HANDLE.NET® (version 9)

Technical Manual
Version 9
Preliminary edition

Corporation for National Research Initiatives
June 2018
hdl:20.1000/113

Version Note

Handle.Net Version 9 Software, released in 2018, was designed to improve performance and compatibility under recent versions of Java. The Version 9 software may cause handle server storage changes which prevent downgrading to Handle.Net Version 8.1 or earlier. See the Version 9 Release Notes at http://www.handle.net for details.

Handle.Net Version 8.1 Software, released in 2015, constituted a major upgrade to the Handle.Net Software. 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 8.1 Release Notes at http://www.handle.net for details.

Please send questions or comments to the Handle.Net Administrator at hdladmin@cnri.reston.va.us, or your prefix administrator.

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, an identifier space, 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 identifier space, 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.

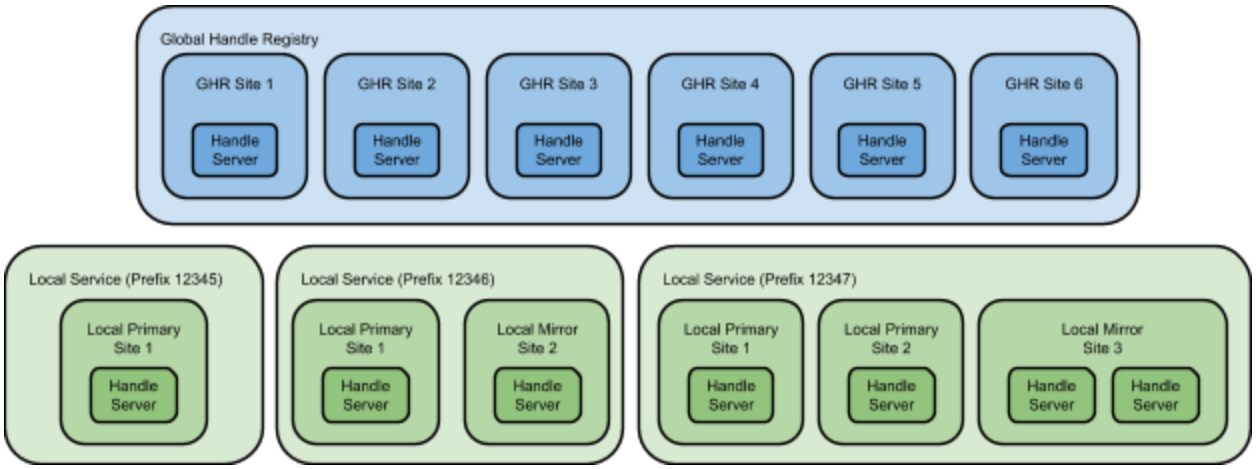
In general, handles are case sensitive. However, any handle service may define its identifier space such that all ASCII characters within any identifier are case insensitive. This is recommended and the default for Handle.Net server software. The Global Handle Registry® (GHR) guarantees that handles 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 identifier space can be considered as a superset of many local identifier spaces, with each local identifier space 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 identifier space can join the global handle identifier space 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 identifier space for that prefix. Any local name must be unique under its local identifier space. 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 how to

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

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.

1.2.1 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.

1.2.2 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.

1.3 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 Chapter 5) 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 Chapter 14. Finally, Handle servers incorporate a modular extension framework in the form of a Java servlet container. 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 4, *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.

1.4 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 identity. A handle 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 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.

1.4.1 Types of Authentication

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 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 record. Read permissions on the handle need to be restricted to ensure the secrecy of the secret key.

1.4.2 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.

1.4.3 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.

The Handle.Net Software 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 and 9 software by default encrypts session communications using AES-128.

For instructions on enabling session encryption see the Handle Tool User Manual and Chapter 4, *Batch Operation*.

2 Upgrading an Existing Handle Server to Version 9

The Handle.Net handle server software has been designed so that new versions can always be run on existing server directories. Thus it is always possible to simply:

(1) Stop the handle server process.
(2) Run the new server software on the same server directory.

Certain changes are recommended however.

Create a backup of the server directory in case you may wish to downgrade to Handle.Net version 8.1. Handle.Net version 9 upgrades the Berkeley DB JE version for the first time in many years. After running a handle server using Handle.Net version 9, the storage directories "bdbje", "replicationDb", and "txns", when present, will all be upgraded automatically to the higher version format. After this, it will no longer be possible to open the same storage using earlier versions of the Handle.Net software. If you wish to leave open the possibility of downgrading, you should make a backup.

Consider pruning transaction storage. By default, the "txns" directory of a Handle server is allowed to grow indefinitely. As of version 9, it is possible to configure the Handle server to automatically prune older transactions. Set property "txnlog_num_days_to_keep" in "server_config" in config.dct to be a number of days after which remembered transactions will be deleted. The default of 0 means to keep forever. Mirrors which are more than the configured number of days out of date will need to redump from that primary.

Previous migration steps given for upgrading to Handle.Net 8.1 are as follows.

Ensure the use of BDBJE storage on older servers. Handle.Net version 6 and previous software defaulted to a built-in storage mechanism called JDB which is deprecated in favor of BDBJE. If your server directory includes files "handles.jdb" and "nas.jdb" you are running the JDB storage; if it includes a subdirectory "bdbje" you are running the BDBJE storage. The BDBJE storage offers significantly better performance for large numbers of handles. If you are using the legacy JDB storage, you can upgrade using the following command (after the handle server is stopped):

On Unix-like systems: bin/hdl-migrate-storage-to-bdbje /hs/svr_1
On Windows systems: bin\hdl-migrate-storage-to-bdbje.bat \hs\svr_1

Add the new browser-based administration tool. This is done by default for new server setups. For an existing server:

(1) Create a subdirectory "webapps" if needed in the server directory.
(2) Copy "admin.war" from the software distribution into "webapps".

