HANDLE.NET (version 8)

Technical Manual
Version 8

Corporation for National Research Initiatives
June 2015
hdl:4263537/5043

Credits

CNRI wishes to thank the prototyping team of the Los Alamos National Laboratory Research Library for their collaboration in the deployment and testing of a very large Handle System implementation, leading to new designs for high performance handle administration, and handle users at the Max Planck Institute for Psycholinguistics and Lund University Libraries NetLab for their instructions for using PostgreSQL as custom handle storage.

Version Note

Handle System Version 8, released in June 2015, constituted a major upgrade to the Handle System. Major improvements include a RESTful JSON-based HTTP API, a browser-based admin client, an extension framework allowing Java Servlet apps, authentication using handle identities without specific indexes, multi-primary replication, security improvements, and a variety of tool and configuration improvements. See the Version 8 Release Notes for details.

Please send questions or comments to the Handle System Administrator at hdladmin@cnri.reston.va.us.

Table of Contents

1 Introduction

The Handle System is a comprehensive system for assigning, managing, and resolving persistent identifiers for digital objects and other resources on the Internet. The Handle System includes an open set of protocols, a namespace, and an implementation of the protocols. The protocols enable a distributed computer system to store identifiers of digital resources and resolve those identifiers into the information necessary to locate and access the resources. This associated information can be changed as needed to reflect the current state of the identified resource without changing the identifier, thus allowing the name of the item to persist over changes of location and other state information.

1.1 Handle Syntax

Within the handle namespace, every identifier consists of two parts: its prefix, and a unique local name under the prefix known as its suffix. The prefix and suffix are separated by the ASCII character "/". A handle may thus be defined as

<Handle> ::= <Prefix> "/" <Handle Local Name>

For example, handle "12345/hdl1" is defined under the Handle Prefix "12345", and its unique local name is "hdl1".

Handles may consist of any printable characters from the Universal Character Set of ISO/IEC 10646, which is the exact character set defined by Unicode. The UCS character set encompasses most characters used in every major language written today. To allow compatibility with most of the existing systems and prevent ambiguity among different encoding, handle protocol mandates UTF-8 to be the only encoding used for handles. The UTF-8 encoding preserves any ASCII encoded names, which allows maximum compatibility to existing systems without causing naming conflict.

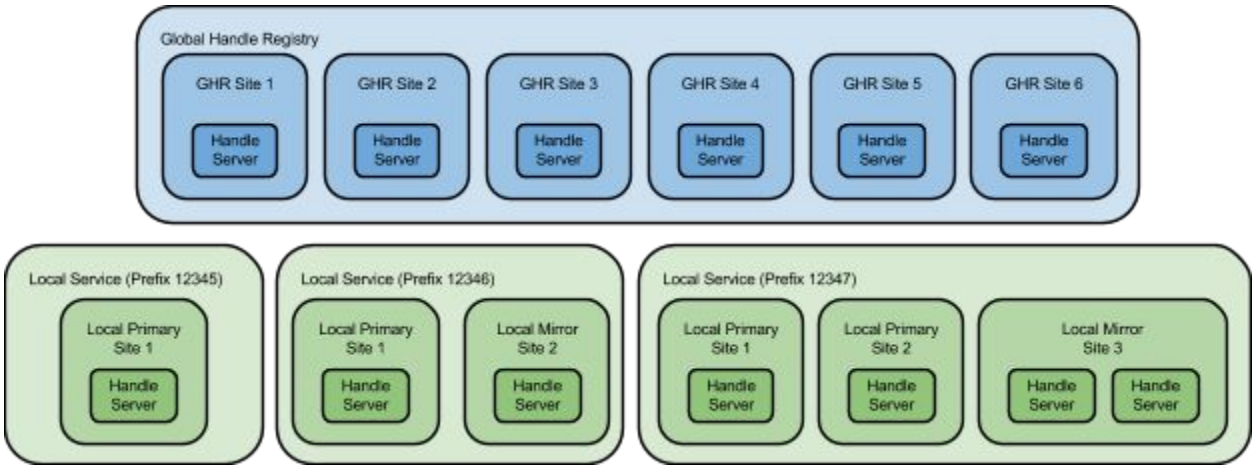
In general, handles are case sensitive. However, any handle service may define its namespace such that all ASCII characters within any identifier are case insensitive. This is recommended and the default for Handle.net server software. The Global Handle Registry® (GHR) guarantees that handles resolved from the GHR are case-insensitive. Note that case-insensitive handle services generally use ASCII case folding only; more general Unicode case folding and Unicode normalization should not generally be expected.

The handle namespace can be considered as a superset of many local namespaces, with each local namespace having its own unique handle prefix. The prefix identifies the administrative unit of creation, although not necessarily continuing administration, of the associated handles. Each prefix is guaranteed to be globally unique within the Handle System. Any existing local namespace can join the global handle namespace by obtaining a unique prefix, with the resulting identifiers being a combination of prefix and local name as shown above. Every handle is then defined under a prefix. The collection of local names under a prefix is the local namespace for that prefix. Any local name must be unique under its local namespace. The uniqueness of a prefix and a local name under that

prefix ensures that any identifier is globally unique within the context of the Handle System.

Each prefix may have many derived prefixes registered underneath. A derived prefix is formed syntactically by appending "." followed by other characters (exception "/" and ".") to an existing prefix. For instance prefix "10.1045" is derived from "10" and prefix "10.978.8896471" is derived from "10.978" which is derived from "10". In general derived prefixes need not be administratively related to the prefixes from which they are derived.

1.2 Architecture



The Handle System has two physical levels of hierarchy, the root service (also known as the Global Handle Registry®) and local services. Local handle services contain the handle records under a specific prefix. Whereas the root service contains handle records that describe who controls which prefixes and how to reach the local handle services for specific prefixes.

A handle service can be composed of one or more sites. Sites can be primary or mirror. Mirror sites replicate the handle records stored on the primary sites. Typically a service has a single primary, but it is possible to have a service with multiple primaries which then replicate from each other. Replication as implemented in the Handle.net software offers eventual consistency.

Typically there is a one-to-one relationship between a site and a handle server. It is however possible to have multiple handles servers in a site. In this case the handle data for the site is partitioned across the handle servers in the site. Having multiple handle servers in a single site is unusual though it can be desirable if you have more handle records than can be managed by a single handle server.

Resolution is performed by first querying the root handle service for the service handle record pertaining to the prefix. This prefix handle is constructed using the syntax 0.NA/<prefix>.[1] The returned handle record will contain one or more HS_SITE handle value. These handle values describe

[1] 0.NA derives from "naming authority", an obsolete term for "prefix".

how to reach the sites that make up the local handle service for that prefix. The resolver selects a site and then sends it a resolution request for the desired handle record.

It is also possible to have prefixes delegated from the GHR to local handle services. In this scenario, the prefix handle records for all prefixes derived from a given prefix P are resolved and administered at a local handle service defined by HS_SITE.PREFIX handle values in the prefix handle record for P. This allows the possibility of additional levels of hierarchy in the prefix resolution process.

1.2.2 Storage

The Handle System has been designed at a very basic level as a distributed system; that is, it will run across as many computers as are required to provide the desired functionality.

Handles are held in and resolved by handle servers and the handle servers are grouped into one or more handle sites within each handle service. There are no design limits on the total number of handle services which constitute the Handle System, there are no design limits on the number of sites which make up each service, and there are no limits on the number of servers which make up each site. Replication by site, within a handle service, does not require that each site contain the same number of servers; that is, while each site will have the same replicated set of handles, each site may allocate that set of handles across a different number of servers. Thus, increased numbers of handles within a site can be accommodated by adding additional servers, either on the same or additional computers, additional sites can be added to a handle service at any time, and additional handle services can be created. Every service must be registered with the Global Handle Registry, but that handle service can also have as many sites with as many servers as needed. The result is that the number of identifiers that can be accommodated in the current Handle System is limited only by the number of computers available.

1.2.3 Performance

Constant performance across increasing numbers of identifiers is addressed by replication, caching and hashing.

The individual handle services within the Handle System each consist of one or more handle service sites, where each site replicates the complete individual handle service, at least for the purposes of handle resolution. Thus, increased demand on a given handle service can be met with additional sites, and increased demand on a given site can be met with additional servers. This allows for clients to optimize resolution performance by selecting the "best" site from a group of replicated sites.

Caching may also be used to improve performance and reduce the possibility of bottleneck situations in the Handle System, as is the case in many distributed systems. The Handle System data model and protocol design includes a space for cache time-to-live values.

Hashing is used in the Handle System to evenly allocate any number of identifiers across any number of servers within a site, and allows a single computation to determine on which server within a set of servers a given handle is located, regardless of the number of handles or the number of servers. Each

server within a site is responsible for a subset of handles managed by that site. Given a specific identifier and knowledge of the handle service responsible for that identifier, a handle client selects a site within that handle service and performs a single computation on the handle to determine which server within the site contains the handle. The result of the computation becomes a pointer into a hash table, which is unique to each handle site and can be thought of as a map of the given site, mapping which handles belong to which servers. The computation is independent of the number of servers and handles, and it will not take a client any longer to locate and query the correct server for a handle within a handle service that contains billions of handles and hundreds of servers, than for a handle service that contains only millions of handles and only a few servers.

1.1 Protocols and APIs

A single handle server typically opens three network listeners, on port 2641 UDP, port 2641 TCP, and port 8000 TCP. This can be changed in configuration (the config.dct and siteinfo.json files, see) as well as in the HS_SITE values of prefix handles that refer to the handle server.

Port 2641 (UDP and TCP) is the IANA-assigned port number for the Handle wire protocol. The Handle service model and wire protocol are described in RFC 3650, RFC 3651, and RFC 3652. Handle clients generally will try to use 2641 UDP for resolution requests, which provides optimum performance in typical scenarios. TCP is generally required for administrative requests, and is used as a fallback for resolution when UDP is slow or unavailable.

Port 8000 offers an HTTP and HTTPS interface. Handle servers (from Handle.net software version 8) use "port unification" so that HTTP and HTTPS are available over the same port. If the standard Handle protocol ports are not available, Handle clients may fall back to tunneling the Handle wire protocol over HTTP. Additionally (from Handle.net software version 8) Handle servers offer a JSON-based HTTP API using many RESTful principles over this interface; see... Finally, Handle servers incorporate a modular extension framework in the form of a Java servlet container, see... Java servlet web-apps can be added to the Handle server and can optionally provide a browser-based UI accessible over the HTTP interface.

The Handle.net software distribution comes with a browser-based administrative client, "admin.war", which could be accessed at https://IP-ADDRESS:8000/admin/, where the IP address should be the public address from siteinfo.json. Note that even though it is accessible through the Handle server as a convenience, this is a pure client, which talks to the handle server over its public HTTP API, and could in principle run anywhere.

The Handle.net distribution comes with GUI client applications and command-line clients to perform Handle operations; see the Handle Tool User Manual and Chapter 3, *Batch Operation*. These tools use the Java Handle client library which comes with the Handle.net software; the Java Handle client library accesses Handle servers using the Handle wire protocol. Users are encouraged to develop their own client tools using the Java Handle client library. There are not maintained client libraries for other development environments than Java; developers using other languages may prefer to use the

Handle HTTP API to communicate with Handle servers.

1.3 Authentication

The current distribution of the Handle.net software uses the Java standard cryptography libraries for low level cryptography routines.

