did not formally register corresponding URN namespaces. This was in part due to the still evolving process to formalize criteria for namespace definition documents and registration, consolidated later in the IETF into RFC 3406 [RFC3406].

URN Namespaces have subsequently been registered for NBN, ISBN and ISSN in RFCs 3188 [RFC3188], 3187 [RFC3187], and 3044 [RFC3044]. The ISBN namespace registration is being revised so that it covers both ISBN-10 and ISBN-13; [I-D.hakala-rfc3187bis-isbn-urn]. The current ISSN registration still does not cover ISSN-L, defined in the new version of ISSN. However, there is no registered namespace for SICI, and no plans to make such registration, due to the low popularity of the standard.

Please note that NBN differs from the other identifiers listed here because there is no standard describing NBNs. The term "National Bibliography Number" encompasses all identifier systems the national libraries use in addition to the more formally established identifiers. Historically, they were only applied in the national bibliography to identify the resources catalogued into it. During the last 10 years NBN scope has been extended to a wide variety of digital resources available via the Internet. Only a minority of these resources are catalogued in the national bibliography. Some national libraries have also allowed other organizations to apply NBNs to their publications and other materials.

Guidelines for using NBNs as URNs and the original namespace registration have been published in RFC 3188 [RFC3188]. The RFC at hand replaces RFC 3188 [RFC3188]; sections discussing the
methods in which URN:NBNs can be resolved have been updated and the text is also compliant with the stipulations of RFC 3406 [RFC3406], which has replaced RFC 2611 [RFC2611] that was applied in the initial registration.

6.3.2 Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

NBN refers to any National Bibliography Number identifier system used by the national libraries and other institutions using the system with the national library’s permission.

6.3.3 Identification and Resolution

As a rule, NBNs identify finite, manageably-sized objects, but these objects may still be large enough that resolution into a hierarchical system is appropriate. The materials identified by an NBN will often be digital, but they may also exist only in printed or other physical form. In such a case, the URN:NBN resolver should nevertheless be able to supply e.g. bibliographic data about the resource, possibly including the address where the resource is available.

National libraries are the key organizations providing persistent URN resolution services for resources identified with NBNs, independent of their form. National libraries may however allow other organizations such as university libraries or governmental organizations to assign NBNs to their resources. In such case the national library will co-ordinate the use of NBNs and support the NBN users in guaranteeing the persistence of these resources and resolution. These other organizations may establish their own resolution services or they may use the infrastructure provided by the national library.

6.3.4 Namespace considerations: National Bibliography Numbers (NBNs)

Overview

National Bibliography Number (NBN) is a generic name referring to a group of identifier systems utilised by the national libraries and institutions authorized by them, such as universities. The common denominator is that all these organizations are committed to preserve their collections for a long time; at least decades and possibly centuries.

Each national library uses its own NBNs independently of other national libraries; there is no global authority that controls NBN usage. For this reason, NBNs as such are unique only on a national level. When used as URNs, NBN strings must be augmented with a controlled prefix such as the ISO 3166 country code. These prefixes guarantee uniqueness of the NBN-based URNs on the global scale.

NBNs have traditionally been given to documents that do not have a formal (standard) identifier, but are catalogued to the national bibliography. Examples of this include books that predate the introduction of the ISBN in the 1970s, or modern books that for some reason have not received an ISBN. NBNs can be seen as a fall-back mechanism: if no other, standards-based identifier such as an ISBN can be given, an NBN is assigned.

URNs may also be used in universities’ open repositories when a resource already has another identifier -- often a DOI (Digital Object Identifier) -- that can only be resolved in a publisher’s (or third party’s) resolution service, which cannot be used to provide linking to the open repository. URN:NBN-based resolution can deliver that, and may also be extended to include access to other
repositories holding the same resource.

In principle, NBNs enable identification of any kind of resource, such as still images published in periodical articles, or short stories and poems published in book form or in the Web. Local policies may limit the NBN usage to for instance documents stored permanently in the national library’s collections. Following the initial registration of a URN namespace for NBN, several countries broadened the scope of NBN assignment significantly to cover broader scope of their digital materials.

Some national libraries (e.g., Finland, Norway, Sweden) have established Web-based URN generators, which enable authors and publishers to fetch NBN-based URNs for their network resources. There are also applications, used for instance in digitization processes, that assign NBNs automatically to resources or even their component parts such as still images published in monographs or serials.