This can be done while the server is already running the new version of the Handle.Net software.

Remove "backlog" properties from config.dct. Earlier versions of the Handle.Net software inserted "backlog" properties in config.dct with the value "5". This can be too low for a busy server. Remove these properties to use the default, currently 50.

Convert siteinfo.bin to editable siteinfo.json. A handle server's site information is the publicly available information used by handle clients to connect to the server. It is also used as part of server configuration. Earlier versions used a non-human-readable binary format for this file; version 8.1 and later prefers a human-readable and human-editable format. To make this change, use this command

> *On Unix-like systems:* bin/hdl-convert-siteinfo < /hs/svr_1/siteinfo.bin > /hs/svr_1/siteinfo.json
> *On Windows systems:* bin\hdl-convert-siteinfo.bat < /hs/svr_1/siteinfo.bin > /hs/svr_1/siteinfo.json

Once siteinfo.json is created, you need to delete siteinfo.bin, which otherwise takes precedence. Once you have an editable siteinfo.json, you can also edit the file to update the "protocolVersion" to "2.10", indicating that your server is running version 8.1 or later. You can also send your updated siteinfo.json to the technical contact of your prefix registrar (for instance, hdladmin@cnri.reston.va.us), noting your handle prefix, and requesting that the public-facing HS_SITE value be updated with the new version information. (This is not strictly necessary, as version 8 and later clients can negotiate the version; knowing the best version available in advance is an optimization.)

Consider inserting "allow_recursion" = "yes" in the "server_config" section of config.dct. The old default for the "allow_recursion" server configuration key is "yes"; the new default is "no". The "no" is the best choice for handle servers which are expected to be asked only for handles for which they are responsible. If you use your handle server as a proxy for resolving handles at other servers, however, you will need to add "allow_recursion" = "yes" to the "server_config" section of config.dct. If you're not sure whether you need this, do not add it.

3 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 values. See also the Handle Tool User Manual.

3.1 Installing Java™

The Handle.Net software requires Java™ to run. If you already have Java™ installed, confirm that you have version 8 or later. You can check this by running the command 'java -version'. Java version 8 is the minimum as of Handle.Net Version 9.

3.2 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 built-in decompression capability of Windows.

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/handle-9.0.0. All of the executable/batch files necessary to setup the server are in the directory /hs/handle-9.0.0/bin.

3.3 Choosing an Installation Directory

Create a folder for the server configuration. We will assume this folder is called "/hs/svr_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 3.7, *Installation Directory*.

3.4 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 "handle-9.0.0" directory and execute the following commands:

On Unix-like systems: bin/hdl-setup-server /hs/svr_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 send to your prefix administrator. 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 administrator concerning your prefix.

3.5 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 by your prefix administrator). 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 property 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/svr_1
On Windows systems: bin\hdl-server.bat \hs\svr_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.

3.5.1 Homing your prefix with the handle 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.

3.6 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 README.txt file (see

Section 3.5, *Running the Handle 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 record for 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

3.7 Installation Directory

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

3.7.1 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

3.7.2 logs/error.log

As suggested by its name, this file contains a log of server errors. It will often be requested by your prefix administrator when asked about unexpected handle server behavior.

3.7.3 config.dct

This is the server configuration file. See Chapter 5, *Advanced Server Configuration*, for more information.

3.7.4 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 5, Advanced Server Configuration, for more information.

3.7.5 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.

3.7.6 replicationDb/

Additional storage of most recent modification timestamps for handle records. This is only used for conflict resolution in multi-primary replication.

3.7.7 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.

3.7.8 delete_this_to_stop_server

A convenient way to stop a running server is to delete this file. The server will shut down cleanly, which may not be immediate after the file is gone.

3.7.9 txns/

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

3.7.10 txn_id

This file stores the most recent transaction number.

3.7.11 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.

3.7.12 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.

3.7.13 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.

3.7.14 serverCertificate.pem, serverCertificatePrivateKey.bin

This is an X.509 certificate (or certificate chain) used by the server when responding to HTTPS requests, and the corresponding private key. By default the server will automatically generate a self-signed certificate.

3.7.15 webapps, webapps-temp, webapps-storage

The version 8 and later 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.

3.7.16 txnsrcsv.bin

In former versions of the software, this was the usual method of configuring replication. This file is the siteinfo, in binary format, of the primary server. It is now possible instead to configure replication by reference to site information in HS_SITE values in handle records. See Chapter 7, *Replication*.

3.7.17 siteinfo.bin

Handle servers which predate version 8 may have a siteinfo.bin file. This carries the site information in a binary format. The siteinfo.json file is more convenient since it is human readable and editable. The Handle.Net software includes a utility 'hdl-convert-siteinfo' for converting between the two:

```
hdl-convert-siteinfo < siteinfo.bin > siteinfo.json
hdl-convert-siteinfo < siteinfo.json > siteinfo.bin
```

If both files exist, siteinfo.bin will take precedence. It is recommended to convert to json and delete the legacy siteinfo.bin format.

3.8 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 Section 5.8, *Client Configuration.*

3.8.1 $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.

3.8.2 $HOME/.handle/root_info

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

3.8.3 $HOME/.handle/config.dct

Client-level configuration. Some configuration may be useful for running a server. For more information see Section 5.8, *Client Configuration.*

3.9 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. Note that the server will attempt to shut down cleanly which may not be immediately after the deletion of the file; check running processes to be sure.

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 your prefix administrator if you plan to shut down your server permanently.

3.10 Inactive Prefixes

Resolution Service Providers are required to ensure that the identifiers they create will resolve. If a handle service is shut down, your prefix administrator must be notified in advance, and arrangements

must be made to enable clients to correctly inform users of the status of those handles.

3.11 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 your prefix administrator for assistance.

