hashdb

USERS MANUAL

Quickstart Guide Included
October 31, 2014

Authored by:
Bruce D. Allen
Jessica R. Bradley
Simson L. Garfinkel

One Page Quickstart for Windows Users

This page provides a very brief introduction to downloading, installing and running *hashdb* on Windows systems.

1. Download the windows installer for the latest version of *hashdb*. It can be obtained from `http://digitalcorpora.org/downloads/hashdb`. The file is named `hashdb-x.y.z-windowsinstaller.exe` where x.y.z is the latest version.

2. Run the installer file. This will automatically install *hashdb* on your machine.

3. Navigate to the directory where you would like to create a hash database. Then, to run *hashdb* from the command line, type the following instructions:

   - `hashdb create sample.hdb`

   In the above instructions, `sample.hdb` is the empty database that will be created with default database settings.

4. Next, import data into the database. You will need a DFXML file containing sector hash values. If you do not already have one, see **Subsection 2.2** for instructions on creating one. To populate the hash database with the hashes from the DFXML file called `sample.xml`, type the following instructions from the directory where you created the database:

   - `hashdb import sample.xml sample.hdb`

   This command, if executed successfully, will print the number of hash values inserted. For example:

   ```
   hashdb changes (insert):
       hashes inserted: 2595
   ```

5. Additionally, the file `log.xml` contained in the directory *sample.hdb* will be updated with change statistics. It will show the number of hash values that have been inserted [see **Subsection 4.5** for more information on the change statistics tracked in the log file].

One Page Quickstart for Linux and Mac Users

This page provides a very brief introduction to downloading, installing and running *hashdb* (creating a database and populating it) on Linux and MacOS systems.

1. Download the latest version of *hashdb*. It can be obtained from `http://digitalcorpora.org/downloads/hashdb`. The file is called `hashdb-x.y.z.tar.gz` where x.y.z is the latest version.

2. Un-tar and un-zip the file. In the newly created *hashdb-x.y.z* directory, run the following commands:

   - `./configure`
   - `make`
   - `sudo make install`

   Note, users will likely need to first download and install dependent library files. Instructions are outlined in the referenced section. [Refer to **Section 3**].

3. Navigate to the directory where you would like to create a hash database. Then, to run *hashdb* from the command line, type the following instructions:

   - `hashdb create sample.hdb`

   In the above instructions, `sample.hdb` is the empty database that will be created with default database settings.

4. Next, import data into the database. You will need a DFXML file containing block hash values. If you do not already have one, see **Subsection 2.2** for instructions on creating one. To populate the hash database with the hashes from the DFXML file called `sample.xml`, type the following instructions from the directory where you created the database:

   - `hashdb import sample.xml sample.hdb`

   This command, if executed successfully, will print the number of hash values inserted. For example:

   ```
   hashdb changes (insert):
       hashes inserted: 2595
   ```

5. Additionally, the file `log.xml` contained in the directory *sample.hdb* will be updated with change statistics. It will show the number of hash values that have been inserted [see **Subsection 4.5** for more information on the change statistics tracked in the log file].

Contents

1 Introduction **1**
- 1.1 Overview of *hashdb* . . . 1
- 1.2 Purpose of this Manual . . . 2
- 1.3 Conventions Used in this Manual . . . 2

2 How *hashdb* Works **2**
- 2.1 Hash Blocks . . . 3
- 2.2 DFXML . . . 4
  - 2.2.1 Creating a DFXML file using **md5deep** . . . 4
  - 2.2.2 Creating a DFXML file using **fiwalk** . . . 5
  - 2.2.3 Creating a DFXML file using **hashdb** . . . 6
- 2.3 Contents of a Hash Database . . . 6
- 2.4 Using the Hash Databases . . . 7
- 2.5 **bulk_extractor** . . . 7
  - 2.5.1 Forensic Path . . . 7

3 Installation Guide **8**
- 3.1 Installing on Linux or Mac . . . 8
- 3.2 Installing on Windows . . . 9
- 3.3 Installing Other Related Tools . . . 10