Within the limitations set by RFC 2141, this document, and other relevant RFCs, both syntax and scope of local NBNs can be decided by each national library independently. Historically, NBNs have consisted of one or more letters and/or digits. For instance, (Finnish) NBN for the Romanian translation of the Finnish classic "Seven Brothers" published in 1957 is f568471. URN strings can contain encoded UNICODE characters, as specified in the declaration of syntactic structure, and there are no length limitations. Therefore, literally billions of NBNs can be allocated, which makes them suitable for, e.g., naming of Web documents.

In Italy a novel hierarchical distributed architecture for NBN assignment has been designed, in order to eliminate the single-point-of-error risks of a centralised system and to reduce the costs of managing a resolution service based on persistent identifiers.

The Central National Library in Florence manages the national domain NBN:IT and the national URN resolution service that contains every URN:NBN assigned in Italy. The library has supplied URN:NBN:IT subdomains to trusted institutions and bodies such as universities that are responsible for digital collections and routinely manage digital resources, for instance via creating and updating metadata on these resources, including location (URL) information. All these institutions have their own resolution services, but the URN – URL mappings in them are periodically harvested to the national resolution service using the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH; see (<http://www.openarchives.org/OAI/openarchivesprotocol.html>)).

Harvesting the data to the central node enables distributed service. The central node can serve the users when the local node is not functional. This architecture increases the robustness of the network via duplication of resolution services and enables peer-to-peer resolution between the second-level institutions. Moreover, it is possible to add yet another layer to the network by creating an international node, which shall contain all the data from the national nodes. Such international service may also be created by building a virtual union resolver that uses all the national nodes. The PersID project is investigating the feasibility of these technical solutions at the European level.

**Encoding Considerations and Lexical Equivalence**

Embedding NBNs within the URN framework initially did not present any particular encoding problems, since the ASCII characters utilized in traditional NBN systems belonged to the URN character set. Machine generated NBNs may be more demanding; if necessary, NBNs must be translated into canonical form as specified in [I-D.ah-rfc2141bis-urn].

When an NBN is used as a URN, the namespace specific string must consist of three parts:

- a prefix, consisting of either a two-letter ISO 3166 country code or other registered string,
- a delimiting character that is either hyphen (-) or colon (:), and
• the NBN string.
Delimiting characters are not lexically equivalent.

A Hyphen MUST be used for separating the prefix and the NBN string.

A Colon MAY be used as the delimiting character if and only if a country code-based NBN namespace is split further in smaller subnamespaces. If there are several national libraries in one country, these libraries can divide the national namespace between themselves using this method.

A national library may also assign to trusted organisation(s) such as a university or a government institution its own NBN sub-namespace. For instance, the national library of Finland has given Statistics Finland (<http://www.stat.fi/index_en.html>) a sub-namespace "st" (i.e., urn:nbn:fi:st). These trusted organisations must follow the general rules of the NBN usage provided by the national library, and take care of the long-term preservation of the identified resources in order to guarantee persistence.

Non-ISO 3166 prefixes, if used, MUST be registered on the global level. The U.S. Library of Congress SHALL maintain the central register of reserved codes.

Sub-namespace codes beneath a country-code-based namespace MUST be registered on the national level by the national library that assigned the code. The national register SHOULD be made available on the Web.

Two-letter codes MUST NOT be used as non-ISO prefixes, since all such codes are reserved for existing and possible future ISO country codes. If there are several national libraries in one country that use the same prefix - for instance, a country code -, they need to agree on how to divide the namespace. They may either share one namespace but agree on how to avoid assigning duplicate identifiers, or they may split the namespace into two sub-namespaces.

Models:

URN:NBN:<ISO 3166 country code>-<assigned NBN string>

URN:NBN:<ISO 3166 country code>:<sub-namespace code>-<assigned NBN string>

URN:NBN:<non-ISO 3166 prefix>-<assigned NBN string>

Examples:

URN:NBN:fi-fe201003181510
(NBN assigned to Tero Frestadius’s Master’s Thesis "In and Out": Segmentary Gang Politics in Los Angeles).

urn:nbn:ch:bel-9039

urn:nbn:se:uu:diva-3475

urn:nbn:hu-3006

From the libraries’ point of view, one of the key benefits of using URNs and other persistent identifiers is that there is only one location - the resolution service - where the linking information has to be maintained. If bibliographic records in library on-line public access catalogues (OPACs) contain URLs, then each record must be modified whenever the URL changes. With URNs, it is only necessary to maintain the URN - URL linking in the mapping table of the resolution service.
Resolution of NBN-based URNs

URNs can be used to provide various services. RFC 2483 [RFC2483] gives a few examples, such as retrieving a single URL or all URLs applying to the resource. Services available may vary, depending on the technical implementation of the URN resolver and the target system contacted. Please note that services must not be hard coded into the URN itself.