4 Batch Operation – Command Line

It may be desirable to perform more handle operations than it is possible to perform using GUI administration tools. In those cases it is possible 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>] [-verbose]
On Windows: bin\hdl-genericbatch.bat <batchfile> [<LogFile>] [-verbose]

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.

4.1 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 4.9, *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

4.2 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

4.3 (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

4.4 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 4.9, *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
```

4.5 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
```

4.6 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 4.9, *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

4.7 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

4.8 Session Setup Information Format

Note that sessions are enabled by default, so in general it is not necessary to include session operations in batch files.

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

4.9 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
```

5 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'.

5.1 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.

5.2 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 Section 5.8, *Client Configuration.*

5.3 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 properties 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 interface should use. This property is the minimum required to configure an interface.
- num_threads: The number of threads that should be reserved for answering requests.
- max_handlers: The maximum number of threads which will be used for answering requests.
- backlog: The number of incoming connections that can be queued while all threads are in use. In general it is recommended to leave this at the default value of 50. The old default of 5 is now considered too small and should be changed or removed.
- log_accesses: "yes" or "no". If set to "yes", each access on the TCP interface will be logged.
- max_idle_time: On a TCP interface, the socket timeout in milliseconds that the TCP socket will be held open waiting for bytes to read.

5.4 HTTP Configuration

The "bind_address", "bind_port", and "log_accesses" keys of the hdl_http_config object have the same meaning as for other interfaces. As of Handle.Net version 8, the HTTP interface of the Handle Server is provided by an embedded Jetty Java Servlet container. It can be further configured using a jetty.xml file in the handle server directory.

Some properties of the HTTP interface can be configured using a block "http_config" inside the "server_config" object of config.dct. Properties inside "http_config" can include:

- enable_trace: If set to "no", HTTP TRACE is disallowed.
- enable_proxy: If set to "no", the HTTP-to-Handle proxy UI will be unavailable. The HTTP JSON API, as well as native Handle resolution tunneled over HTTP, will still be available.
- headers: A table with all keys and values being strings, each key-value pair of which will be added as a header to all HTTP responses.
- robots_txt: The full path to a file which will be served when robots.txt is requested.
- favicon: The full path to a file which will be served when favicon.ico is requested.
- remote_address_header: A request header (for example "X-Forwarded-For") which will be used

for logging of client IP addresses.

- remote_address_internal_proxies: A list of trusted proxies in CIDR subnet notation; if this list exists and is non-empty, only those proxies will be trusted to set the remote_address_header.

5.4.1 Running an HTTP Proxy

It is possible to use a handle server as a HTTP-to-Handle proxy, which behaves much like the global hdl.handle.net proxy. By default the handle server will only resolve handles stored on the server. If it is desired for the proxy to resolve any handles anywhere, the server configuration option "allow_recursion", in the "server_config" object, must be set to "yes".

For a proxy server for public use, it is recommended to run a standalone proxy server, which can be downloaded from Handle.Net and run in any Java Servlet container.

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. 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_handle' can be resolved from a web browser through the URL http://127.0.0.1:8000/my_handle.

To support legacy built-in HTTP proxy behaviors, the Handle.Net version 8 and later handle server software supports the following configuration options on the hdl_http_config object.

- 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 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 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 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"
}
```

5.5 server_config

This section contains the configuration variables for the server.

- server_admins: A list of administrators with (at least) permission to un/home prefixes and perform replication. 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-handle> in the list of server admins.

  If "server_admin_full_access" = "yes", the server_admins will have all permissions over every handle record in the handle server's store.

- 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, and perform replication. 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.
- auto_homed_prefixes: A list of prefixes that are automatically homed on server startup.
- replication_admins: A list of administrators with permission to replicate handles from this server.
- replication_interval: Time interval in milliseconds between mirror server updates; default is one minute.
- replication_timeout: Time in milliseconds before an unresponsive replication operation will timeout. Defaults to 5 minutes.
- 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"; 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.

  The replication_authentication on the mirror should correspond to (one of) the replication_admins on the primary. See Chapter 7, *Replication.*
- replication_sites_handle, replication_site_handle_value, replication_pull_other_transactions: Configuration properties for replication. See Chapter 7, *Replication.*
- this_server_id: The identification number of this particular server. Used to distinguish from other servers within the same site. In a single-server site (by far the common case) this is not

significant.

- 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. The default value is "no", in order to facilitate debugging.
- max_session_time: Time in milliseconds that an authenticated client session can persist. Minimum is one minute. See the description of sessions in Section 4.8, *Session Setup Information Format* for more information. Defaults to 24 hours.
- 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 was the default in earlier Handle.Net software distributions and is now deprecated. The "sql" option allows use of a SQL database for handle storage. See Section 8.1 *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. Generally that can be done by including the relevant jar file in the "lib" directory of the extracted Handle.Net software distribution.
- allow_list_hdls: "yes" or "no". If set to "no" the 'list handles' operation will be disabled on the server.
- 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.
- txnlog_num_days_to_keep: a number of days after which remembered transactions will be deleted. The default of 0 means to keep forever. Mirrors which are more than the configured number of days out of date will need to redump from that primary.

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.

5.6 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 "Monthly"). 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.

5.7 Example config.dct File

```
{
"hdl_http_config" = {
        "bind_address" = "132.151.1.132"
        "bind_port" = "8000"
        "log_accesses" = "yes"
}
"hdl_tcp_config" = {
        "bind_address" = "132.151.1.132"
        "num_threads" = "15"
        "bind_port" = "2641"
```

```
        "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" = "yes"
"interfaces" = (
        "hdl_udp"
        "hdl_tcp"
        "hdl_http"
)
"server_type" = "server"
}
```

5.8 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.

6 Other Tools and Features

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

6.1 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.* (The default BDBJE handle storage will simply prevent its use.)