In order to authenticate a user needs to have a handle system identity. A handle system identity is either a public key or secret key stored on a handle record. This identity is expressed as a string defining the index of the value that contains the key separated from the handle by a colon. For example if you had a public key stored at index 300 on handle 12345/abc the authentication identity would be "300:12345/abc". When you register a prefix a prefix handle is created on the root service and your public key is stored on that handle at index 300. As such it is typical for handle users to have a handle system identity of the form:

300:0.NA/<prefix>

Since the prefix handle 0.NA/<prefix> is under the control of GHR administrators rather than the LHS administrators, many users will find it more useful to have administrators on handles under their prefix. There could either be multiple handles, or multiple indexes in one handle, for instance

300:12345/ADMIN1
300:12345/ADMIN2

or

300:12345/ADMIN
301:12345/ADMIN

The index in a handle identity is generally a positive integer, but the special value 0 indicates that the user's public key or secret key is stored at some unspecified index. Tools can use this to allow handle identities which are effectively just handles rather than index:handle pairs. If such an unindexed handle identity is referenced for authorization, any index will be considered authorized; if a client authenticates using an unindexed handle identity, then the client will be authorized to perform operations allowed to the unindexed identity but also, in the case of public key authentication, operations allowed to the identity using an actual index with the correct public key. (For secret key authentication the server may not be able to determine the "actual" index so only the unindexed authorization holds.) This feature is supported starting in Handle.net software version 8.

1.3.1 Types of Authentication

As further described in *System Fundamentals:* Authentication on the Handle System web site, the Handle System provides two forms of authentication: public key and secret key.

In the current implementation, public key authentication is performed using the DSA or RSA

algorithm. DSA generally uses a key length of 1024 bits; RSA allows a key length variable from 1024 to 4096 bits or higher, and can be chosen by the user when generating keys. The Handle.net software distribution defaults to a 2048 bit RSA key.

Public key authentication requires two keys: a public key and a private key. The public key is stored in a handle to make it available to the public. The private key should be securely stored on the computer with the handle client that will be authenticated. To prevent unauthorized use of a private key, it can be encrypted using a symmetric algorithm. The current Handle.net implementation of the Handle System uses AES-128 for this purpose.

Secret key authentication relies on a secure MAC algorithm. In general, secret key authentication uses three parties: (1) the authenticating client; (2) the server where the client is performing an operation; and (3) another server which is able to check the client's authentication. Handle.net software version 8 will by default use PBKDF2-HMAC-SHA1 to generate a derived key from the secret key, and then use HMAC-SHA1 to generate the MAC. Older software will use the SHA1 hash of the secret key concatenated with a challenge concatenated again the with secret key. A secret key consists of a single byte string. This byte string is stored as plain text in a handle. Read permissions on the handle need to be restricted to ensure the secrecy of the secret key.

1.3.2 Certification

Clients can request that a handle server cryptographically certify its messages with its public key. This certification can be used to verify the authenticity of handle server transmissions. The public key for a handle server is stored in its site information record.

1.3.3 Sessions

Establishing sessions with a handle server offers a performance benefit by allowing the client to perform an expensive authentication only once for many individual requests. Sessions also offer additional security functionality. For background on handle server sessions, see *System Fundamentals:* Sessions on the Handle System web site.

The Handle System allows for encryption of communication after establishing a session with a handle server. This is equivalent to SSL or TLS as used in protocols such as HTTPS, as it affords protection from eavesdropping and man-in-the-middle attacks. When encryption is requested, Handle.net version 8 software by default encrypts session communications using AES-128.

For instructions on enabling session encryption see the Handle Tool User Manual and Chapter 3, *Batch Operation*.

2 Installing and Running a Handle Server

This section outlines the steps required to get the Handle.net software, install it, set it up, and acquire a prefix.

The distribution includes a GUI tool developed by CNRI for performing handle operations such as creating and deleting handles, authenticating, and setting up a handle server. The Handle Tool has been found to be useful for quickly creating and updating handle URL values. See also the Handle Tool User Manual.

2.1 Installing Java™

The Handle.net software requires Java™ to run. If you already have Java™ installed, confirm that you have version 6 or later. You can check this by running the command 'java –version'.

2.2 Unpacking the Distribution

Download the Handle.net software distribution from http://www.handle.net. To uncompress the distribution package on Unix-like platforms use the gunzip and tar programs. The distribution package can be uncompressed on Windows-based systems using the program WinZip©.

The distribution will uncompress to have subdirectories bin, doc, lib, as well as the zipped source in src.zip. The examples in this section will assume that the distribution was unpacked into /hs/hsj-8.0.0. All of the executable/batch files necessary to setup the server are in the directory /hs/hsj-8.0.0/bin.

2.3 Choosing an Installation Directory

Create a folder for the server configuration. We will assume this folder is called "/hs/srv_1". Be sure to create a new directory for each server on the same machine. To create this directory on Unix, run this command:

```
mkdir -p /hs/svr_1
```

For more information on the files contained in the installation directory, see Section 2.7, *Installation Directory*.

2.4 Running the Setup Program

The HANDLE.NET software includes an installation program. The program requires Java™, so make sure you have the java binary directory in your system path. Navigate to the "hsj-8.0.0" directory and execute the following commands:

On Unix-like systems: bin/hdl-setup-server /hs/srv_1
On Windows systems: bin\hdl-setup-server.bat \hs\svr_1

The installation program guides you through a series of configuration options. Once complete, there will be a file called 'sitebndl.zip' in your handle server directory which you will email to the Handle System Administrator at hdladmin@cnri.reston.va.us. The Administrator will use the 'sitebndl.zip' file to create a prefix in the root service (GHR), and will notify you when this has been completed. You will not be able to continue the install until you receive information from the Handle System Administrator concerning your prefix.

Please note: A Handle System Service Agreement is required in order to provide identifier/ resolution services. (Please see http://hdl.handle.net/4263537/5029 ***for more information.) You will not receive a prefix from the Handle System Administrator until you have agreed to the terms of the Service Agreement. Once you have agreed to the Service Agreement, paid the appropriate fees, and received prefix information from the Handle System Administrator, you may proceed with the following steps to 'Home' your prefix to your new handle service.***

2.5 Running the Handle Server for the First Time

Go to your 'svr_1' directory (where you installed your Handle.net software) and edit the 'config.dct' file. Replace the YOUR_PREFIX with your prefix (as indicated in your email message from hdladmin). This will need to be done three times, under server_admins, replication_admins, and auto_homed_prefixes. The server_admins and replication_admins entries allow anyone with the private key that matches your public key to be an administrator for your server.

It is necessary to tell the handle server which prefix(es) it is responsible for. This is called homing a prefix. There are a couple of ways you can home a prefix on a handle server. The simplest way to home your prefix is to add it to the "auto_homed_prefixes" list in the config.dct file.

e.g. If your prefix is 12345 you should set the "auto_homed_prefixes" attribute in the config.dct to:

```
"auto_homed_prefixes" = (
      "0.NA/12345"
)
```

When the handle server starts up it checks the attribute and ensures that any prefixes listed there are homed.

Start the server using the configuration just generated.

On Unix-like systems: bin/hdl-server /hs/srv_1
On Windows systems: bin\hdl-server.bat /hs/srv_1

Note: If you chose to encrypt your private key(s), you will be prompted for your passphrase here. Also

note that Java™ does not have the ability to disconnect from a terminal so you will have to put the process in the background. On Unix systems type Ctrl Z, then bg, then press ENTER. Once the process is running in the background, you can use the "disown" command to ensure that the handle server process survives the end of the terminal process.

2.5.1 Homing your prefix with the admin tool

If you did not home your prefix using the auto_homed_prefixes configuration option, you will need to send an administrative request to the server, while the server is running, in order to home the prefix.

Start the Handle Tool using the following command:

On Unix-like systems: bin/hdl-admintool
On Windows: bin\hdl-admintool.bat

Click on the 'Authenticate' button. You will be prompted for your authentication information.

The 'Your ID' will be 0.NA/YourPrefix.
The blank field to the right is the Index and should be 300.
The 'Key Type' should be Public/Private Key.

Browse to find your private key file. It will be in the "svr_1" directory (where you installed your server) and is named "admpriv.bin". Click "OK". You may be prompted for your secret passphrase. This is the password you entered for Administration when you ran the setup program.

Next, "Home" your prefix. (See the Handle Tool User Manual, Chapter 11, "Homing/Unhoming a Prefix".) Select "Tools->Home/Unhome Prefix". This example assumes you were allotted handle "0.NA/YourPrefix".

"Prefix" field should contain: YourPrefix
Select the "Home Prefix" radio button.
Select "By Site Info File" and locate your "siteinfo.bin" file in "svr_1"
OR
Select "By Address" and enter the "Server Address" and "Server Port"
Click "Do It"

Please note that the Handle System does not use DNS.

2.6 How Your Prefix Was Set Up

A prefix handle record will have been created for you on the root handle service. The handle record for prefix 12345 would have the following handles values:

Prefix Handle: 0.NA/12345

Prefix Admin Group: 0.NA/12345 index 200 type HS_VLIST
Prefix Public Key: 0.NA/12345 index 300 type HS_PUBKEY

When authenticating, identify yourself using the Prefix Admin Public Key and the associated private key which is in your 'admpriv.bin' file on your computer. The instructions are in the 'INSTALL.txt' file (see Section 2.5, *Running the HANDLE.NET Server For the First Time*). This is what you used to authenticate yourself to "home" your prefix or perform other administrative operations.

When creating new identifiers, specify an administrator who will have permission to modify or delete each new identifier (handle) by adding an HS_ADMIN value that references a public key, secret key or admin group to each new handle. We recommend that you specify your Prefix Admin Group (see above) value as the administrator for each new handle.

Every value in a handle has a different index. The following pattern works well. Start with 100 for all admin values, start admin group values at 200 and make the public key index 300. So the values of a handle (12345/hdl1), might look like this:

100 HS_ADMIN 0.NA/12345 index 200
3 URL http://www.someorg.com/info
4 email someone@someorg.com

2.7 Installation Directory

The installation directory contains a number of files. This section explains the function of each.

2.7.1 logs/access.log

If you enabled 'log accesses' on any of your handle server interfaces, all requests sent to those interfaces are logged here. Below is a sample line from this file.

10.0.1.105 TCP:HDL(2.10) "2015-05-27 13:23:54.019-0400" 1 100 57ms 12345/1

The first column is the IP address of the host that made the request. The second column shows the interface the request was made on, with the Handle protocol version used in parentheses. Next is the date and time the request was made. The time is followed by the Handle Operation Requested Code (OP) of the request. In this case the OP is 1, for resolution. The OP is followed by the Handle Server Response Code (RC). In this case, the RC is 100, for handle not found. Here is a list of possible OPs and RCs:

	Handle Operation Requested Code (OP)		**Handle Server Response Code (RC)**
1	Resolve Handle	1	Success
2	Get Site Information	2	Error
100	Create Handle	3	Server Too Busy

101	Delete Handle	4	Protocol Error
102	Add Value	5	Operation Not Supported
103	Remove Value	6	Recursion Count Too High
104	Modify Value	7	Server Read-only
105	List Handles	100	Handle Not Found
200	Challenge Response	101	Handle Already Exists
201	Verify Challenge	102	Invalid Handle
300	Home Prefix	200	Values Not Found
301	Unhome Prefix	201	Value Already Exists
302	List Homed Prefixes	202	Invalid Value
400	Session Setup	300	Out of Date Site Info
		301	Server Not Responsible
		302	Service Referral
		303	Prefix Referral
		400	Invalid Admin
		401	Insufficient Permissions
		402	Authentication Needed
		403	Authentication Failed
		404	Invalid Credential
		405	Authentication Timed Out
		406	Authentication Error
		500	Session Timeout
		501	Session Failed
		502	Invalid Session Key
		504	Invalid Session Setup Request
		505	Session Duplicate Msg Rejected

After the response, the log entry shows the number of milliseconds the server took to respond to the request. This is useful for gauging the performance of a handle server. The final column of the log entry indicates the handle that was requested (if applicable).

Since Handle.net version 8 software, the log files will note the authenticated administrator for admin operations. This is in a column between the response time and requested handle, as follows.

10.0.1.105 TCP:HDL(2.10) "2015-05-27 13:31:42.011-0400" 100 1 78ms adm=300:200/23 12345/1

2.7.2 logs/error.log

As suggested by its name, this file contains a log of server errors.