Eventually, URNs will be resolved with the help of a resolver discovery service (RDS). However, no such system is widely available yet. Therefore, URNs are usually embedded in HTTP URIs in order to make them actionable in the present Internet. In these HTTP URIs, the authority part must point to the appropriate URN resolution service. In Finland, the address of the national URN resolver is <http://urn.fi>. Thus the HTTP URI for the URN in the example is <http://urn.fi/URN:NBN:fi-fe201003181510>. This in turn resolves to the actual address of the thesis, which as of this writing (2010-03-27) was <https://oa.doria.fi/bitstream/handle/10024/59475/inandout.pdf?sequence=1>.

The country code-based prefix part of the URN namespace specific string will provide a guide to finding the correct national resolution service for URN:NBNs from the resolver discovery service when it is established. If there are multiple URN:NBN resolvers in the country, there are two possible approaches for making sure that RDS will work. All URN:NBN mappings can be harvested to the national node (the Italian approach). The other approach is to make the RDS aware of all the URN:NBN resolution services, and specify which parts of the national URN:NBN namespace they are capable of serving.

URN:NBN - URL mappings maintained nationally can be harvested using e.g. OAI-PMH from abroad to other national and international URN resolvers. This makes it possible to improve the reliability of the system; if the Finnish national resolver node does not respond, its URN - URL mappings may be available at other resolvers.

Persistence of any resolution service is not only a technical issue, but also an organisational and legal one. National libraries are in ideal position to provide persistent resolution services, since most of them maintain (legal) deposit collections, in which domestic publications shall be preserved for future generations. Increasingly these collection contain also digital resources.

Additional Considerations

Guidelines adopted and promoted by each national library define when different manifestations of a work should be assigned the same or differing NBNs. These rules apply only if identifier assignment is done manually. If identifiers are allocated programmatically, each manifestation of a resource will get a new NBN. For instance, over time the national library may digitize the same photograph many times from different sources, but each copy will get its own NBN. However, if a checksum such as MD5 (see RFC 1321 [RFC1321]) is used as NBN, the resources that are identical at bit level will receive the same NBN. Dissimilar resources may get the same checksum; with a reliable message digest algorithm such probability is however very small.

The rules governing the usage of NBNs are local and usually less strict than those specifying the usage of ISBNs and other standard identifiers. As long as the NBNs were assigned only in the national libraries, the identifier use was however well co-ordinated in practice. Now, following significant broadening in the scope of the NBN to cover Web resources, NBN assignment is less tightly controlled even within one national library. One resource -- for instance, a photo that has been published in many newspapers -- is likely to receive multiple NBNs. If two national libraries are digitizing the same resources and use NBNs for identification of these resources, the result will be duplicate NBN assignment since NBNs as a rule have national scope. If the metadata describing the resource is harvested into a common service, and the bibliographic records are similar enough to be merged, the user may see two NBNs and HTTP URIs pointing to two different resolution services and physical copies of the same resource. If the two copies had the same identifier such
as an ISBN, there would still be two HTTP URIs providing access to different physical copies of the resource.

If the same resource -- for instance, a master’s thesis published by two universities -- is held in two repositories located in different NBN-sub-namespaces within one national namespace, these copies may also receive different NBNs especially if there is no easy way of checking if the resource has already been identified somewhere. In this case, duplication may be revealed in the national level when the metadata records of the two copies are compared. In such case, like in the previous example, both URN:NBNs are equally valid.

Security Considerations
This document proposes means of encoding national bibliography numbers (NBNs) within the URN framework. It discusses resolution only at a very generic level; thus, questions of secure or authenticated resolution services and authentication of users of such services are out of scope. This document does not address means of validating the integrity or authenticating the source or provenance of URNs that contain NBNs. Issues regarding intellectual property rights associated with objects identified by national bibliography numbers are also beyond the scope of this document, as are questions about rights to the bibliographic databases that may be used to construct resolution services.

URN Namespace ID Registration for the National Bibliography Number (NBN)
This registration describes how National Bibliography Numbers (NBNs) can be supported within the URN framework.