6.2 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 DBRemove on a database that is currently in use by a handle server as it could lead to database corruption.* (The default BDBJE handle storage will simply prevent its use.) The DBList tool is safe to use with the BDBJE handle storage, *but may not be with other handle storage modules.*

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>

6.3 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>

6.4 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 /hs/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 /hs/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 /hs/svr_1

6.5 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.

6.6 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>

7 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 general, handles under a particular prefix are available at a *handle service* which may comprise multiple *handle sites*, each of which is a replica of the others. Rarely, each handle site may comprise multiple handle servers, such that the handles are distributed evenly across the servers; in most cases a handle site is a single handle server.

Handle server replication is done by having mirrors *pull* new transactions from primaries at regular intervals, by default every minute. The guarantee is *eventual consistency:* at any given point in time, some servers in the service may have out of date information, but over time (generally within one replication interval, usually one minute) servers will catch up in a quiet system.

When a new mirror is added to a handle service, it must be initialized manually, before starting the server, with a command to *dump from primary.* It will retrieve the entirety of the primary's handle storage as well as the current replication state. The mirror can then be started and replication will proceed from the initial point.

Advanced topics: Handle replication supports *multi-primary* configuration where handles can be administered at more than one server. In this case, all mirrors (which now include the primaries themselves) will in most cases pull from all primaries. Handle servers can also be configured to use *proxied mirroring* where they can obtain transactions from any server in the replication group; by default, they can only obtain transactions from the server on which the transaction originated.

7.1 Setting up a Single Mirror Handle Server

- Run the Setup as explained in the README.txt' file. Be sure to choose "Mirror Server".
- Send the resulting sitebndl.zip to your prefix 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 administrator 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 README.txt' file
    - 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 value

index:handle to the primary server's replication_admins group.

- When your prefix administrator notifies you that your prefix has been updated, you need to ensure that the new values indicating that the mirror is allowed to replicate are visible to the primary. Either wait for the primary's cache to clear (generally one hour) or restart the primary.
- Run the hdl-dumpfromprimary command to initialize the mirror:

  On Unix-like systems: bin/hdl-dumpfromprimary /hs/srv_1
  On Windows systems: bin\hdl-dumpfromprimary \hs\svr_1

  You can check the logs/error.log file to confirm that the dump is successful.
- Now start the mirror server. If you do not see handles being replicated immediately, or see errors in the log, contact your prefix administrator 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. The mirror server creates and saves the 'txnstat.dct' file with the current transaction ID from each primary server.

If the handle store is sufficiently large the initial dump of handles may fail. It is possible to manually initialize the replication state of the new mirror by copying the files directly. Please contact your prefix administrator for assistance.

7.2 Further Replication Configuration

Setting up replication requires setting up the source of the transactions, and the appropriate authentication and authorization.

7.2.1 Configuring the Transaction Sources

Absent any other configuration in the "server_config" section of the mirror's config.dct file, the mirror will pull transactions from the site whose site information is in the txnsrcsv.bin file. This is automatically set up by the server setup script.

Alternately the transaction sources can be configured manually, using one of these two properties in the "server_config" section of config.dct:

- "replication_sites_handle" : A handle, any primary HS_SITE values of which are used as transaction sources
- "replication_site_handle_value": An index:handle reference to a particular HS_SITE value, which will be used as the transaction source.

7.2.2 Authentication and Authorization

Each mirroring server has a replication identity, which is uses to authenticate to servers from which it pulls. This identity is defined in the "replication_authentication" property, together with the replpriv.bin file. The "replication_authentication" property has the form "privatekey:index:handle", e.g. "privatekey:301:0.NA/1234", indicating the mirror will authenticate using the private key corresponding to the public key at index 301 of the handle record for 0.NA/1234. The private key is in the file replpriv.bin.

It is possible (although not recommended) to use secret key authentication instead of public/private key. In this case use "secretkey" instead of "privatekey" in "replication_authentication", and the secret key should be the contents of the replsec.bin file.

In order for replication to succeed, the identity used by the mirror must be considered authorized to replicate by the primary. In the primary configuration the property "replication_admins" is a list of authenticated "index:handle" references to handle values. These handle values must either include the replication_authentication index:handle of the mirror, or else must be HS_VLIST handle values which recursively include the authentication index:handle of the mirror.

7.3 Multi-primary Replication

It is possible to configure a handle service so that more than one of the servers can accept administrative requests. Such a service is called multi-primary.

In a multi-primary service, there is a possibility of conflicts if the same handle record is administered on multiple handle servers. Handle replication eventual consistency guarantees that each server in the service will settle on the same handle record. Which edit will actually take effect is determined by timestamp. It is recommended that handle applications using multi-primary services provide some external mechanism for avoiding the need for conflict resolution.

To set up a multi-primary service, create a handle containing at least the primary HS_SITE values for the servers in the replication group. Configure each server to use that handle as its "replication_sites_handle" value. Since primaries do not by default replicate, it is also necessary to either set the "multiPrimary" flag to true in siteinfo.json, or set "do_replication" = "yes" in the "server_config" section of config.dct.

By default, servers in a replication group will pull transactions separately from each primary server. It is also possible to configure some servers (either mirrors or primaries) to pull all transactions, no matter where they originated, from only one server in the group. That server should be identified by the index:handle of its HS_SITE in the "replication_site_handle_value" property; additionally "replication_pull_other_transactions" = "yes" should be set.

8 Using Custom Handle Storage

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

8.1 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.

8.1.1 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.
- store_handle_as_string, store_na_as_string, store_handle_value_type_as_string: By default, handles, prefixes, and handle value types are stored as bytes. Some JDBC drivers may not behave correctly when using a varchar column type in the default configuration. Depending on your JDBC driver and how your tables are set up, you may find it preferable to store these as strings; it should allow the use of a varchar column type with any JDBC driver. Note that handle value data can not in general be assumed to be representable as a string and should have a blob or raw column type in the SQL tables.
- trace_sql: set to "yes" for extra debugging output about SQL calls made by the server.