2.7.3 config.dct

This is the server configuration file. See Chapter 4, *Advanced Server Configuration*, for more information.

2.7.6 siteinfo.json

This contains the 'HS_SITE' record for this server. It is stored in the handle prefixes that this server is responsible for. It is also returned directly by the server to a get-site-information request. Along with config.dct it is partially responsible for server configuration. See Chapter 4, *Advanced Server Configuration*, for more information.

2.7.5 bdbje/

By default the handle server uses a Berkeley DB Java Edition database to store handles and handle values as well as prefixes that are homed to the server. This database is located in the bdbje subdirectory.

2.7.8 pubkey.bin, privkey.bin

These are the public and private keys for the server. The public key is stored in the server's 'HS_SITE' entry and so is duplicated in siteinfo.json. The private key is used to sign responses to requests. In a way similar to HTTPS, this allows handle clients to confirm that they are talking to the expected server.

2.7.9 delete_this_to_stop_server

A convenient way to stop a running server is to delete this file.

2.7.9 txns/

This directory stores the server transaction queue. This keeps track of handle administration operations in order to replicate to secondary servers.

2.7.9 txn_id

This file stores the most recent transaction number.

2.7.8 replpub.bin, replpriv.bin

In a mirroring server, these are the public and private keys the server uses to authenticate itself, as a replicating client, to pull transactions from other servers.

2.7.9 txnstat.dct

In a mirroring server, this file stores the replication status of how up-to-date the mirror is with the

transactions from other servers.

2.7.7 admpub.bin, admpriv.bin

These are the public and private keys that were created for the administrator during the installation process. These files are not actually used by the server itself. Typically the admpub.bin public key is stored in the prefix handle. This key pair is used by the server administrator to authenticate to the server.

2.7.9 serverCertificate.pem

This is an X.509 certificate used by the server when responding to HTTPS requests. By default the server will automatically generate a self-signed certificate.

2.7.9 webapps, webapps-temp, webapps-storage

The version 8 handle server allows for modular extension using Java servlet technology. Servlet applications ("web apps") may be dropped into the webapps folder. These applications may expose a client interface accessible over the handle server's HTTP/HTTPS interface, or they may simply act within the handle server. The webapps-temp folder is used for temporary files (notably exploded war files), and the webapps-storage folder is used to supply a storage directory to each extension in a generic way. For more details see

2.9 Client configuration

Each handle server incorporates a handle client, used by the server to resolve prefix handles and check client authentication. Every handle client using the Handle.net software refers to certain files stored in the subdirectory '.handle' under the home directory of whatever user runs the handle server. Certain of these files are created by running the handle server so are included here. For more information see

2.7.4 $HOME/.handle/bootstrap_handles

This JSON-formatted file contains information about handle records necessary for bootstrapping handle resolution, notably the HS_SITE records of the global handle servers. This file should not generally need to be edited.

2.7.4 $HOME/.handle/root_info

The bootstrapping information in a format used by Handle.net software prior to version 8.

2.7.4 $HOME/.handle/config.dct

Client-level configuration. Some configuration may be useful for running a server. For more

information see

2.9 Restarting a Handle Server

To stop the handle server delete the file named "delete_this_to_stop_server" that is located in your handle server directory.

Then restart the server using the command:

```
hdl-server your_svr_dir
```

The "delete_this_to_stop_server" file is recreated each time the handle server is started.

Please notify CNRI via email to hdladmin@cnri.reston.va.us if you plan to shut down your server permanently.

2.10 Inactive Prefixes

The Handle System Service Agreement requires Resolution Service Providers to ensure that the identifiers they create will resolve. If a handle service is shut down, the Handle System Administrator must be notified in advance, and arrangements must be made to enable clients to correctly inform users of the status of those handles.

To help ensure proper resolution of such identifiers, the handle type HS_NAMESPACE can be used to alert handle clients that a prefix is no longer supported or being used and handles created under that prefix have been removed. The Handle System Administrator would add an HS_NAMESPACE value type to an inactive prefix that includes status, contact information, and a customized message :

```
<namespace>
<contact>yourName@yourOrg.org</contact>
<status>inactive</status>
<statusmsg>This prefix has been deactivated by the administrator as of September
2006.</statusmsg>
</namespace>
```

Namespace-aware native handle clients, including the proxy server, that attempt to resolve handles under a prefix with an HS_NAMESPACE value indicating the prefix is inactive, will return a "Handle Not Found" error message including the contact and statusmsg data (if provided) stating that the prefix is no longer being maintained and all handles beginning with the prefix have been removed.

3 Batch Operation – Command Line

It may be desirable to perform more handle operations than it is possible to perform using either of the administration tools. In those cases it is best to use the batch facilities included with the HANDLE.NET software distribution.

Submit batches using the 'GenericBatch' command line utility, which can be invoked using the following command:

bin/hdl-genericbatch <batchfile> [<LogFile>]
On Windows: bin\hdl-genericbatch.bat <batchfile> [<LogFile>]

All batch files are plain text format. One batch file can have more than one handle operation. The handle operations are: Create Handle, Delete Handle, Home/Unhome Prefix, Add Handle Value, Remove Handle Value, Modify Handle Value, Authenticate User, Setup Session.

If you need to change authentication for subsequent batch operations, the new authentication information should be put before the batch block. If you authenticate during the batch submission, then you need not include the authentication information in the batch file.

3.1 Create Handle Batch Format

Operation name is 'CREATE'. The first line is composed of the following:

CREATE + space + handle_name

The next lines are handle value lines. (See Section 3.9, *Handle Value Line Format*.) There must be a handle value line to define the administrator of the handle. End the 'CREATE' handle operation with a blank line.

The list of predefined handle value types is as follows: HS_ADMIN, HS_VLIST, HS_SECKEY, HS_PUBKEY, HS_SITE, HS_SERV, HS_ALIAS, EMAIL, URL, URN, INET_HOST, 10320/LOC. Each handle value line must start with a unique index number, followed by the handle value type from the list above, ttl (the time to live in seconds), the permission set (admin read, admin write, public read, public write), and the value data. See Section 4.9, *Handle Value Line Format* for more detail.

Example:

```
CREATE 12345/hdl1
100 HS_ADMIN 86400 1110 ADMIN
300:111111111111:12345/hdl1
300 HS_SECKEY 86400 1100 UTF8 my_password
3 URL 86400 1110 UTF8 http://www.handle.net
```

```
CREATE 12345/hdl2
100 HS_ADMIN 86400 1110 ADMIN 200:111111111111:0.NA/12345
3 URL 86400 1110 UTF8 http://www.yourorg.org
```

3.2 Delete Handle Batch Format

Operation name is 'DELETE'. This operation deletes an existing handle completely. Every record is a line with:

DELETE + space + handle_name

Example:

```
DELETE 12345/hdl1
DELETE 12345/hdl2
```

3.3 (Un)Home Prefix Batch Format

Operation name is 'HOME' or 'UNHOME'. This operation associates a prefix with a handle server. It only works on existing prefixes and active handle servers. It tells the server which prefixes will be homed or unhomed to it. The first line provides the service information:

HOME/UNHOME + space + server_ip:server_port:protocol(tcp,udp,http)

The next lines give the prefix names which will be homed/unhomed at this server.

Examples:

```
HOME 10.27.10.28:2641:TCP
0.NA/12345

UNHOME 10.27.10.28:2641:TCP
0.NA/12345
0.NA/TEST1.t1
```

3.4 Add Handle Value Batch Format

Operation name is 'ADD'. This operation adds new handle values to an existing handle. The first line is composed of the following:

ADD + space + handle_name

The next lines are handle value lines. (See Section 3.9, *Handle Value Line Format*.) There must be a

handle value line to define the administrator of the handle. End the CREATE handle operation with a blank line. The list of predefined handle value , is as follows: HS_ADMIN, HS_VLIST, HS SECKEY, HS PUBKEY, HS SITE, HS SERV, HS ALIAS, EMAIL, URL, URN, INET HOST, 10320/LOC. Each handle value line must start with a unique index number, followed by the handle value type from the list above, ttl (the time to live in seconds), the permission set (admin read, admin write, public read, public write), and the value data.

```
ADD 12345/hdl1
5 URL 86400 1110 UTF8 http://www.handle.net/admin.html
6 EMAIL 86400 1110 UTF8 hdladmin@cnri.reston.va.us

ADD 12345/hdl2
5 URL 86400 1110 UTF8 http://www.cnri.reston.va.us
6 EMAIL 8600 1110 UTF8 hdladmin@cnri.reston.va.us
```

3.5 Remove Handle Value Batch Format

Operation name is 'REMOVE'. This operation removes one or more handle values from an existing handle. Every record is a line with:

```
REMOVE + space + indexes:handle_name
```

Each index is separated by ','

Example:

```
REMOVE 5:12345/hdl1
REMOVE 5,6:12345/hdl2
```

3.6 Modify Handle Batch Format

Operation name is 'MODIFY'. This operation changes one or more handle values for an existing handle. The first line is composed of the following:

```
MODIFY + space + handle_name
```

The next lines are handle value lines. (See Section 3.9, *Handle Value Line Format*.) There must be a handle value line to define the administrator of the handle. End the CREATE handle operation with a predefined. The list of predefined handle value types is as follows: HS_ADMIN, HS_VLIST, HS_SECKEY, HS_PUBKEY, HS_SITE, HS_SERV, HS_ALIAS, EMAIL, URL, URN, INET_HOST, 10320/LOC. Each handle value line must start with a unique index number, followed by the handle value type from the list above, ttl (the time to live in seconds), the permission set (admin read, admin write, public read, public write), and the value data.

Example:

MODIFY 12345/hdl1
2 URL 86400 1110 UTF8 http://www.handle.net/newadmin.html
3 EMAIL 86400 1110 UTF8 hdladmin@cnri.reston.va.us

MODIFY 12345/hdl2
2 URL 86400 1110 UTF8 http://www.cnn.com/newentainment.html
3 URL 86400 1100 UTF8 http://www.cnn.com/newshow.html

3.7 Authentication Information Format

Operation name is 'AUTHENTICATE'. For secret key authentication:

First line: AUTHENTICATE+space+SECKEY:admin_index:admin_handle
Second line: Password

Example:

AUTHENTICATE SECKEY:301:0.NA/12345
my_password

For private key authentication:

First line: AUTHENTICATE PUBKEY:admin_index:admin_handle

Second line: If your private key was created and encrypted by passphrase, then:
private_key_file_path + '|' + passphrase
Otherwise:
private_key_file_path

Example:

AUTHENTICATE PUBKEY:300:0.NA/12345
c:\home\keyfile|my_pass_phrase
AUTHENTICATE PUBKEY:300:0.NA/12345
c:\home\keyfile

3.8 Session Setup Information Format

Operation name is 'SESSIONSETUP'. The 'USESESSION' flag is mandatory. Remaining fields are used to specify optional public key pair information, session attributes (e.g., "Encrypted", "Authenticated"), "If session fails, use challenge response" flag and "Timeout".

The first line is composed of the following:

SESSIONSETUP

Use the following lines to specify mandatory and optional session setup data:

USESESSION:<session_on_or_off_flag>
PUBEXNGKEYFILE:public_exchange_key_file
PUBEXNGKEYREF:pub_exchange_key_ref_index:pub_exchange_key_ ref_handle
PRIVEXNGKEYFILE:private_exchange_key_file
PASSPHRASE:pass_phrase_for_private_exchange_key
OPTIONS:<encrypt><authenticate><fallback on challenge response>
TIMEOUT:time_out_in_hours

End the 'SESSIONSETUP' operation with a blank line.

In the above lines, the 'USESESSION' flag is mandatory. Either 'PUBEXNGKEYFILE:' or 'PUBEXNGKEYREF:', and 'PRIVEXNGKEYFILE:', 'OPTIONS:', 'TIMEOUT:' are optional. 'PASSPHRASE:' is conditional.