[ RFC Editor: please replace "XXXX" in all instances of "RFC XXXX" below by the RFC number assigned to this document. ]

Namespace ID: NBN

This Namespace ID was formally assigned to the National Bibliography Number in October 2001 when the namespace was registered officially. Utilization of URN:NBNs started in demonstrator systems in 1998; production use has been going on for several years, and millions of URN:NBNs have been assigned.

Registration Information:

Version: 4
Date: 2010-05-18

Declared registrant of the namespace:

Name: Mr. Juha Hakala
Affiliation: Senior Adviser, The National Library of Finland
Email: juha.hakala@helsinki.fi
Postal: P.O.Box 15, 00014 Helsinki University, Finland
Web URL: http://www.nationallibrary.fi/

The National Library of Finland registered the namespace on behalf of the Conference of the European National Librarians (CENL) and Conference of Directors of National Libraries (CDNL), which have both made a commitment in 1998 to foster the use of URNs. The NBN namespace will be available for free for all national libraries in the world.

Declaration of syntactic structure:

The namespace specific string (NSS) will consist of three parts:
a prefix, consisting of either a two-letter ISO 3166 country code or other registered string and sub-namespace codes,
delimiting characters (colon (: ) and hyphen (-)), and
an NBN string assigned by the national library or sub-delegated authority.

Formal declaration of the NSS, using ABNF [RFC5234].

```
nbn_nss   = prefix "-" nbn_string
prefix    = cc_prefix / reg_prefix
cc_prefix = iso_cc *( ":" subspc )
iso_cc    = 2ALPHA
           ; country code as assigned by ISO 3166, part 1 --
           ; identifies the national library
           ; to which the branch is delegated
subspc    = 1*ALPHANUM
           ; as assigned by the respective national library
reg_prefix = 3*ALPHANUM
           ; as assigned by the Library of Congress --
           ; identifies a trusted third party
           ; to which the branch is delegated
nbn_string = <specific per prefix>
           ; must follow RFC 3986 <path-rootless> syntax
```

Colon may be used as a delimiting character only within the prefix, between ISO 3166 country code and sub-namespace code, which splits the national namespace into smaller parts.

Dividing non-ISO 3166-based namespaces further with sub-namespace codes MUST NOT be done.

Hyphen MUST be used as the delimiting character between the prefix and the NBN string. Within the NBN string, hyphen MAY be used for separating different sections of the code from one another.

Non-ISO prefixes used instead of the ISO country code must be registered. A global registry, maintained by the Library of Congress, shall be created and made available via the Web. Contact information: nbn.register@loc.gov.us.

All two-letter codes are reserved for existing and possible future ISO country codes and MUST NOT be used as non-ISO prefixes.

Sub-namespace codes MUST be registered on the national level by the national library which assigned the code. The list of such codes SHOULD be available via the Web.

Models:

```
URN:NBN:<ISO 3166 country code>-<assigned NBN string>
URN:NBN:<ISO 3166 country code:sub-namespace code>-<assigned
```
NBN string>
URN:NBN:<non-ISO 3166 prefix>-<assigned NBN string>

Example:

urn:nbn:de:gbv:089-3321752945

Relevant ancillary documentation:

National Bibliography Number (NBN) is a generic name referring to a group of identifier systems used by the national libraries and other organizations for identification of deposited publications and other resources that lack a ‘canonical’ identifier, or to descriptive metadata (cataloguing) that describes the resources. Each national library uses its own NBN system independently of other national libraries; there is neither a general standard defining NBN syntax nor a global authority to control the use of these identifier systems.

Each national library decides locally which resources shall receive NBNs. These identifiers have traditionally been assigned to documents that do not have a publisher-assigned identifier, but are nevertheless catalogued to the national bibliography. Typically identification of grey publications have largely been dependent on NBNs. With the introduction of the Internet and URN: NBN namespace in 1998, the scope of NBN assignment has been extended to a broad spectrum of Internet resources including, e.g., harvested Web pages.

Some national libraries (Finland, Norway, Sweden) have established Web-based URN generators that enable authors and publishers to fetch NBN-based URNs for the resources they publish in the Web. The most significant group of publications to which NBNs have been applied are doctoral theses.

The syntax of NBNs is decided by each national library independently. Historically, NBNs used in national bibliographies contained only characters that belong to the URN character set. Following the expansion of NBN scope and semi- and fully automated NBN assignment processes, some NBNs may contain characters that must be translated into canonical form according to the specifications in [I-D.ah-rfc2141bis-urn].