4 Running *hashdb* **11**
- 4.1 General Usage . . . 11
- 4.2 Creating a Hash Database . . . 12
- 4.3 Importing and Exporting using DFXML . . . 14
- 4.4 Database Manipulation . . . 15
  - 4.4.1 Tracking Changes in Hash Databases . . . 15
- 4.5 Scan Services . . . 15
- 4.6 Statistics . . . 21
- 4.7 Tuning . . . 22
- 4.8 Performance Analysis . . . 22
- 4.9 Importing and Scanning Using the **bulk_extractor** *hashdb* Scanner . . . 24

5 Use Cases for *hashdb* **25**
- 5.1 Querying for Source or Database Information . . . 25
  - 5.1.1 Querying a Remote Hash Database . . . 26
- 5.2 Writing Software that works with *hashdb* . . . 27
- 5.3 Scanning or Importing to a Database Using **bulk_extractor** . . . 27
- 5.4 Updating Hash Databases . . . 27
  - 5.4.1 Update Commands and "Duplicate" Hashes . . . 28
- 5.5 Optimizing a Hash Database . . . 28
- 5.6 Exporting Hash Databases . . . 29

6 Worked Example: Finding Similarity Between Disk Images **29**

7 Troubleshooting **32**

8 Related Reading **32**

Appendices **33**

A *hashdb* Quick Reference **33**

B Output of *hashdb* Help Command **34**

C *hashdb* API: `hashdb.hpp` **41**

D bulk_extractor *hashdb* Scanner Usage Options **44**

1 Introduction

1.1 Overview of *hashdb*

hashdb is a tool that can be used to find data in raw media using cryptographic hashes calculated from blocks of data. It is a useful forensic investigation tool for tasks such as malware detection, child exploitation detection or corporate espionage investigations. The tool provides several capabilities that include:

- Creating hash databases of MD5 block hashes, as opposed to file hashes.
- Importing hash values from Digital Forensic XML (DFXML) files created by other programs such as **md5deep** or **fiwalk**.
- Scanning the hash database for matching hash values using either the local or remote system.
- Providing the source information for hash values.

Using *hashdb*, a forensic investigator can take a known set of blacklisted media and generate a hash database. The investigator can then use the hash database to search against raw media for blacklisted information. For example, given a known set of malware, an investigator can generate a sector hash database representing that malware. The investigator can then search a given corpus for fragments of that malware and identify the specific malware content in the corpus using *hashdb* and the **bulk_extractor** program.

hashdb relies on block hashing rather than full file hashing. Block hashing provides an alternative methodology to file hashing with a different capability set. With file hashing, the file must be complete to generate a file hash, although a file carver can be used to pull together a file and generate a valid hash. File hashing also requires the ability to extract files, which requires being able to understand the file system used on a particular storage device. Block hashing, as an alternative, does not need a file system or files. Artifacts are identified at the block scale (usually 4096 bytes) rather than at the file scale. While block hashing does not rely on the file system, artifacts do need to be sector-aligned for *hashdb* to find hashes [3].

hashdb provides an advantage when working with hard disks and operating systems that fragment data into discontiguous blocks yet still sector-align media. This is because scans are performed along sector boundaries. Because *hashdb* works at the block resolution, it can find part of a file when the rest of the file is missing, such as with a large video file where only part of the video is on disk. *hashdb* can also be used to analyze network traffic (such as that captured by **tcpflow**). Finally, *hashdb* can identify artifacts that are sub-file, such as embedded content in a `.pdf` document.

hashdb stores cryptographic hashes (along with their source information) that have been calculated from hash blocks. It also provides the capability to scan other media for hash matches. Many of the capabilities of *hashdb* are best utilized in connection with the **bulk_extractor** program. This manual describes uses cases for the *hashdb* tools, including its uses with **bulk_extractor** and demonstrates how users can take full advantage of all of its capabilities.