If 'OPTIONS:' is omitted, session messages will neither be encrypted nor authenticated; however, the "If session fails, use challenge response" flag will be set to make sure requests are carried through without session. The 'SESSIONSETUP' line must come first. The remaining lines can be in any order. Do not include a blank line until it ends.

Example 1: Use public exchange key from server.

SESSIONSETUP
USESESSION:1

Example 2: Use public exchange key from a file (client provides exchange keys).

SESSIONSETUP
USESESSION:1
PUBEXNGKEYFILE:c:\hs\bin\PubKey.bin
PRIVEXNGKEYFILE:c:\hs\bin\PrivKey.bin
PASSPHRASE:secret
OPTIONS:111
TIMEOUT:24

Example 3: Use public exchange key from a handle value reference (client provides exchange keys).

SESSIONSETUP
USESESSION:1
PUBEXNGKEYREF:300:0.NA/12345

PRIVEXNGKEYFILE:c:\hs\bin\PrivKey.bin

3.9 Handle Value Line Format

Each handle value line is composed of:

value_index + space + value_type + space + ttl + space + permission_ set + space + value_data

The value index is a unique integer within the specific handle. The value types are: HS_ADMIN, HS_SECKEY, EMAIL, URL, HS_PUBKEY, URN, HS_SERV, HS_VLIST, HS_ALIAS.

ttl: handle's time to live in cache counted by seconds. Default is 86400(24 hours).
Permission set: permission values indicated by 4 characters, '1' is true, '0' is false, order is: admin read, admin write, public read, public write.
Value data: If the handle value data defines an Administrator, its data format is:
ADMIN + space + admin index:admin permission set + admin handle

The admin permission set is twelve characters with the following order: add handle, delete handle, add naming authority, delete naming authority, modify values, remove values, add values, read values, modify administrator, remove administrator, add administrator and list handles.

If the handle value type is one of HS_SECKEY, HS_SERV, HS_ALIAS, EMAIL, URL, URN, its data will be a string. The value data format is:

UTF8 + space + string_content

If the handle value data is a local file, its data format is:

FILE + space + file_path

If the handle value data is a value reference list, its data format is:

LIST + space + index1:handle1;index2:handle2;

Examples:

(1) Where handle value data is an administrative record:

100 HS_ADMIN 86400 1110 ADMIN
300:110011111111:0.NA/12345

Explanation:
100 is index;
HS_ADMIN is type;
86400 is the time to live in cache in seconds;

1110 is the value permissions which allow admin write, admin read, public read;
ADMIN indicates that this value data is an administrator record;

300 is the administrator handle index;
110011111111 defines the administration permissions (add handle, delete handle, no add naming authority, no delete naming authority, modify values, remove values, add values, read values, modify administrator, remove administrator, add administrator, list handles);
0.NA/12345 is the administrator handle name;

(2) Where handle value data is a string:

2 URL 86400 1110 UTF8 http://www.handle.net/

(3) Where handle value data comes from a local file:

300 HS_PUBKEY 86400 1110 FILE
c:\somewhere\pubkey.bin 2 HS_SITE 86400 1110 FILE
c:\somewhere\siteinfo.bin

(4) Where handle value data is a handle value reference list:

1 HS_VLIST 86400 1110 LIST 300:10.50/USR1; 300:10.50/USR2;

(5) Example using some of the registered handle value types:

100 HS_ADMIN 86400 1110 ADMIN 300:111111111111:0.NA/12345
1 HS_SITE 86400 1110 FILE c:\somewhere\siteinfo.bin
2 HS_SERV 86400 1110 UTF8 0.NA/12345
300 HS_PUBKEY 86400 1110 FILE c:\somewhere\publickey.bin
301 HS_SECKEY 86400 1100 UTF8 my password
400 HS_VLIST 86400 1110 LIST 300:12346/USR1; 300:12347/USR2;
7 EMAIL 86400 1110 UTF8 hdladmin@cnri.reston.va.us
8 URL 86400 1110 UTF8 http://www.handle.net
9 DESC 86400 1110 UTF8 Info about this handle

4 Advanced Server Configuration

A handle server can be further configured through the 'config.dct' file located in its installation directory. The 'siteinfo.json' file can also play a smaller role in configuration. The handle server incorporates a handle client, which can be configured by the files in '$HOME/.handle/', notably '$HOME/.handle/config.dct'.

4.1 The .dct file format

Version 8 of the Handle.net software makes increasing use of the JSON format. However, many existing files continue to use the .dct format, which is very similar in spirit.

The .dct file format has 3 basic types.

- Objects: An unordered, white space separated collection of key value pairs enclosed in curly braces, with the '=' character separating the key and the value; the keys must be strings and should be distinct from each other.
- Strings: Double-quoted Unicode
- Lists: An ordered, white space separated sequence of values enclosed in parentheses.

There is no comment syntax for .dct files, but comments can be included by added unused fields to objects, for example "comment" = "This is a comment".

The config.dct file contains various objects, each of which consists of a set of configuration values related to a specific part of the server. These objects are detailed below.

4.7 Top-Level Settings

A handful of settings are at the top level of the configuration object.

- **server_type**: This is a single setting which should generally be "server". An additional setting "cache" exists to configure a cache server. A cache server does not store any handles or answer for a particular prefix, it just acts as a caching gateway. The benefit of clients using a caching server is that they can all make use of the single cache that the caching server provides. Caching servers are not currently in use.
- **interfaces**: This is a list of interfaces that the server should answer on. It should contain one or more of "hdl_udp", "hdl_tcp", and "hdl_http". A dual-stack server, accessible over IPv4 and IPv6, may have "hdl_udp4", "hdl_tcp4", and "hdl_http4" as well. If you wish to disable access via a protocol, remove that protocol from this list. You must also ensure that the protocol is not shown as available in the siteinfo.json file or in the HS_SITE values of prefix handles which point to your server.

Other settings may be put at the top level in order to configure the internal handle client used by the

server when it resolves handles. These are the same settings that can be included in $HOME/.handle/config.dct; see

4.3 hdl_udp_config, hdl_tcp_config, hdl_http_config

For each interface listed in the "interfaces", there should be a corresponding "_config" object. There are several keys used for configuring the interfaces:

- bind_address: The IP address the interface should use. If omitted, the interface will bind to any address.
- bind_port: The port the TCP interface should use. This key is the minimum required to configure an interface.
- num_threads: The number of threads that should be reserved for answering requests.
- max_handlers
- backlog: The number of incoming connections that can be queued while all threads are in use.
- log_accesses: "yes" or "no". If set to "yes", each access on the TCP interface will be logged.

4.4 hdl_udp_config

This section contains the configuration variables for the UDP interface to the handle server.

- bind_address: The IP address the UDP interface should use.
- num_threads: The number of threads that should be reserved for answering requests.
- bind_port: The port the UDP interface should use.
- backlog: The number of incoming connections that can be queued while all threads are in use.
- log_accesses: "yes" or "no". If set to "yes", all accesses on the UDP interface will be logged.
- track_threads: "yes" or "no". If set to "yes", debugging messages concerning thread usage on the UDP interface will be printed to stdout periodically.

4.2 hdl_http_config

This section contains the configuration variables for the HTTP interface to the HANDLE.NET server. You can access this interface by pointing a web browser at your server's IP and HTTP port.

- bind_address: The IP address this interface should use.
- num_threads: The number of threads that should be reserved for answering requests.

- bind_port: The port which the HTTP interface should use.
- backlog: The number of incoming connections that can be queued while all threads are in use.
- log_accesses: "yes" or "no". If set to "yes", each access on the HTTP interface will be logged.
- track_threads: "yes" or "no". If set to "yes", debugging messages concerning thread usage on the HTTP interface will be printed to stdout periodically.

4.2.1 Running an HTTP Proxy

TODO

- query_page: Full path and name of the HTML file that your HTTP proxy should use as the query page. If no page is specified, the default Handle System query page will be used.
- response_page: Full path and name of the HTML file that your HTTP proxy should use as the data response page. If no page is specified, the default Handle System response page will be used.
- error_page: Full path and name of the HTML file that your HTTP proxy should use as error page. If no page is specified, the default Handle System error page will be used.
- virtual_hosts: One or more virtual host names and corresponding pages can be specified in this entry. Details of virtual hosts are specified by subentries inside.

An example of virtual host setting would be:

```
"virtual_hosts" =  {
        "hostname" = "significant.myvirtualhost.com"
        "query_page" = "/home/www/query_page.html"
        "response_page" = "/home/www/response_page.html"
        "error_page" = "/home/www/error_page.html"
}
```

4.5 server_config

This section contains the configuration variables for the server.

- server_admins: A list of administrators with permission to un/home prefixes, perform replication, and perform database backups. each entry in the list should indicate a handle value that contains the HS_PUBKEY of an administrator of the server in the form index:handle. When you register your prefix and submit your sitebndl.zip the will contain the admpub.bin key file. This public key will be placed in your prefix handle at index 300. As such it is typical to

include 300:<your-prefix-hande> in the list of server admins.

- replication_admins: A list of administrators with permission to replicate handles from this server.
- replication_interval: Time interval in milliseconds between mirror server updates.
- replication_authentication: Specified on mirror handle servers. This is the identity of the mirror server. It is used to identify the mirror server when it contacts a primary server to pull transactions. As such it specifies an index, a handle, and either HS_PUBKEY or HS_SECKEY. The common case is HS_PUBKEY which, in this config value, is marked with "privatekey" e.g this value should be of the form"privatekey:index:handle". When using HS_PUBKEY authentication for replication, the server expects the private key to be in the file replpriv.bin.
- replication_accept_prefixes: Specified on mirror handle servers. A list of prefixes used as a filter to only accept transactions with the specified prefix. If omitted all changes on the primary will be applied to the mirror.
- this_server_id: The identification number of this particular server. Used to distinguish from other servers within the same site.
- max_auth_time: The number of seconds to wait for a client to respond to an authentication challenge.
- case_sensitive: "yes" or "no". Whether or not handles are case sensitive. It is highly recommended to always leave this set to "no".
- allow_recursion: "yes" or "no". If set to "yes", the server will act as a proxy for resolving external handles. When a client requests a handle that the server is not responsible for, the server will resolve the handle and return the results, just as if it were stored locally. If allow_recursion is set to "no", the proxy will only allow resolution of handles stored on the local handle server. It is usually safe to set this to "yes", which is the default value.
- preferred_global: A handle server will often need to resolve prefixes from one of the Global Handle Registry servers. Setting this configuration variable to the IP address of a particular global handle server will force the handle server to always use that global handle server or another server within its site (See Section 1.2.1, *Storage*). This may be desirable to reduce latencies if there is a global handle server that is significantly closer geographically than the others.
- max_session_time: Time in milliseconds that an authenticated client session can persist. Minimum is one minute. See the description of sessions in Section 3.8, *Session Setup Information Format* for more information. Defaults to 24 hours.
- replication_timeout: Time in milliseconds before an unresponsive replication operation will timeout. Defaults to 5 minutes.
- storage_type: "bdbje", "jdb", "sql", or "custom"; defaults to "bdbje". This allows manual selection of the storage mechanism used by the server. The "bdbje" storage option uses the Berkeley DB JE hash-style database. The "jdb" storage option is a custom hash style database

that comes included with the Handle System Server distribution. The "sql" option allows use of a SQL database for handle storage. See Section 7.1 *Using a SQL Database for Storage*, for more information. Finally, the "custom" setting directs the handle server to use an external Java™ class specified using the storage_class setting.