When you start the handle server, you must have the JDBC driver 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.

8.1.2 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 blob,
        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)
);
```

8.1.3 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 most handle servers are case insensitive. You should also note the handle replication will not work correctly if the storage is manipulated directly. As an alternative, you can mirror the storage directly using database tools.

8.1.4 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.

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

have na s**tmt**

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 (e.g. 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.

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

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

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.

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

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

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

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

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

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

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

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

8.2 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.

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

The Handle.Net software includes a Client Library for Java development. Developers wishing to produce applications incorporating handle clients but not using Java may wish to consider the HTTP REST API; see Chapter 14.

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

(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> <auth index> <privkey> <server ip> <NA handle>
On Windows: bin\hdl-home-na.bat <auth hdl> <auth index> <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 their prefix 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.json 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.json. 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":
"AAAAC0RTQV9QVUJfS0VZAAAAAAAVAJdgUI8VIwvMspK5gqLrhAvwWBz1AAAAgQD9f1OBHXUSKVLfSpwu7OTn9hG3UjzvR
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":
"AAAAC0RTQV9QVUJfS0VZAAAAAAAVAJdgUI8VIwvMspK5gqLrhAvwWBz1AAAAgQD9f1OBHXUSKVLfSpwu7OTn9hG3UjzvR
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" = "yes"

"allow_na_admins" = "no"

"template_ns_override" = "yes"

(2) To home a prefix on your handle server without contacting the Global Handle Registry, add the prefix to the handle storage using the "auto_homed_prefixes" configuration option, or else the DBTool (See Section 6.1, *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 HS_NAMESPACE value, 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 information (either from the prefix or the config file) the server will use the "template_delimiter" value in the "server_config" section of config.dct. The server will also prefer the values from "server_config" (and decline to resolve 0.NA/prefix) if there is a "template_ns_override" value set to "yes".

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 HS_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 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 HS_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://docs.oracle.com/javase/8/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 Value 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.

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,address,country,score,weighted") is assumed.

For each location the following attributes are defined:

href (required)

The URL for the location.

score (optional)

A number which will be used to rank locations not otherwise filtered out. The "score" selection method will retain only those locations with the highest score available.

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 locatt. 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).

address

Selects only locations that have an 'addresses' attribute which matches the IP address of the client. The 'addresses' attribute should have a comma-separated list of IP addresses ranges in CIDR subnet format.

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.

score

Selects only locations that have the highest available 'score' attribute from among remaining locations.

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 Handle Server Backup

Previous versions of this document have recommended the use of a backup/checkpoint network request to be sent to the handle server by an authenticated server administrator. This was necessary for the backup of the legacy JDB storage module, but is now obsolete.

A handle server using BDBJE storage can be backed up using filesystem operations. BDBJE supports hot backup as long as the files in use by BDBJE are copied in alphabetical order. This is done, for instance, by the command **rsync**, which we recommend for handle server backup.

In the simplest case, the "bdbje" subdirectory of a running handle server can be backed up using rsync. That suffices for reconstituting the handle server, as long as replication is started from scratch. So the bdbje directory can be backed up on the primary server, and then if a restore from backup is required, all mirror servers can be re-synced using the hdl-dumpfromprimary command.

In some cases it is also possible to back up the replication state in order to allow backup and restore without restarting replication from scratch.

A mirror server (on which administrative operations are not performed) or a **quiescent** primary server (where it can be guaranteed that no administrative operation are being performed) can be backed up by copying the following files/directories in the specified order (when they exist):

(1) txnstat.dct
(2) txns/
(3) txn_id
(4) replicationDb/
(5) bdbje/

When a mirror is backed up in this way and then restored from backup, it will automatically resume replicating from the primary server. When a primary is ensured to not be receiving admin requests, and is backed up in this way and then restored from scratch, it will automatically resume replicating to mirror servers. In both cases the usual eventual consistency guarantees of handle replication apply.

If the primary can not be guaranteed to be quiescent, the backup and restore will succeed, but some transactions which were performed during the backup may not be replicated. It is recommended to create a mirror server and perform backups on the mirror, or else to be willing to restart replication using hdl-dumpfromprimary after restoring from backup, or else be able to manually re-execute any administrative operations that occurred during the backup.

14 Handle HTTP JSON REST API

The Handle server HTTP interface has in the past responded to requests in the native binary Handle protocol of RFC 3652. As of version 8 requests can also be made using a REST API, with requests and responses expressed via JSON encoding. This API may be useful for server administration or for developing applications using Handle, especially those not developed in Java and thus unable to use the Client SDK.

By default the Handle server HTTP interface will respond to requests using HTTP or HTTPS, on the same port. This is called "port unification" and was preferred in order to allow legacy servers to upgrade to v8 and receive the full benefits of the REST API without requiring a new port to be made available. Administrative operations using the REST API require the use of HTTPS.

Handle servers generally use self-signed certificates which can not be validated using the typical browser certificate validation. See "Authenticating the Server" in this chapter for one way to authenticate that the correct server has been reached.

14.1 Resources

The primary resource in this api is the handle record for a handle:

/api/handles/{handle}
The handle record for handle {handle}.

/api/handles/{handle}?index={index}
URI query parameters can be used to specify that the resource in question is restricted to one or more of the handle values from the handle record. This is detailed in the method documentation.

Another resource is the collection of handles:

/api/handles?prefix={prefix}
The list of handles available at this server under prefix {prefix}.

Another resource is the collection of homed prefixes:

/api/prefixes
The list of prefixes homed at this server.

14.2 Requests

All request entities are JSON.

Generally supported URI query parameters:

- **callback={callback}**
  Allows the use of JSONP. The response entity will be wrapped in {callback}(...).
- **pretty=[true|false]**
  If true responses are pretty-printed. Default false.

For any boolean query parameter, the parameter without any value (e.g. ?pretty) is considered to have value true.

14.3 Cross-Origin Resource Sharing

All resources support Cross-Origin Resource Sharing (CORS). Note that
Access-Control-Allow-Credentials: is *not* set true, to prevent CSRF attacks. Cross-origin applications *can* send the Authorization: Handle header which is constructed by the application rather than the browser.

14.4 Responses

All response entities are JSON.

Response statuses:

- 200 OK
  The operation succeeded.
- 201 Created
  The operation succeeded and resulted in a new handle or new handle values being created.
- 400 Bad Request
  Returned for a incorrectly formatted or otherwise invalid request.