1.2 Purpose of this Manual

This Users Manual is intended to be useful to new, intermediate and experienced users of *hashdb*. It provides an in-depth review of the functionality included in *hashdb* and shows how to access and utilize features through command line operation of the tool. This manual includes working examples with links to the input data used, giving users the opportunity to work through the examples and utilize all aspects of the system.

1.3 Conventions Used in this Manual

This manual uses standard formatting conventions to highlight file names, directory names and example commands. The conventions for those specific types are described in this section.

Names of programs including the post-processing tools native to *hashdb* and third-party tools are shown in **bold**, as in **bulk_extractor**.

File names are displayed in a fixed width font. They will appear as `filename.txt` within the text throughout the manual.

Directory names are displayed in italics. They appear as *directoryname/* within the text. The only exception is for directory names that are part of an example command. Directory names referenced in example commands appear in the example command format.

Database names are denoted with bold, italicized text. They are always specified in lower-case, because that is how they are referred in the options and usage information for *hashdb*. Names will appear as ***databasename***.

This manual contains example commands that should be typed in by the user. A command entered at the terminal is shown like this:

■ `command`

The first character on the line is the terminal prompt, and should not be typed. The black square is used as the standard prompt in this manual, although the prompt shown on a users screen will vary according to the system they are using.

2 How *hashdb* Works

The *hashdb* tool provides capabilities to create, edit, access and search databases of cryptographic hashes created from hash blocks. The cryptographic hashes are imported into a database from DFXML files created by other programs (which could include **md5deep**) or exported from another *hashdb* database. *hashdb* databases can also be populated using **bulk_extractor** and the *hashdb* scanner. Once a databases is created, *hashdb* provides users with the capability to scan the database for matching hash values and identify matching content. Hash databases can also be exported, added to, subtracted from and shared.

Figure 1 provides an overview of the capabilities included with the *hashdb* tool. *hashdb* populates databases from DFXML files created by other programs. The sources of those

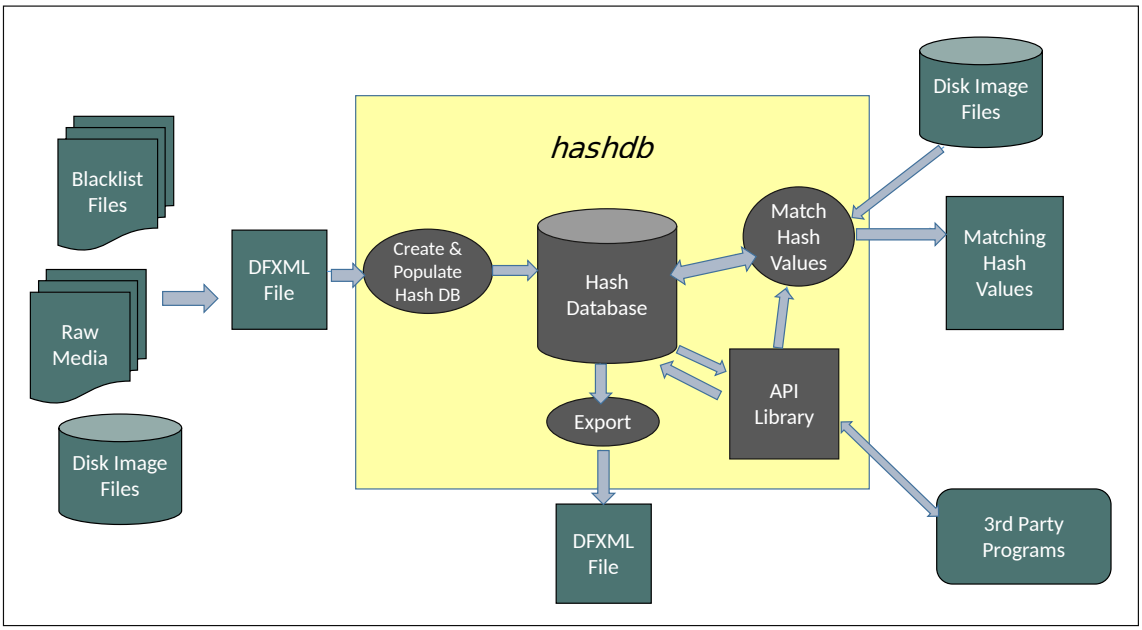

Figure 1: Overview of the *hashdb* system

```
|-512-|-512-|-512-|-512-|-512-|-512-|-512-|-512-|-512-|-512-| ...
|----------------------4K----------------------|
      |----------------------4K----------------------|
            |----------------------4K----------------------|
                                          etc.
```