- storage_class: The name of the Java™ class that should be used when storage_type is set to custom. This class must implement the net.handle.hdllib.HandleStorage interface included with the distribution. It must also be in the Java™ classpath when the handle server is started.
- server_admin_full_access: "yes" or "no". If set to "no" the "server_admins" will have default permissions at the prefix level. These include the ability to home and unhome prefixes, perform replication, and perform database backups. If set to "yes" the "server_admins" will have the default permissions to do any prefix level operations as well as handle level operations, such as creating, deleting, and modifying handles. When server_admin_full_access is enabled, server_admins will be able to modify and delete existing handles, even if they are not explicitly given permission in the handle.
- allow_list_hdls: "yes" or "no". If set to "no" the 'list handles' operation will be disabled on the server.
- auto_homed_prefixes: A list of prefixes that are automatically homed on server startup.
- enable_monitor_daemon: "yes" or "no". If set to yes an additional service is run that monitors the state of the handle server, free disk space, free ram and details on the number of resolution requests made. This data can be seen by resolving the handle "0.SITE/status" this server. This data is only available in the handle is the request is authenticated by either one of the server_admins or an identity listed in status_handle_admins.
- status_handle_admins: A list of administrators with permission to resolve the handle "0.SITE/status" on this server.

If you wish to configure additional settings specific to the BDBJE handle storage you can add any of the following lines. The values below are the default values for each setting.

- "db_directory" = "bdbje": This tells the BDBJE which folder should contain the database files.
- "bdbje_no_sync_on_write" = "false": This tells BDBJE to write changes to the database, but do not synchronize afterwards. Setting this to true improves performance, with a slight cost in reliability (which may be negated when using a journaling file system).
- "bdbje_enable_status_handle" = "true": This tells the storage module to send database status information in response to a query for the 0.SITE/DB_STATUS handle, as long as that handle doesn't already exist in the database.

4.6 log_save_config

An entry named log_save_config determine the log rotation method. The value of the log_save_config

is a object similar to the following:

```
"log_save_config" = {
        "log_save_interval" = "Weekly"
        "log_save_weekday" = "Wednesday"
        "log_save_directory" = "/var/log/hdl/"
}
```

The log_save_interval can be Monthly, Weekly, Daily, or Never (the default value is "Never"). If it is "Weekly" then there is a log_save_weekday entry that should contain one of Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday, with the default set to Sunday. (English language weekday names are required.)

There is also a log_save_directory with the value a directory/path where the log files are to be saved. If this is a relative path, it will be interpreted as relative to the server directory. The default is to store the log files in the subdirectory "logs" of the server directory.

4.8 Example config.dct File

```
{
"hdl_http_config" = {
        "bind_address" = "132.151.1.132"
        "num_threads" = "15"
        "bind_port" = "8000"
        "backlog" = "50"
        "log_accesses" = "yes"
}
"hdl_tcp_config" = {
        "bind_address" = "132.151.1.132"
        "num_threads" = "15"
        "bind_port" = "2641"
        "backlog" = "50"
        "log_accesses" = "yes"
}
"hdl_udp_config" = {
        "bind_address" = "132.151.1.132"
        "num_threads" = "15"
        "bind_port" = "2641"
        "log_accesses" = "yes"
}
"server_config" = {
        "server_admins" = (
                "300:0.NA/1234"
```

```
        )
        "replication_admins" = (
                "300:0.NA/1234"
        )
        "auto_homed_prefixes" = (
                "0.NA/1234"
        )
        "max_session_time" = "86400000"
        "this_server_id" = "1"
        "max_auth_time" = "60000"
        "case_sensitive" = "no"
}
"log_save_config" = {
        "log_save_weekday" = "Sunday"
        "log_save_time" = "00:00:00"
        "log_save_directory" = "/dspace/handle-server"
        "log_save_interval" = "Weekly"
}
"no_udp_resolution" = "y"
"interfaces" = (
        "hdl_udp"
        "hdl_tcp"
        "hdl_http"
)
"server_type" = "server"
}
```

4.8 Client configuration via $HOME/.handle/config.dct

The $HOME/.handle/config.dct file will be used to configure all clients using the Java Handle client library, when those clients are run by the user with this file under the user's home directory. This includes the internal handle clients used by handle servers run by the user. The following keys can be used in this object to configure clients.

- **trace_resolution**: Setting this to "yes" will cause the handle server to print out debugging messages concerning handle resolution.
- **tcp_timeout**: This is the number of milliseconds that will pass before an outgoing TCP connection fails. This number can be set lower to avoid wasting threads due to broken connections. However, setting too low will cause slow connections to fail unnecessarily.
- **no_udp_resolution**: Setting this to "yes" will prevent the server from resolving external

handles using the UDP based handle protocol. This may be necessary to run a handle server from behind a firewall.

- **ipv6_fast_fallback:** Setting this to "no" will turn off the default fast fallback behavior for IPv6 connections. By default, when handle clients try to connect to IPv6 servers, they also try to simultaneously connect to IPv4 servers after a short delay; whichever connection is established first will be used. When this setting is "no" the handle client will instead wait until the end of ordinary timeout periods to fall back to IPv4.
- **site_filter_keywords:** Advanced option. A list of strings; the client will prefer to contact sites with a site_filter_keywords attribute in their HS_SITE information matching one of the strings.
- **preferred_global_service_handle:** Advanced option. The value is a handle; the client will prefer to talk to GHR servers listed in that handle.
- **auto_update_root_info:** Advanced option. Defaults to yes. If set to no, clients will not automatically update the ~/.handle/bootstrap_handles file, which contains information about the GHR servers.

5 Other Tools and Features

The Handle System Server distribution also includes other small utilities for maintaining a handle server. A selection of them are described below.

5.1 DBTool

DBTool is a graphical utility for operating on a handle server's built-in database. DBTool can be invoked using the following command from the directory containing the "bin" and server configuration directories:

bin/hdl-dbtool <serverdir>
On Windows: bin\hdl-dbtool.bat <serverdir>

where <serverdir> is the directory containing the server configuration and database files.

WARNING: *Do not run this command on a database that is currently in use by a handle server as it could lead to database corruption.*

5.2 DBList/DBRemove

If you would like to operate directly on a handle server's database, but without a GUI, there are two other utilities: DBList and DBRemove.

WARNING: *Do not run these commands on a database that is currently in use by a handle server as it could lead to database corruption.*

DBList will list all the handles in a handle server database. It can be invoked using the following command from the directory containing the "bin" and server configuration directories:

bin/hdl-dblist <serverdir>
On Windows: bin\hdl-dblist.bat <serverdir>

DBRemove will remove a specified handle from the database. It can be run using the command:

bin/hdl-dbremove <serverdir>
On Windows: bin\hdl-dbremove.bat <serverdir>

5.3 Query Resolver

The handle server distribution includes a simple resolver GUI that may be preferable over the resolution facilities in the Admin Tool. It can be run using the following command from the directory containing the "bin" and server configuration directories:

bin/hdl-qresolverGUI
On Windows: bin\hdl-qresolverGUI.bat

There is also a command line resolver that can be run using the command:

bin/hdl-qresolverGUI <handle>
On Windows: bin\hdl-qresolverGUI.bat <handle>

5.4 Test Tool

This tool performs client and local/global handle server diagnostic tests. It is run from the command line and requires certain arguments. The following commands should be run in the same directory containing the "bin" directory.

Client Test with no arguments sends a request to each Global server and tests each interface. Also pings each server within the site and reports average round trip time and percent packet loss.

bin/hdl-testtool client
On Windows: bin\hdl-testtool.bat client

Client Test with prefix argument sends a request to server based on a specified prefix and tests each interface. Also pings each server and reports average round trip time and percent packet loss.

bin/hdl-testtool client <Prefix>
On Windows: bin\hdl-testtool.bat client <Prefix>

Example:

bin/hdl-testtool client 0.NA/12345

Server Test tests a local server by sending a request to the server and testing each interface.

bin/hdl-testtool server <config.dct>
On Windows: bin\hdl-testtool.bat server <config.dct>

Example:

bin/hdl-testtool server /hsj-7.0/svr_1/config.dct

Write Test tests handle operations: add, add value, modify value, delete value, delete. If 'admpriv.bin'

is not located in the same directory as 'config.dct', user will be prompted for location.

bin/hdl-testtool write <config.dct>
On Windows: bin\hdl-testtool.bat write <config.dct>

Example:

bin/hdl-testtool write /hsj-7.0/svr_1

Test All performs server test, write test, and Global sites client test.

bin/hdl-testtool all <config.dct>
On Windows: bin\hdl-testtool.bat all <config.dct>

Example:

bin/hdl-testtool all /hsj-7.0/svr_1

5.5 KeyUtil

The KeyUtil.java program allows decrypting and encrypting of private key files. It can be invoked using the command:

bin/hdl-keyutil <privkey.bin>
On Windows: bin\hdl-keyutil.bat <privkey.bin>

If you chose not to encrypt your key at installation, and later change your mind, use this program to encrypt your existing key, and then send the encrypted file to the Handle System Administrator.

5.6 GetRootInfo

GetRootInfo retrieves a current copy of the Global service information for the Handle System. This is the information found in the 'root_info' file.

Usage:

bin/hdl-getrootinfo <rootserver> <port> <outfile>
On Windows: bin\hdl-getrootinfo.bat <root-server> <port> <out-file>

5.7 GetSiteInfo

GetSiteInfo retrieves the service information for a handle server. This is the information found in the 'siteinfo.bin' file.

Usage:

bin/hdl-getrootinfo <server> <port> <outfile>
On Windows: bin\hdl-getrootinfo.bat <server> <port> <outfile>

6 Replication

The handle server allows for automatic replication of handles to one or more mirror servers. These mirror servers can be used to provide redundancy for resolution or simply as backup for disaster recovery. In sites with multiple servers, handles are distributed evenly across the site. It should be noted that mirror servers cannot be used for handle administration.

When servers are in the same site, the handles for that service are distributed evenly across the site. Mirrors should be able to replicate from either the primary or existing mirrors.

6.1 Setting up a Single Mirror Handle Server

See Chapter 4, *Advanced Server Configuration*, for details on the configuration settings outlined below.

- Run the Setup as explained in the 'INSTALL.txt' file. Be sure to choose "Mirror Server".
- Send the resulting sitebndl.zip to the Handle System Administrator. Setting up a mirror server will have created a replication key pair (replpub.bin and replpriv.bin) in the server directory. The sitebndl.zip includes the replpub.bin. The HSA will add the replication public key value (replpub.bin) to your prefix handle, typically at index 301 (or a different index if 301 is already in use). You will be notified when the change has been made. This is the mirror server's replication identity.
- Modify the mirror's config.dct:
    - server_admins, as described in the 'INSTALL.txt' file
    - this_server_id (if more than 1 server in HS SITE)
    - replication_authentication ("privatekey:index:handle") The is the mirror server's replication identity.
- Edit the primary server's config.dct by adding the replication authentication handle to the primary server's replication_admins group.
- When the Handle System Administrator notifies you that your prefix has been updated, start the mirror server, and restart the primary server so that it will recognize the replication handle. If you do not see handles being replicated immediately, or see errors in the log, contact the HSA for assistance, because once the mirror's site info has been added to your prefix, clients will attempt to use it to resolve handles.

The 'txnsrcsv.bin' file is the 'siteinfo.bin' file from the primary server. The 'txnstat.dct' file will be created once the server has replication information to store. When the mirror server is first started, the server has to "dump" all of the handles from the primary server. When that process is done, the mirror server creates and saves the 'txnstat.dct' file with the current transaction ID from each primary server.

6.2 Setting up a Second Mirror Handle Server

If the first mirror server's db file is relatively large the initial dump of handles may fail. As such it may be necessary to copy the 'bdbje' directory, as well as the 'txnstat.dct' file, to the second mirror's directory before starting the second mirror server. Be sure the first mirror server is shut down while copying the files.

7 Using Custom Handle Storage

This section explains how to configure your HANDLE.NET server to use a database for handle storage other than the built-in database. Instructions follow for using SQL, and in particular PostgreSQL.

7.1 Using a SQL Database for Storage

Using a SQL database as storage for a handle server allows greater control over data deposits as well as permitting complex data query.

7.1.1 Configuring the Handle Server

To configure a handle server with an SQL storage module, first run the Setup program for the HANDLE.NET software. Once the setup process is completed, a directory will exist that contains the files necessary to run the new handle server.