  Also returned for a request for a handle for which a server is not responsible.
- 401 Unauthorized
  The operation requires an authorized caller, but the call in not authenticated.
- 403 Forbidden
  The caller is authenticated but not authorized to perform the operation.
- 404 Not Found
  The handle in question does not exist.
- 409 Conflict
  A request to PUT a handle or handle values specified not to overwrite, but the handle or handle values already exist.
- 500 Internal Server Error
  Something unexpected has gone wrong on the server.

Many response entities include a "responseCode" property which is the Handle protocol response code. Some common response codes and the corresponding HTTP status codes are:

- 1: Success (200 OK or 201 Created)
- 2: An unexpected error on the server (500 Internal Server Error)
- 100: Handle not found (404 Not Found)
- 101: Handle already exists (409 Conflict)
- 102: Invalid handle (400 Bad Request)
- 200: Values not found (in resolution, 200 OK; otherwise 400 Bad Request)
- 201: Value already exists (409 Conflict)
- 202: Invalid value (400 Bad Request)
- 301: Server not responsible for handle (400 Bad Request)
- 402: Authentication needed (401 Unauthorized)
- 40x: Other authentication errors (403 Forbidden)

The bulk of request and response entities are JSON representations of handle values. The syntax for this is described at the end of this document.

14.5 Methods

14.5.1 GET /api/handles/{handle}

Resolves the handle record for handle {handle}.

URI query parameters:

- **index={index}**
  Specifies that only the handle value with index {index} should be resolved. The query parameter can be repeated to indicate a collection of handle values.
- **type={type}**
  Specifies that only the handle values with type {type} should be resolved. If {type} ends with a period all period-delimited subtypes are included. The query parameter can be repeated. Multiple index and type parameters indicate that all handle values either of a matching index or a matching type should be resolved.
- **auth=[true|false]**
  If true, perform an authoritative resolution, bypassing cache and sending the request to a primary server. Default: false. This flag is ignored in requests sent directly to an end server (instead of a proxy).
- **publicOnly=[true|false]**
  If true, only resolve publicly readable handle values. If false, resolve all values, potentially resulting in a 401 Unauthorized response. Default: true for unauthenticated requests, false for authenticated requests.

Response entity: an object with the following properties:

- "responseCode": Handle protocol response code for the message.
- "handle": The handle specified in the request.
- "message": For error responses, an error message.
- "values": An array of handle values.

14.5.2 PUT /api/handles/{handle}
PUT /api/handles/{handle}?index={index}

Create the handle {handle} or replace its handle record. If query parameters specify specific indices, add or replace those specific handle values in the handle record.

Request entity: an array of handle values, or an object with property values an array of handle values (other properties are ignored). A single value is accepted as well.

URI query parameters:

- **index={index}**
  Specifies that only the handle value with index {index} should be added or replaced. The query parameter can be repeated to indicate a collection of handle values. The indices must match the indices of the handle values in the request entity.
- **index=various**
  A shortcut to indicate that the handle values given in the request entity should be added or replaced.
- **overwrite=[true|false]**
  If true, replace handle records of handles which already exist, or replace handle values which already exist. If false, return 409 Conflict for attempts to PUT an existing handle or existing handle values. Default: true.
- **mintNewSuffix=[true|false]**
  If true, the handle to be created is formed by appending a random server-generated string to the {handle} parameter. Note that the slash should be included in the {handle} parameter. Default: false.

Response entity: an object with the following properties:

- "responseCode": Handle protocol response code for the message.
- "handle": The handle specified in the request.
- "message": For error responses, an error message.

14.5.3 DELETE /api/handles/{handle}
DELETE /api/handles/{handle}?index={index}

Delete the handle {handle}. If query parameters specify specific indices, delete those specific handle values from the handle record.

URI query parameters:

- **index={index}**
  Specifies that only the handle value with index {index} should be deleted. The query parameter can be repeated to indicate a collection of handle values.

Response entity: an object with the following properties:

- "responseCode": Handle protocol response code for the message.
- "handle": The handle specified in the request.
- "message": For error responses, an error message.

14.5.4 GET /api/handles?prefix={prefix}

List handles under prefix {prefix}.

URI query parameters:

- **prefix={prefix}**
  Required. Specifies the prefix of the handles to be listed.
- **page={page}**
  pageSize={pageSize}
  Specify paginated listing. The page number {page} is zero-based. If the page size {pageSize} is zero a count of handles is returned but no handles. If either parameter is missing or negative all handles are returned.

Response entity: an object with the following properties:

- "responseCode": Handle protocol response code for the message.
- "prefix": The prefix specified in the request.
- "message": For error responses, an error message.
- "totalCount": The total number of handles under the given prefix.
- "handles": An array of strings.

14.6 Authentication

All authenticated transactions *must* use HTTPS. Over HTTP authentication is ignored and requests which require authorization return 403 Forbidden.

14.6.1 Handle-Based Certificates

Requests over HTTPS involve certificates, but generally self-signed certificates which wrap public keys stored in the handle records. A handle server's SSL certificate, for instance, will generally be a self-signed certificate where the public key is the public key from the handle server's siteinfo (HS_SITE handle value). Client applications should authenticate the servers they contact, by comparing the public key in the server certificate with the public key in the previously obtained HS_SITE information for the server.

Since browser-based JavaScript does not have access to the server certificate, there is another way to authenticate the server using the Authorization: Handle mechanism described below.

14.6.2 Client-Side Certificates

If a client-side certificate is used, it will provide authentication for any request where an Authorization: header is not otherwise sent. This corresponds to Handle protocol HS_PUBKEY authentication. The handle identity of the caller is given as the string {index}:{handle} and must be in either a UID attribute or, if there is no UID attribute, a CN attribute in the distinguished name of the certificate subject. The public key of the certificate must correspond to the HS_PUBKEY value stored at that handle and index. A self-signed certificate is acceptable; the issuer of the certificate is ignored.

In general, handle servers do not request client-side certificates. This is to prevent unpleasant behavior in browser-based applications. A server can be asked to request a client-side certificate by sending the header

Authorization: Handle clientCert="true"

To ask the handle server to renegotiate an existing HTTPS connection, such as to change which client-side certificate is used, send the header

Authorization: Handle clientCert="true", renegotiate="true"

14.6.3 Basic Access Authentication

Any individual request may be authenticated by sending an Authorization: Basic header. This corresponds to Handle protocol HS_SECKEY authentication.

The Authorization: Basic header, as usual, takes as parameter the Base64-encoding of {username}:{password}. Here, the {username} is the usual handle identity {index}:{handle}, however, to deal with the colon, it must be *percent-encoded.* A minimal percent encoding is to replace every percent-sign in {handle} with %25, and then the colon between {index} and {handle} and every colon in {handle} with %3A. Characters in {handle} outside ASCII may be either percent-encoded or UTF-8-encoded before the Base64 encoding. The secret key {password} corresponds to the bytes in the HS_SECKEY value at the given index and handle; if it is considered as text it will need to be UTF-8-encoded before the Base64 encoding.

14.6.4 Authentication via Authorization: Handle

A challenge-response framework based on Handle protocol authentication is available. It is the only option for HS_PUBKEY authentication in environments which can not easily specify client-side certificates (as in cross-origin browser-based JavaScript).

This framework uses Authorization: Handle and WWW-Authenticate: Handle headers which contain

parameters of the form key="value", key="value". Note the HTTP headers are restricted to ASCII; the documentation below clarifies how various parameters are encoded to avoid that being a problem.

14.6.5 Challenge from Server to Client

Any unauthenticated request will come back with a challenge as follows:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Handle sessionId="1j06a95ekxkyyit9wj5gxro94",
                        nonce="5xvRCCVh8wwu1iOvsaFzIQ=="
```