Figure 2: Hashes generated over overlapping sector boundaries. 4K lines represent the hash blocks.

files can be virtually any type of raw digital media including black list files and disk images. Users can also add or remove data from the database after it is created. Once the database is populated, *hashdb* can export content from the database in DFXML format. It also provides an API that can be used by third party tools (as it is used in the **bulk_extractor** program) to create, populate and access hash databases. Finally, *hashdb* allows users to scan the hash database for matching hash values.

2.1 Hash Blocks

hashdb relies on block hashing rather than file hashing. A hash block is a contiguous sequence of bytes, typically 4KiB in size. Tools using block hashing calculate cryptographic hashes from hash blocks, along with information about where the hash blocks are sourced from. To increase the probability of finding matching hashes in sector-based disk images, hashes are generated at each sector boundary. Figure 2 illustrates cryptographic hashes generated from 4KiB hash blocks aligned on 512 byte sector boundaries. Block size is selectable in tools such as **md5deep**. In our work, we use a block size of 4KiB.

Listing 1: Excerpt of a DFXML report file showing the MD5 output

```
  <fileobject>
    <filename>/home/bdallen/demo/mock_video.mp4</filename>
    <filesize>10630146</filesize>
    <ctime>2014-01-30T20:20:39Z</ctime>
    <mtime>2014-01-30T19:04:59Z</mtime>
    <atime>2014-01-30T20:04:52Z</atime>
    <byte_run file_offset='0' len='4096'>
      <hashdigest type='MD5'>63641a3c008a3d26a192c778dd088868</hashdigest>
    </byte_run>
    <byte_run file_offset='4096' len='4096'>
      <hashdigest type='MD5'>c7dd2354e223c10856469e27686b8c6b</hashdigest>
    </byte_run>
    <byte_run file_offset='8192' len='4096'>
      <hashdigest type='MD5'>ff540fda05d008ccebf2cca2ec71571d</hashdigest>
    </byte_run>
    <byte_run file_offset='12288' len='4096'>
      <hashdigest type='MD5'>d3de47d704e85e0f61a91876236593d3</hashdigest>

...

    <byte_run file_offset='10625024' len='4096'>
      <hashdigest type='MD5'>d2d958b44c481cc41b0121b3b4afae85</hashdigest>
    </byte_run>
    <byte_run file_offset='10629120' len='1026'>
      <hashdigest type='MD5'>4640564a8655d3b201a85b4a76411b00</hashdigest>
    </byte_run>
    <hashdigest type='MD5'>a003483521c181d26e66dc09740e939d</hashdigest>
  </fileobject>
```

2.2 DFXML

hashdb can be used to populate hash databases by importing block hashes from DFXML files. DFXML is an XML language designed to represent a wide range of forensic information and forensic processing results. It allows the sharing of structured information between independent tools and organizations [2].

Note that *hashdb* does not require DFXML files to import hashes. The **bulk_extractor** *hashdb* scanner can import hashes directly into a new hash database, see **subsection 4.9** for importing using the **bulk_extractor** *hashdb* scanner. Also, third party tools can be created for importing hashes directly into a hash database by interfacing with the *hashdb* library API, see **autorefusingSection**.

2.2.1 Creating a DFXML file using md5deep

The **md5deep** tool creates cryptographic hashes from hash blocks and produces DFXML files. Listing 1 shows an excerpt of the DFXML file created by **md5deep**. The portion of the file of interest to *hashdb* is contained in the "byte_run" tag. The "file_offset" attribute is the number of bytes into the file where the cryptographic block hash was calculated. The "len" attribute indicates the size of the block. The "hashdigest" tag identifies that hash algorithm (MD5) and the long hexadecimal hash value. The "filename" tag indicates the filename to which the hashes can be attributed.