In the directory for the new handle server is a file named 'config.dct' that can be modified using a text editor. The 'config.dct' file contains all of the settings for the handle server. The 'config.dct' file has some server-wide settings as well as several subsections that affect different parts of the server. For example, the 'config.dct' file for most handle servers will have sections named hdl tcp config, hdl udp config and hdl http config. Each of these sections holds the settings for one type of "listener" for the handle server.

Normal handle servers (as opposed to simple handle caching servers or http gateways) will also have a section named "server_config" that maintains the settings for the core part of the server. To tell the server to use an SQL backend for storing and retrieving the handles, add the following value to the server config section:

```
"storage_type" = "sql"
```

Since the specified storage type for this handle server is SQL, some extra settings need to be provided. The following subsection should also be added to the server config section:

```
"sql_settings" = {
   "sql_url" = "jdbc:mysql://localhost/test?user=root&password="
   "sql_driver" = "com.mysql.jdbc.Driver"
   "sql_login" = "root"
   "sql_passwd" = ""
   "sql_read_only" = "no"
}
```

You will need to change the values to suit your particular installation. Here is an informal description of what each item in this section is for:

- sql_url: This setting should specify the JDBC URL that is used to connect to the SQL database. Consult the documentation for the database or JDBC driver for a description of what this setting should look like.
- sql_driver: This is the name of a Java™ class that contains the driver for the JDBC connection. Consult the documentation for the database or JDBC driver for a description of what this setting should look like.
- sql_login: The user name that should be used by the handle server to connect and perform operations on the database.
- sql_passwd: The password that should be used by the handle server to connect and perform operations on the database.
- sql_read_only: a boolean setting (can be "yes" or "no") that indicates whether or not the server should ever need to modify the database in any way. This is a safeguard used for query-only handle servers.

When you start the handle server, you must have the JDBC for your database in your classpath. You can place the jar file (e.g. mysql-connector-java-5.1.29-bin.jar) in the "lib" subdirectory of the unzipped HANDLE.NET distribution.

7.1.2 Example SQL Tables

The default configuration assumes a specific database table setup. The following tables were used for RDBMS storage using MySQL.

```
create table nas (
        na varchar(255) not null,
        PRIMARY KEY(na)
);
create table handles (
        handle varchar(255) not null,
        idx int4 not null,
        type blob,
        data blob,
        ttl_type int2,
        ttl int4,
        timestamp int4,
        refs blob,
        admin_read bool,
        admin_write bool,
        pub_read bool,
        pub_write bool,
        PRIMARY KEY(handle, idx)
);
```

These tables were used for Oracle.

```
create table handles (
        create table nas (
        na raw(512)
);
create table handles (
        handle raw(512),
        idx number(10),
        type raw(128),
        data raw(600),
        ttl_type number(5),
        ttl number(10),
        timestamp number(10),
        refs varchar2(512),
        admin_read varchar2(5),
        admin_write varchar2(5),
        pub_read varchar2(5),
        pub_write varchar2(5)
);
```

7.1.3 Depositing Handles Outside the Handle Server

If you wish to create or modify handles in the SQL database using custom tools, rather than the handle server, you must use all capital letters for data in the "handle" field, since the Handle System is case insensitive.

7.1.4 Using Custom SQL Statements

It is also possible to specify the SQL that is used by the handle server to query the database. Changing these SQL statements is required if you do not use the same setup as above. The SQL handle storage object used by the handle server has default SQL statements that are used to query and update the database. To replace the default SQL statements with custom statements, simply add the corresponding configuration setting to the sql settings section described above. The following is a list of the SQL statements, their configuration setting, default values, and a short description of what the statement is used for.

7.1.4.1 get handle stmt

Default:

```
select idx, type, data, ttl_type, ttl, timestamp, refs, admin_read, admin_write, pub_read,
pub_write from handles where handle = ?
```

Description: This statement is used to retrieve the set of handle values associated with a handle from the database.

Input: The name of the handle to be queried.

Output:

idx positive integer value; unique across all values for the handle

type alphanumeric value; indicates the type of the data in each value

data alphanumeric value; the data associated with the value ttl type byte/short; 0=relative, 1=absolute

ttl numeric; cache timeout for this value in seconds (if ttl type is absolute, then this indicates the date/time of expiration in seconds since Jan 1 0:00:00 1970.

timestamp numeric; the date that this value was last modified, in seconds since Jan 1 0:00:00 1970

refs alphanumeric; list of tab delimited index:handle pairs. In each pair, tabs that occur in the handle part are escaped as \t.

admin read boolean; indicates whether clients with administrative privileges have access to retrieve the handle value

admin write boolean; indicates whether clients with administrative privileges have permission to modify the handle value

pub read boolean; indicates whether all clients have permission to retrieve the handle value

pub write boolean; indicates whether all clients have permission to modify the handle value

7.1.4.2 have na stmt

Default: select count(*) from nas where na = ?

Description: This statement is used to query whether or not the specified prefix is "homed" to this server.

Input: The prefix (eg 0.NA/12345)
Output: One row, with one field. The value of that field is >0 if this server is responsible for the given prefix, or <=0 if not.

7.1.4.3 del na stmt

Default: delete from nas where na = ?

Description: This statement is used to remove a prefix from the list of prefixes for which this server is responsible.

Input: The prefix handle (e.g., 0.NA/12345)
Output: None

7.1.4.4 add na stmt

Default: insert into nas (na) values (?)

Description: This statement is used to add a prefix to the list for which this server is responsible.

Input: The prefix to "home" (e.g., 0.NA/12345)
Output: None

7.1.4.5 scan handles stmt

Default: select distinct handle from handles

Description: This statement is used to get a list of all of the handles in the database.

Input: None
Output: a row for each distinct handle in the database.

7.1.4.6 scan by prefix stmt

Default: select distinct handle from handles where handle like ?

Description: This statement is used to get a list of all handles in the database that have a given prefix.

Input: The prefix, including the slash ('/') character
Output: A row for each distinct handle in the database that starts with the given prefix

7.1.4.7 scan nas stmt

Default: select distinct na from nas

Description: This statement is used to get a list of distinct prefixes that call this server home.

Input: None
Output: A row for each distinct prefix

7.1.4.8 delete all handles stmt

Default: delete from handles

Description: This statement is used to delete all of the handles in the database (!) This is only used when the handle server is acting as a secondary/mirror to a primary service and has gotten so far out

of sync that it tries to delete and recopy the entire database from the primary.

Input: None
Output: None

7.1.4.9 delete all nas stmt

Default: delete from nas

Description: This statement is used to delete all of the prefixes in the database. This is only invoked under the same circumstances as delete all handles stmt.

Input: None
Output: None

7.1.4.10 create handle stmt

Default: insert into handles (handle, idx, type, data, ttl_type, ttl, timestamp, refs, admin_read, admin_write, pub_read, pub_write) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

Description: This statement is used to insert a single handle value into the database.

Input: The fields of the handle and handle value, with the same order and type specified in the default statement above. See get handle stmt for type information for each field.
Output: None

7.1.4.11 handle exists stmt

Default: select count(*) from handles where handle = ?

Description: This statement is used to query whether or not a given handle exists in the database.

Input: The handle being queried
Output: None

7.1.4.12 delete handle stmt

Default: delete from handles where handle = ?

Description: This statement is used to delete a given handle from the database.

Input: The handle being deleted
Output: None

7.1.4.13 modify value stmt

Default: update handles set type = ?, data = ?, ttl_type = ?, ttl = ?, timestamp = ?, refs = ?, admin_read

= ?, admin_write = ?, pub_read = ?, pub_write = ? where handle = ? and idx = ?

Description: This statement is used to update a single handle value with new values. The value to update is identified by the handle and index.

Input:

type - alphanumeric; the new type for the handle value
data - alphanumeric; the new data value
ttl_type - byte/short int; indicates whether the cache timeout is specified in relative or absolute terms (1=absolute, 0=relative)
ttl - numeric; indicates the cache timeout in seconds (if ttl type is absolute then the ttl indicates the expiration date in seconds since Jan 1 0:00:00 1970
timestamp - numeric; date of the last modification to this handle value (should be the current date/time!)
refs - alphanumeric; tab delimited list of references for this handle value. See get handle stmt for encoding.
admin_read - boolean; indicates whether clients with administrative privileges have access to retrieve the handle value
admin_write - boolean; indicates whether clients with administrative privileges have permission to modify the handle value
pub_read - boolean; indicates whether all clients have permission to retrieve the handle value
pub_write - boolean; indicates whether all clients have permission to modify the handle value

7.2 Using PostgreSQL Database

The following instructions, provided courtesy of handle users at the Max Planck Institute for Psycholinguistics, are for setting up a PostgreSQL database for handle storage.

As postgres: createuser -PEDA handleserver

Make sure to define a password for that user. Add to

/var/lib/pgsql/data/pg_hba.conf
host handlesystem handleserver 192.168.0.0/16 md5

(This assumes that your intranet uses 192.168.x.x IP addresses.)

Activate the new account: pg_ctl restart -D /var/lib/pgsql/data/

(You may, depending on your configuration, have to replace /var/lib/pgsql/data here and above with something else.)

As an alternative to pg_ctl restart you may use: /etc/init.d/postgresql restart

Create the database and make sure that it uses Unicode: createdb -O handleserver -E unicode handlesystem

Now use the psql shell to create the tables, etc.:

```
psql -h yourservername -U handleserver -d handlesystem
create table nas (na bytea not null, primary key(na));
create table handles (handle bytea not null, idx int4 not
null, type bytea, data bytea, ttl_type int2, ttl int4, timestamp int4, refs text, admin_read bool,
admin_write bool, pub_read bool,
pub_write bool, primary key(handle, idx));
create index dataindex on handles ( data );
create index handleindex on handles ( handle );
grant all on nas,handles to handleserver;
grant select on nas,handles to public;
\q
```

The \q leaves psql. Note that many columns are bytes, not text.

To backup and restore your handle database, use:

```
to backup: pg_dump handlesystem -F t | gzip -c > handletable.tgz
to list: zcat handletable.tgz | pg_restore -F t -l
to restore: zcat handletable.tgz | pg_restore -F t
```

(With "ddlutils", you can also backup / restore between various databases and XML files, which might be useful for some people.)

To get a description of a database or table, in psql, use:

```
\d (describes the whole database)
\d tablename (describes one table)
```

(As usual, use \q to leave psql again. Note that psql also has nice features like history (cursor up/down) and tab completion.)

To "defragment" and auto-tune for the current contents, use in psql: vacuum analyze handles;

Do this from time to time, especially after larger writes, to gain speed.

The config.dct section for a PostgreSQL database:

```
"storage_type" = "sql"
"sql_settings" = {
"sql_url" = "jdbc:postgresql://YourServerIPAddress/handlesystem"
"sql_driver" = "org.postgresql.Driver"
```

```
"sql_login" = "handleserver"
"sql_passwd" = "yourpassword"
"sql_read_only" = "no"
}
```

When you start the handle server, you must have the JDBC for your database in your classpath. You can place the jar file (e.g. postgresql8jdbc3.jar) in the "lib" subdirectory of the unzipped HANDLE.NET distribution.

For the GUI, as usual:

```
bin/hdl-admintool
On Windows:  bin\hdl-admintool.bat
```

Note: These instructions are included courtesy of handle users at the Max Planck Institute for Psycholinguistics and Lund University Libraries NetLab. It is possible that your settings may differ slightly from those in the examples above.