The sessionId is an opaque string which is used as a session secret, like session cookies or OAuth bearer tokens. Once a challenge-response communication is completed, the sessionId can continue to be used to make authenticated requests. It must be kept secret.

The nonce is 16 bytes in Base64 encoding.

14.6.6 Challenge-Response Request from Client to Server

The client can now repeat the request, adding an Authorization: Handle header to authenticate.

```
PUT /api/...      (or DELETE or POST or GET)
Authorization: Handle sessionId="1j06a95ekxkyyit9wj5gxro94",
                    id="300:200/23",
                    type="HS_PUBKEY",
                    cnonce="jizZxJAR/hCGlovkvJXtjsCTNj8=",
                    alg="SHA1",
                        signature="CYKX3+PbhMfsmvOqKQcB7c2RTeGPXOamjRX5987x8tUIy3ilqtyCSh6r
                        2GIuzfKsomwPyORZ/BHIrGwM2pBJjiXxOYJ7jn4MjvYgpOtXlh9BRfRRqkTXcM2ECr
                        x1z0jPBwO+R/E6Uzxyw2Eu0G4RsaiAYQV1t/6SbXOVS+JNcl3yqZBC6/jThDPMBGx
                        2ZvfWiPzdJZSbGQE9HpZJyb4T3VEnJiJCy9hqah1qluzBpSZFfsqTOsddYV9t0Y7bFUL
                        W5PtmTsyQ5pLIrfBFfQrj2U6EmUaF+CpzcuFHEkgPg/fRK3IPHKiNN4/POy98NhhO
                        UjBpwC3wrBzg25HkdKrFDA=="
```

The parameters are:

- sessionId: Must match the sessionId sent in the challenge.
- id: The handle identity of the caller, as {index}:{handle}. Non ASCII-characters must be percent-encoded (as UTF-8).
- type: Either HS_PUBKEY or HS_SECKEY.
- cnonce: 16 bytes, randomly chosen by the client, in Base64 encoding.
- alg: The digest algorithm used to create the client signature. Generally "SHA1" or "SHA256". See below.
- signature: The bytes of the client signature in Base64 encoding, see below.

The client will sign the concatenation of the server's "nonce" (sent in the challenge) and the client's

"cnonce" (sent here in the challenge response). The alg and signature fields correspond to the "Challenge-Response" defined in RFC3652 section 3.5.2, with the concatenation of "nonce" and "cnonce" taking the place of the Handle protocol server challenge.

For HS_PUBKEY, the concatenation of "nonce" and "cnonce" is digested using the digest algorithm specified in "alg", and "signature" is the Base64-encoded bytes of the signature of the digest using the client's private key. If the signing key is an RSA key, then the signature bytes form a RSASSA-PKCS1-v1_5 signature. If the public key is a DSA key, then the signature bytes are the DER-encoded ASN.1 SEQUENCE of two ASN.1 INTEGER values corresponding to the r and s values of the DSA signature.

For HS_SECKEY, the "signature" is actually a MAC. The "alg" specifies how the secret key is digested together with the "nonce" and "cnonce" into the MAC. In the case of alg="SHA1", the "signature" will be the Base64-encoded bytes of SHA-1({secret-key} + {nonce} + {cnonce} + {secret-key}), which is the traditional HS_SECKEY approach. The recommended algorithm at present is alg="PBKDF2-HMAC-SHA1", in which case additional parameters are required in the header: salt (Base64-encoded bytes), iterations (an integer), and length (an integer, usually 160). The secret key is used to create a derived key using PBKDF2 with the salt, iterations, and length parameters, which is then used with HMAC-SHA1 to create a MAC over the nonce and cnonce.