Users may create DFXML files to import hashes from by using the **md5deep** tool. **md5deep** is available at `http://md5deep.sourceforge.net`. For additional instructions on downloading and installing **md5deep**, go to `http://github.com/simsong/hashdb/wiki/Installing-md5deep`.

Choose a file or directory to use as the source of data for the hash file output. For this manual, we use the file `mock_video.mp4` available at `http://digitalcorpora.org/downloads/hashdb/demo/`. Then, run **md5deep** with the following command:

■ `md5deep -p 4096 -d mock_video.mp4 > mock_video.xml`

The above command specifies:

- a block size of 4096 bytes (**-p** option)
- that the hash output will be written to a DFXML file (**-d** option)
- to write the output to the file `mock_video.xml`. The > symbol specification writes the output into the file

The file `mock_video.xml` will be used in the next step to create the hash database. However, any DFXML file containing block hash values can be used in *hashdb*.

Note, for this example we are using only one file to populate the DFXML. However, users will typically be creating a block hash file from thousands of files in hundred of directories. To create a block hash file that recursively includes all files and directories contained within a directory, use the command **mdf5deep -r <directoryname>** along with the other options specified above.

2.2.2 Creating a DFXML file using fiwalk

The **fiwalk** tool can create block hashes of files in filesystems in an image, see `http://www.forensicswiki.org/wiki/Fiwalk`. **fiwalk** is part of The Sleuth Kit® (TSK), available from `https://github.com/sleuthkit/sleuthkit`.

For example run **fiwalk** with the following command:

■ `fiwalk -x -S 4096 my_image.E01 > my_image.xml`

The above command specifies:

- Send output to stdout **-x** option.
- Perform sector hashes every 4096 bytes **-s** option.
- Perform sector hashes on the file system in the `my_image.E01` image.
- Direct output to file `my_image.xml`.

2.2.3 Creating a DFXML file using hashdb

The `export` command of the **hashdb** tool writes out the block hashes in a hash database along with their source information.

For example run **hashdb** with the following command:

- `hashdb export mock_video.hdb demoVideoHashes.xml`

The above command specifies to export hashes and their source information from hash databse `mock_video.hdb` to DFXML file `demoVideoHashes.xml`.

2.3 Contents of a Hash Database

Each *hashdb* database is contained in a directory called *<databasename>.hdb* and contains a number of files. These files are:

```
Bloom_filter_1
hash_store
history.xml
log.xml
settings.xml
source_filename_store.dat
source_filename_store.idx1
source_filename_store.idx2
source_lookup_store.dat
source_lookup_store.idx1
source_lookup_store.idx2
source_metadata_store
source_repository_name_store.dat
source_repository_name_store.idx1
source_repository_name_store.idx2
```

These files include XML files containing configuration settings and logs, a Bloom filter file used for improving the speed of hash lookups, binary files containing stored hashes from multiple sources and binary files that allow lookup of hash source names. Of these files, the history, settings, and log files may be of interest to the user:

- `log.xml`
  Every time a command is run that changes the content of the database, this file is replaced with a log of the run. The log includes the command name, information about *hashdb* including the command typed and how *hashdb* was compiled, information about the operating system *hashdb* was just run on, timestamps indicating how much time the command took, and the specific *hashdb* changes applied, described in more detail in **section 4**.

- `history.xml`
  The purpose of this file is to provide full attribution for a database. Every *hashdb* command executed that changes the state of the database is logged into the `log.xml` file and is appended to the `history.xml` file. For *hashdb* commands that involve manipulations from another database (or from two databases, as is the case with the `add_multiple` command), the history file of those databases are

also appended. It can be difficult to follow the `history.xml` file because of its XML format, but it provides full attribution nonetheless.

- `settings.xml`
  This file contains the settings requested by the user when the block hash database was created, see *hashdb* settings and Bloom filter settings options. This file also contains internal *hashdb* configuration and versioning information that is specific to how the *hashdb* tool was compiled.