8 The Handle HTTP Proxy

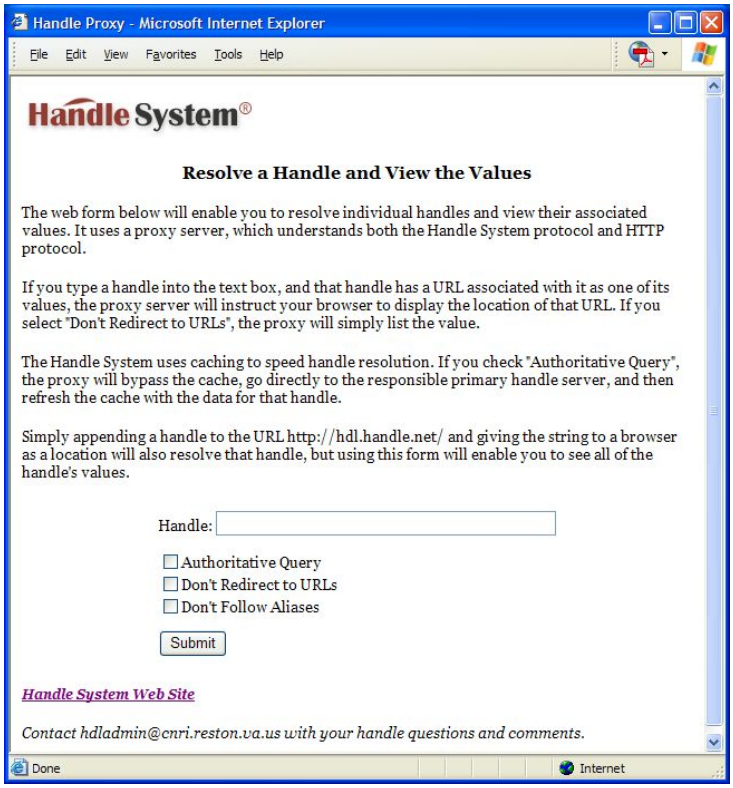
Proxy servers are not part of any Handle System administration or authentication hierarchy. The Handle System protocol does not authenticate any response from a proxy server. Use of a proxy server is a client option, and the client may have to rely on the proxy server to authenticate any service response from Handle System service components. Clients are responsible for setting up their own trust relationship with the proxy server they select.

8.1 Using the Proxy

Unless you disabled the http interface after setting up your handle server, handles can be directly resolved on your handle server using a web browser. A handle server used in this manner is often referred to as a proxy server.

Using HTTP URLs allows handles to be resolved from standard web browsers without additional client software, but requires that the handles be associated with a specific proxy server. If that proxy server changes its DNS name or otherwise becomes invalid, the reference (i.e., the HTTP URL) to the handle will break. Thus selection or use of proxy servers should be carefully evaluated.

You can connect to your server's HTTP interface by opening a URL like http://127.0.0.1:8000. Replace 127.0.0.1 with the IP address or hostname of your handle server. If you changed the HTTP port for the server, replace 8000 with the correct port number. You should see a page like the one below.



Please note that this web form is not part of the standard HANDLE.NET protocol and may change or be completely removed from future versions of the software.

It is also possible to build URLs to the proxy which will automatically resolve or redirect to a specified handle. For a handle server with an IP address of 127.0.0.1 and HTTP interface port 8000 the handle 'my_handle00' can be resolved from a web browser through the URL http://127.0.0.1:8000/my_handle.

If the 'allow_recursion' option is set in the server's configuration (config.dct) file, the HTTP interface will allow resolution of handles which are not stored on the local handle server. When a client requests an external handle, the handle server will resolve the handle and return the results, just as if it were stored locally. This is how public proxy servers like hdl.handle.net and dx.doi.org are configured. If 'allow_recursion' is disabled, the proxy will only allow resolution of handles stored on the local handle server.

8.2 Using Custom HTML Pages

The HANDLE.NET proxy code supports customization of the query, response and error pages. Simple customization of the pages can be performed by modifying copies of the template files included in the 'handle.jar' file. These templates are located in the jar directory 'net/handle/server/html'.

If new templates are created, the server configuration must be modified to use them. See Section 4.1, *hdl_http_config* for instructions.

9 Handle Clients & the Client Library (Java™ Version)

Communicating with the Handle System is accomplished by sending requests to servers which then return a response. To resolve a handle, a ResolutionRequest is sent to a server. To create a handle, a CreateHandleRequest is sent. To modify, remove, or add values to (or from) a handle, a ModifyValueRequest, RemoveValueRequest, or AddValueRequest is sent to a server.

There is an object for each of these requests in the net.handle.hdllib java package. One way to send these messages to a server is to use a HandleResolver object which is located in the net.handle.hdllib package. For most messages, the HandleResolver object will locate the server that your messages should go to, send them, and return the response that was sent by the server. The following is an example that shows one way of programmatically resolving a handle:

```
import net.handle.hdllib.*;
...
// Get the UTF8 encoding of the desired handle.
byte someHandle[] = Util.encodeString("45678/1");

// Create a resolution request.
// (without specifying any types, indexes, or authentication info)
ResolutionRequest request = new ResolutionRequest(someHandle, null, null, null);

HandleResolver resolver = new HandleResolver();

// Create a resolver that will send the request and return the response.
AbstractResponse response = resolver.processRequest(request);

// Check the response to see if the operation was successful.
if(response.responseCode == AbstractMessage.RC_SUCCESS) {

        // The resolution was successful, so we'll cast the response
        // and get the handle values.
        HandleValue values[] =((ResolutionResponse)response).getHandleValues();
        for (int i=0; i < values.length; i++) {
                System.out.println(String.valueOf(values[i]));
        }
}
```

To simply resolve a handle, the much simpler resolveHandle method of the HandleResolver can be used, as shown below.

```
import net.handle.hdllib.*;
...
HandleValue values[] = new HandleResolver().resolveHandle("12345/1", null, null);

for (int i=0; i < values.length; i++){
        System.out.println(String.valueOf(values[i]));
}
```

The HANDLE.NET software distribution include a "simple" package with command line tools to create, delete, and list handles. It also includes programs to home a prefix and trace handle resolution. These programs provide a good starting point and simple guide to developing Java™-based custom handle

client software with the API. Each example program includes steps needed to form a handle request to send to a handle server. The programs are run from the command line and require certain arguments. The following commands should be run from the directory containing the "bin" directory.

(1) Create Handle:

Simple tool for handle creation. It uses public key authentication.

bin/hdl-create <auth handle> <auth index> <privkey> <handle>
On Windows: bin\hdl-create.bat <auth handle> <auth index> <privkey> <handle>

(2) Delete Handle:

Simple tool for handle deletion. It uses public key authentication.

bin/hdl-delete <auth handle> <privkey> <filename_of_file_with_handles_to_delete>
On Windows: bin\hdl-delete.bat <auth handle> <auth index> <privkey> <handle>

(3) List Handles:

Simple tool for listing handles. It uses public key authentication.

bin/hdl-list <auth handle> <auth index> <privkey> <prefix>
On Windows: bin\hdl-list.bat <auth handle> <auth index> <privkey> <prefix>

(4) Trace handle:

Simple tool for resolving a handle.

bin/hdl-trace <handle>
On Windows: bin\hdl-trace.bat <handle>

(5) Home Prefixes:

Simple tool for homing Prefixes. It uses public key authentication.

bin/hdl-home-na <auth hdl> <privkey> <server ip> <NA handle>
On Windows: bin\hdl-home-na.bat <auth hdl> <privkey> <server ip> <NA handle>

10 Configuring an Independent Handle Service

An independent or private handle service (such as a service maintained behind a firewall that is not publicly accessible) operates without contacting the Global Handle Registry. Configuring an independent service requires changes to the client for resolution to occur, and to the server for enabling authentication, homing and administrative tasks to be performed without the GHR.

Resolution Service Providers who wish to operate an independent handle service must notify the Handle System Administrator in advance.

10.1 Client Configuration Details

This section explains how to configure the java client software to resolve handles locally, either through a resolution/caching server, or by directing specific prefixes to a certain service/site.

To specify a local handle server that should be used to process all resolution requests, follow these instructions:

Copy the siteinfo.bin file that describes the site/server where all resolution should be performed into a file called "resolver_site" in the ".handle" sub-directory of the user's "home" directory. This will cause all non-administrative requests to be sent through the site described by siteinfo.bin. This should make resolution faster for organizations that can use the resolution server as a shared cache.

By default, no administrative messages are sent through this site (because administration must be done directly with the site that is responsible for each prefix and cannot be "tunneled"). To force all messages (including administration messages) to go to the local resolution server described above, the user must specify the prefixes that are "homed" on the resolution server. All other prefixes will bypass the local resolution server. To specify the prefixes, do the following:

Create a file called "local_nas" in the ".handle" sub-directory of the users home directory. This file should contain one prefix handle on each line (e.g., "0.NA/11234"), encoded in UTF8 (ASCII is OK as long as there are no special characters). Every request for a handle having a prefix contained in this file will be sent to the local resolution site. If no resolver_site file is provided, the local_nas file is ignored.

A line containing a single asterisk * in the local_nas file will indicate that requests for *all* handles should be sent to the local resolution site.

If there is more than one local handle gateway the methods described in the previous sections will not work. The following is an advanced technique. In this case a local_info or local_info.json file must be placed in the .handle directory on each local client machine. The local_info file can be given in JSON format, as a JSON array, each element of which is an object indicating which naming authorities are managed by a site. Those objects have two properties: "nas", a JSON array of strings, each of which is the prefix handle of a naming authority; and "site", a JSON object representing a site. The format of

the site object is the same as that used by the siteinfo.json file. The script "hdl-convert-siteinfo" can be used to convert siteinfo between binary format (as it is stored in handle values) and JSON format.

In previous versions, the local_info file could only use a binary format. The binary format continues to function. A script "hdl-convert-localinfo" has been provided to convert between the binary format and the JSON format.

Here is an example of a local_info.json file. One site is used for handles under 0.NA/1, a different site is used for handles under 0.NA/3, and both sites are used for handles under 0.NA/2.

```
[
  {
    "nas": [ "0.NA/1", "0.NA/2" ],
    "site": {
      "version": 1, "protocolVersion": "2.1", "serialNumber": 3,
      "primarySite": false, "multiPrimary": false,
      "servers": [
        {
          "serverId": 1,
          "address": "132.151.20.9",
          "publicKey": {
            "format": "base64",
            "value":
"AAAAC0RTQV9QVUJfS0VZAAAAAAAAVAJdgUI8VIwvMspK5gqLrhAvwWBz1AAAAgQD9f1OBHXUSKVLfSpwu7OTn9hG3UjzvR
ADDHj+AtlEmaUVdQCJR+1k9jVj6v8X1ujD2y5tVbNeBO4AdNG/yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophU
PBPuD9tPFHsMCNVQTWhaRMvZ1864rYdcq7/IiAxmd0UgBxwAAAIEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0Hg
mdRWVeOutRZT+ZxBxCBgLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqh
RkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoAAACAWEDRKsfiT3pY+zhrq6bROhVJ+H9ezrs0yjKjweWwklXsQ2HA2XyCh
c0J9eHkL3bLwsG1FpM+vIQ9jG+M3qtASX91oV+je1B3RxmAdnsRbcZ3UsXVn7/LW2K1nABchzTqCV6FqPodufxzj6Rp9ht
8Njc99eMwmnjdxHjZAHONwSI="
          },
          "interfaces": [
            { "query": true, "admin": true, "protocol": "TCP", "port": 2641 },
            { "query": false, "admin": false, "protocol": "UDP", "port": 2641 },
            { "query": true,  "admin": true, "protocol": "HTTP", "port": 8000 }
          ]
        }
      ]
    }
  },
  {
    "nas": [ "0.NA/2", "0.NA/3" ],
    "site": {
      "version": 1, "protocolVersion": "2.1", "serialNumber": 5,
      "primarySite": false, "multiPrimary": false,
      "servers": [
        {
          "serverId": 1,
          "address": "132.151.1.179",
          "publicKey": {
            "format": "base64",
            "value":
"AAAAC0RTQV9QVUJfS0VZAAAAAAAAVAJdgUI8VIwvMspK5gqLrhAvwWBz1AAAAgQD9f1OBHXUSKVLfSpwu7OTn9hG3UjzvR
ADDHj+AtlEmaUVdQCJR+1k9jVj6v8X1ujD2y5tVbNeBO4AdNG/yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophU
PBPuD9tPFHsMCNVQTWhaRMvZ1864rYdcq7/IiAxmd0UgBxwAAAIEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0Hg
```

```
mdRWVeOutRZT+ZxBxCBgLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqh
RkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoAAACBAI4u2BolfvcyIPPtuSaRFtG6NVvqF4BONxHznO3Cg8gtCOG+nt81e
/AFrc3XLA7en+iXMt8LCaZnoxC0XhLa/2vh74MynSoG8iYRHv6D2mTYKltsyR41VavyikoOZ5df6tDHMsibEcQ1htdGO02
gAUIj63cVcTO0Nh8MSfd5ODBu"
          },
          "interfaces": [
            { "query": true, "admin": true, "protocol": "TCP", "port": 2641 },
            { "query": false, "admin": false, "protocol": "UDP", "port": 2641 },
            { "query": true,  "admin": true, "protocol": "HTTP", "port": 8000 }
          ]
        }
      ]
    }
  }
]
```


10.2 Server Side Configuration

By default, authentication, homing, and administering handles on a handle server require your handle server to communicate with the Global Handle Registry. This section describes how to configure your handle server so that administration of handles can be done without communicating with the Global Handle Registry.