Identifier uniqueness and persistence considerations:

NBN strings assigned by two national libraries may be identical. In order to guarantee global uniqueness of NBN-based URNs, therefore a controlled prefix is present in the namespace specificstring. These NBNs, once given to the resource, will be persistent. Persistence of the resources themselves will be guaranteed by the national libraries as a part of their legal deposit activities. This applies to publications and Web resources only; long-term preservation of other resources such as governmental documents will be dependent on other actors like national archives.

An NBN, once it has been assigned, must never be re-used for another resource.

At the national level, libraries may utilise different policies for guaranteeing uniqueness of NBNs. They may be assigned sequentially by programs (URN generators) in order to avoid human mistakes. It is also possible to use checksums such as SHA-1 or MD5 as NBN.

Process of identifier assignment:

Assignment of NBN-based URNs MUST be controlled on national level by the national library / national libraries. Although the basic principles are the same, there are differences in
scope; for instance in the Netherlands URN:NBNs are used -- among other things -- to identify scientific articles stored in the national library’s long term preservation system, whereas most other URN users are not applying them on scientific articles. Finland, on the other hand, is using URNs extensively to identify and provide access to the digitized content.

National libraries may choose different strategies in assigning NBN-based URNs, and different approaches have varying levels of control. Manual URN assignment by the library personnel only provides the best possible control, especially if this is done traditionally, that is, only when the document is catalogued into the national bibliography. Usually the scope of URN:NBN is much broader than this; NBNs may for instance be automatically generated for each archived resource by a long term preservation system. From control point of view, the most liberal approach is a URN generator which builds URNs for everyone, with no guarantee that the resource identified will be preserved or accessible. Every national library must decide the degree of freedom it allows to the URN:NBN users. Usage rules may of course vary from one sub-namespace to the next. As of yet there are no international guidelines for NBN use, but they may be developed in the future.

Process for identifier resolution:

See Section 4.3 of RFC XXXX.

Rules for lexical equivalence:

None in the global level, beyond those expressed in [I-D.ah-rfc2141bis-urn].

National libraries may develop their own rules for the NBNs they use.

Conformance with URN syntax:

Traditional NBNs (those applied in the national bibliographies) consisted of ASCII 7-bit letters and digits (a-z and 0-9). For instance, the NBN of the first Hungarian translation of the Finnish national epos Kalevala is f20043425. The book was published in 1853 and therefore does not have an ISBN. Machine-generated NBNs must follow the stipulations of [I-D.ah-rfc2141bis-urn].

Validation mechanism:

None specified on the global level (beyond a routine check of those characters that require special encoding when expressed as URIs). A national library may use NBNs, which contain a checksum and can therefore be validated, but as of this writing there are no NBNs which incorporate a checksum.

Scope:

NBN is a global identifier system used for identification of diverse publications and other resources. It is widely used and supported by the national libraries.

6.3.5 Security Considerations

This document proposes means of encoding NBNs within the URN framework. An NBN-based URN resolution service is depicted, but only in a fairly generic level; thus questions of secure or authenticated resolution mechanisms are excluded. It does not deal with means of validating the integrity or authenticating the source or provenance of URNs that contain NBNs. Issues regarding intellectual property rights associated with objects identified by the NBNs are also beyond the scope
of this document, as are questions about rights to the databases that might be used to construct resolvers.

6.3.6 IANA Considerations

IANA is asked to update the existing registration of the Formal URN Namespace ‘NBN’ using the template given above in Section 4.6.

6.3.7 Community Considerations

National bibliography numbers enable the national libraries and organisations which liaise with them to uniquely identify resources and provide persistent links to the resource accessible in the Internet. Since most digital documents held in national libraries’ digital collections are not eligible for other, more formal identifiers such as ISBN, NBNs are a valuable asset for the community. A proof of this are the millions of URN:NBNs that have been allocated since the NBN namespace was reserved, and the operational services that have been built, using these identifiers and resolver applications.

For library users, URN-based identification and resolution services mean more efficient and reliable access to resources in general. No special tools are needed for this; Web browsers are sufficient. The users may also be able to acquire URN:NBNs to their own key resources such as university thesis.

6.3.8 Acknowledgements

The author wishes to thank his colleagues in the PersID project for their support.

6.3.9 References

Normative References


Informative References


Authors' Addresses

Juha Hakala
The National Library of Finland
P.O. Box 15
Helsinki, Helsinki University FIN-00014
Finland
Email: juha.hakala@helsinki.fi

Alfred Hoenes (editor)
TR-Sys
Gerlinger Str. 12
Ditzingen D-71254
Germany
Email: ah@TR-Sys.de