14.6.7 Further Requests in Session

Once the client has authenticated, the client can continue to use the sessionId to perform authenticated requests:

```
PUT /api/...
Authorization: Handle sessionId="1j06a95ekxkyyit9wj5gxro94"
```

14.6.8 Authenticating the Server

All authenticated requests must happen over HTTPS, so in most environments the client can already verify the server's public key. However, in browser-based JavaScript there is no access to the server certificate. To compensate, the client is allowed to send a "cnonce" value before authenticating. The server will respond as usual with "sessionId" and "nonce" but will also add "serverAlg" and "serverSignature" values comprising a signature over the concatenation of "nonce" and "cnonce".

```
PUT /api/...
Authorization: Handle cnonce="jizZxJAR/hCGlovkvJXtjsCTNj8="

...

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Handle sessionId="1j06a95ekxkyyit9wj5gxro94",
                          nonce="5xvRCCVh8wwu1iOvsaFzIQ==",
                          serverAlg="SHA1",
```

serverSignature="MCwCFHtAibtYzxe4Ne/0j5FfMmeayeYrAhRcmcUFhTU
EKJVbMzqN6H7IuQmCpA=="

The "serverSignature" does not affect the remainder of the communication. It can be used by the client to ensure that it is talking to the intended server.

14.6.9 Deleting a Session

The client may terminate a session using DELETE /api/sessions/this as described below.

14.7 Sessions API

Clients wishing to proactively initiate and authenticate in a session may prefer to use the sessions API to interact with the sessions directly. The parameters communicated are exactly the same as those sent via WWW-Authenticate: and Authorization: headers in the challenge/response framework, but the endpoints used refer to the sessions themselves rather than to operations to be performed, and the information can generally be exchanged in JSON entities rather than in headers.

There are two resource endpoints:

/api/sessions
The collection of sessions; POST to this endpoint to create a session.

/api/sessions/this
A particular session. The path component *this* is used instead of the secret sessionId in deference to the security advice that secrets should not be included in URIs even in HTTPS. The particular session in question is indicated either in the Authorization: Handle header or in the request entity.

14.7.1 POST /api/sessions

Initiate a session.

The client may optionally send a "cnonce" value, either in an Authorization: Handle header or in a JSON object in the request entity. This indicates a request for a server signature.

The response entity will be a JSON object with properties "sessionId" and "nonce", and also "serverAlg" and "serverSignature" if requested.

An Authorization: Basic header may be included to authenticate in the session using an HS_SECKEY via Basic auth (see 14.6.3).

14.7.2 GET /api/sessions/this

Verify a session.

Since GET requests do not allow request entities, the client must specify the session using a header Authorization: Handle sessionId="...".

The response entity will be a JSON object with properties "sessionId" and "nonce", and also "serverAlg" and "serverSignature" if a "cnonce" was sent with Authorization: Handle header, and finally "authenticated":true and the authenticated "id" if there has been a successful authentication in this session.

14.7.3 PUT /api/sessions/this

Authenticate in a session.

The client must send, in either an Authorization: Handle header or a JSON object in the request entity, the fields "sessionId", "id", "type", "cnonce", "alg", and "signature". Note that although "id" may need to be percent-encoded in the header, it must not be percent-encoded in a JSON entity.

The response entity will be a JSON object with properties "sessionId" and "nonce", and "authenticated":true and the authenticated "id" if the authentication was successful.

If "sessionId" alone is sent in the request entity, an Authorization: Basic header may be included to authenticate in the session using an HS_SECKEY via Basic auth (see 14.6.3).

14.7.4 DELETE /api/sessions/this

Delete a session.

Since request entities may not be supported for DELETE requests, the client should send the sessionId in an Authorization: Handle header.

A successful response is empty, with HTTP status 204 No Content. After this request is processed the session may no longer be used.

14.8 JSON Representation of Handle Values

Each value is a JSON object with generally 5 attributes:

- "index" : an integer
- "type" : a string
- "data" : an object, see below
- "ttl" : the time-to-live in seconds of the value, an integer (or, in the rare case of an absolute expiration time, that expiration time as an ISO8601-formatted string)
- "timestamp" : an ISO8601-formatted string

Plus two attributes which are omitted in the common case:

- "permissions" : a string representing the bitmask of permissions. Generally this is "1110" (admin read, admin write, public read, not public write) in which case it is omitted. Values of type "HS_SECKEY" generally use "permissions":"1100".
- "references": an array of objects, each of which has attributes "index", an integer, and "handle", a string. Omitted when empty which is essentially always the case. (Perhaps references should be considered deprecated or reserved.)

Handle value data is binary data as a byte array. For ease of use, the JSON representation of handle values allows for a separate structured format of certain typical binary formats. In all cases the underlying data is simply a byte array.

Handle value data is either a string or an object with properties "format", a string, and "value".

- If "format"="string", "value" is a string, representing the data as a UTF-8 string.
- If "format"="base64", "value" is a string, with a Base64 encoding of the data.
- If "format"="hex", "value" is a string, with a hex encoding of the data.
- If "format"="admin", "value" is an object, representing an HS_ADMIN value, with properties "handle" (a string), "index" (an integer), and "permissions" (a string, representing the bitmask of permissions).
- If "format"="vlist", "value" is an list of objects, representing an HS_VLIST value; each object in the list has properties "handle" (a string) and "index" (an integer).
- If "format"="site", "value" is an object, representing an HS_SITE value. As the structure of this object is complicated and generally of limited technical interest it is currently omitted from this documentation.
- If "format"="key", "value" is an object in Json Web Key format, representing a public key.