(1) Modify the config.dct file:

"server_admin_full_access" = "y"

"allow_na_admins" = "no"

(2) To home a prefix on your handle server without contacting the Global Handle Registry, add the prefix to the handle storage using the DBTool (See Chapter 5.1, *DBTool*).

(3) Once a prefix has been homed, create a new admin handle for it. (The default admin handle is the prefix itself. This default value cannot be used because it requires communication with the Global Handle Registry.) Create the new admin handle using the DBTool, and associate a secret key (password) with it at index 300. For example, if your prefix is 1234, add 0.NA/1234 to the homed prefixes using the DBTool, then create the new admin handle 1234/ADMIN (use upper case when using the DBTool) with a secret key at index 300.

(4) Edit the config.dct file to change the "server_admins" entry to the new admin handle.

(5) Restart the server.

11 Template Handles

A single template handle can be created as a base that will allow any number of extensions to that base to be resolved as full handles, according to a pattern, without each such handle being individually registered. This would allow, for example, the use of handles to reference an unlimited number of ranges within a video without each potential range having to be registered with a separate handle. If the pattern needs to be changed, e.g., the video moves or a different kind of server is used to deliver the video clips, only the single base handle needs to be changed to allow an unlimited number of previously constructed extensions to continue to resolve properly.

When a server receives a resolution request for a handle which is not in its database, it tries to determine if there is template for constructing the handle values.

11.1 The Template Delimiter

First, it looks for a *template delimiter*, which is a string dividing the original handle into a *base* and an *extension*. The delimiter is generally defined in an HS_NAMESPACE value of the prefix handle 0.NA/prefix.

An example:

```
<namespace>
<template delimiter="@" />
</namespace>
```

If there is no namespace info, the server will use the "namespace" value in the "server_config" section of its config.dct configuration file. If there is no value, or if no template delimiter is defined in the namespace (either from the prefix or the config file) the server will use the "template_delimiter" value in the "server_config" section of config.dct.

If a delimiter is found, the server looks up the base handle (i.e., the part before the delimiter). For example, in cnri.test.1/weather@foo, with delimiter @, the base handle is cnri.test.1/weather and the extension is foo.

A delimiter of "/" will enable an entire prefix to be templated. In this case the base handle is considered to have no values.

11.2 Template construction

Any HS_NAMESPACE value in the base handle will override any prefix namespace info — in particular, templates can be put directly into the base handle.

If there is a namespace, it is used to construct the values of the template handle. Each <template> element within the namespace is applied in order. If no template is found, enclosing parent

namespaces are tried. If no template is found at all, the server returns "handle not found".

(1) The template XML itself will contain <value> tags defining the handle values of the eventual result. The <value> tags can specify index, type, and data using attributes. The values of these attributes can refer to the parameters "${handle}", "${base}" and "${extension}". Data can also be specified as the contents of the <value> tag, instead of as an attribute.

(2) Some <value> tags may only be conditionally part of the constructed handle; these are enclosed in <if> tags. The only tests useable in an <if> are string equality and regular-expression matching. With a regular-expression match, various submatches can be referred to in enclosed values with syntax like "${extension[2]}". There is also an <else> tag.

Details of <if> syntax:

- value attribute is some parameter name (e.g., handle, base, extension; inside of a <foreach>, index, type, or data). The syntax value="extension[2]" works inside a nested if.
- parameter attribute is the name of the parameter used to refer to submatches. Default is same as value.
- test attribute is "equals" or "matches"
- negate="true" negates the test
- expression is the string used for equality comparison or RE matching.

(3) Any <notfound/> tag will cause the handle server to return "handle not found".

(4) With <def parameter="param"> ... </def> a new parameter ${param} can be defined. The value of the parameter is obtained by processing the contents of the <def> tag as a template. The data of any constructed handle value if the definition of the parameter. The simplest example is <def parameter="param"><value data="foo"/></def> which defines ${param} to be foo.

(5) Finally, it is possible to produce a value or values for each value already in the base handle. This is useful for having template handles which are identical to the base handle except perhaps for transforming the data of some particular handle value (e.g. a URL). Any values enclosed in a <foreach> tag will be constructed for each value of the base handle. Within the <foreach>, the parameters "${index}", "${type}", and "${data}" can be used to refer to the original handle value from the base handle. Within a <foreach>, <value> tags can omit type or data attributes, in which case the type or data from the original value will be used unchanged.

Here is an example.

```
<namespace>
  <template delimiter="@">
    <foreach>
      <if value="type" test="equals" expression="URL">
```

```
            <if value="extension" test="matches"
expression="box\(([^,]*),([^,]*),([^,]*),([^,]*)\)"                    parameter="x">
              <value data=
"${data}?wh=${x[4]}&amp;amp;ww=${x[3]}&amp;amp;wy=${x[2]}&amp;amp;wx=${x[1]}" />
            </if>
            <else>
              <value data="${data}?${x}" />
            </else>
          </if>
          <else>
            <value />
          </else>
        </foreach>
      </template>
    </namespace>
```

This example produces exactly one value for each value in the base handle. If the type of the original value is "URL", we produce a new value with changed data; the new data depends on the format of the extension. An extension of the form "box(#,#,#,#)" produces a new URL with the four values to be used as query parameters; any other extension is appended as written onto the original URL. If the type of the original value is not "URL", the original value is used unchanged.

For example, suppose we have the above namespace value in 0.NA/1234, and 1234/abc contains two handle values:

1	URL	http://example.org/data/abc
2	EMAIL	contact@example.org

Then 1234/abc@box(10,20,30,40) resolves with two handle values:

1	URL	http://example.org/data/abc?wh=40&ww=30&wy=20&wx=10
2	EMAIL	contact@example.org

For more on the RE language, see
http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html.

11.3 Template handles by reference

XML of the form

```
<template ref="10:abc/def" />
```

will be taken to refer to the handle value of index 10 of handle "abc/def". The data of the handle value is parsed as XML and interpreted as above. Too much recursion, or a "handle not found" result,

any resolution error, or failure to parse the referenced tag, all lead to a "handle not found" result.

12 The 10320/loc Handle Type

For handles with multiple URL values, the proxy server (or web browser plug-in) simply selects the first URL value in the list of values returned by the handle resolution. Because the order of that list is nondeterministic, there is no intelligent selection of a URL to which the client would be redirected. The 10320/loc handle value type was developed to improve the selection of specific resource URLs and to add features to the handle-to-URL resolution process. (Note that the prefix '10320' has been set aside by the Handle System administrator for handle application types.)

Type 10320/loc specifies an XML-formatted handle value that contains a list of locations. Each location has a set of associated attributes that help determine if or when that location is used. The overall list of locations can include hints for how the resolving client should select a location, including an ordered set of selection methods. Resolvers can apply each known selection method, in order, to choose a location based on the resolver's context (the HTTP request in the case of the proxy server) and the attributes of each location.

The attributes for the set of locations, as well as each location entry in the set, are open-ended to allow for future capabilities to be added in a backwards-compatible way. A small number of attributes have been defined as "standard" that all resolvers should understand.

At the top level of the XML structure are the following defined attributes:

chooseby (optional)

The chooseby attribute identifies a comma-delimited list of selection methods. If no chooseby attribute is specified then the default (currently "locatt,country,weighted") is assumed.

For each location the following attributes are defined:

href (required)

The URL for the location.

weight (optional)

The weight (from zero to one) that should apply to this location when performing a random selection. Setting the weight attribute to zero results in the location not being selected unless a) it is explicitly referenced by another attribute; b) there are no other suitable locations; or c) the location is selected based on one of the other selection methods, such as country or language. If a location has no weight attribute then it is assumed to have a weight of one.

The currently defined selection methods are:

locatt

Select only locations from an attribute passed in the proxy/handle-URI link. If someone constructs a link as hdl:123/456?locatt=id:1 then the resolver will return the locations that have an "id" attribute of 1 (i.e., the second location in the resolution example below).

country

Selects only locations that have a 'country' attribute matching the country of the client. If no matching locations are found then this selects locations that have no country attribute (i.e., not a mismatch). The Proxy determines the country of the client using a GeoIP lookup.

language

Selects only locations that have a 'language' attribute matching the language in the Accept-Language header of the request.

weighted

Selects a single location based on a random choice. The Proxy will observe the 'weight' attribute for each location, which should be a floating point non-negative number. The weighting allows for a very basic load balancing, but is also a way to ensure that some locations can only be addressed directly (for example by country or locatt/attributes). If the weighted selection method is applied to locations that all have non-positive weights, then this selects one of the remaining locations randomly while disregarding location weights.

The Proxy will iterate over the known selection methods, in order, until a single location has been selected. After each iteration the Proxy will take one of four steps:

- if there is only one remaining location element, it is returned as a redirect;
- if there are no remaining location elements, the Proxy reverts to the location elements as they were before the last method was applied;
- if there are multiple location elements the Proxy will apply the remaining selection methods to those locations;
- if there are no more selection methods to try, the weighted random selection method is applied, which is guaranteed to return a single location. In a sense, the weighted random is always the "fallback".

For handle 123/456, with a value type 10320/loc that has this list of location attributes:

```
<locations>
 <location id="0" href="http://uk.example.com/" country="gb" weight="0" />
 <location id="1" href="http://www1.example.com/" weight="1" />
```

```
  <location id="2" href="http://www2.example.com/" weight="1" />
</locations>
```

the following selections could be made:

Reference: 123/456 from a client located in the UK
Result: The "country" selection method selects the first location based on the 'country' attribute of the first location and the client's position.

Reference: 123/456 from a client located outside the UK
Result: The "country" selection method removes the first location from consideration based on its 'country' attribute and chooses one of the last two locations using the "weighted" random selection method.

Reference: 123/456?locatt=id:1
Result: The second location is used based on the "locatt" selection method and the 'id' attribute.

Reference: 123/456?locatt=id:0
Result: The first location is used based on the "locatt" selection method and the 'id' attribute. The resolver never gets to the "country" selection method as the "locatt" selection method resulted in only a single matching location.

Reference: 123/456?locatt=country:uk
Result: The first location is used based on the "locatt" selection method and the 'country' attribute.

Reference: 123/456?locatt=country:us
Result: The "country" selection method removes the first location from consideration based on its 'country' attribute, finds no US-specific location, and chooses one of the last two locations using the "weighted" random selection method.

13 Splitting a Handle Server

A single handle server can contain in excess of a billion handles. If you have a need to host more handles than can be stored on a single machine it is possible to split a handle server onto multiple machines. If you have this need please contact hdladmin@cnri.reston.va.us for assistance.