2.4 Using the Hash Databases

hashdb provides the capability for users to scan the database for matching hash blocks locally or remotely via a socket. Users can also query for hash source information and information about the hash database itself. *hashdb* provides an API to access the import and scan capabilities. The import capability allows third party tools to create a new database at a specified directory, import an array of hashes with source information and write changes to the `log.xml` file. The scan capability provided by the API allows third party tools to open an existing database and perform a scan. Most importantly, the **bulk_extractor** *hashdb* scanner uses the *hashdb* API to provide users with the capability to create databases from disk images or scan digital media and find matching hash blocks within the data **bulk_extractor** is processing. In later sections, this manual describes the methods for using **bulk_extractor** together with the *hashdb* tool.

2.5 bulk_extractor

bulk_extractor is an open source digital forensics tool that extracts features such as email addresses, credit card numbers, URLs and other types of information from digital evidence files. It operates on disk images, files or a directory of files and extracts useful information without parsing the file system or file system structures. For more information on how to use **bulk_extractor** for a wide variety of applications, refer to the separate publication *The **bulk_extractor** Users Manual* available at `http://digitalcorpora.org/downloads/bulk_extractor/BEUsersManual.pdf` [1].

bulk_extractor has multiple scanners that extract features. One particular scanner, the *hashdb* scanner links the full set of **bulk_extractor** capabilities directly to the *hashdb* tool. The *hashdb* scanner uses the *hashdb* API to create and import data into hash databases directly from the data processed by **bulk_extractor**. The scanner also can be run with a hash database as input (again using the *hashdb* API) will scan the data processed by **bulk_extractor** for matching hash values.

2.5.1 Forensic Path

The **bulk_extractor** program introduced the concept of the "forensic path". The forensic path is a description of the origination of a piece of data. It might come from, for example, a flat file, a data stream, or a decompression of some type of data. Consider an HTTP stream that contains a GZIP-compressed email as shown in Figure 3. A series of **bulk_extractor** scanners will first find the ZLIB compressed regions in the HTTP stream that contain the email, decompress them, and then find the features in that email which may include email addresses, names and phone numbers. Using this

Listing 2: Forensic Path of email address features found in **bulk_extractor**

```
11052168704-GZIP-3437   live.com   eMn='domexuser@live.com';var   srf_sDispM
11052168704-GZIP-3475   live.com   pMn='domexuser@live.com';var   srf_sDreCk
11052168704-GZIP-3512   live.com   eCk='domexuser@live.com';var   srf_sFT='<
```

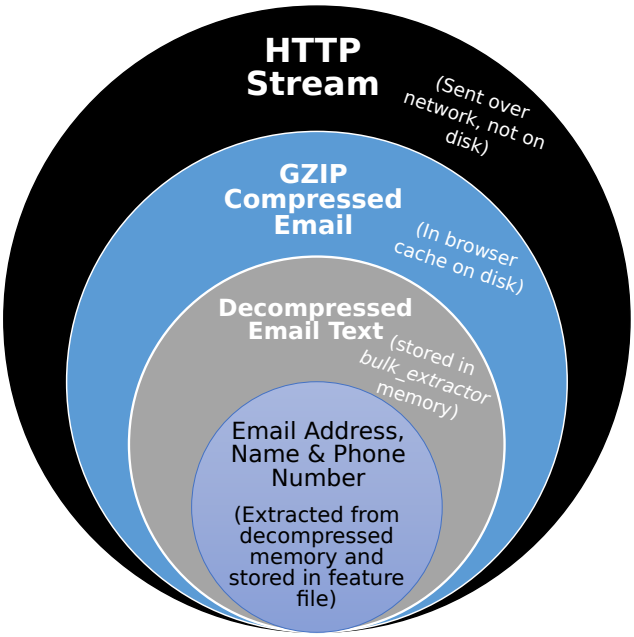

Figure 3: Forensic path of features found in email lead back to HTTP Stream

method, **bulk_extractor** can find email addresses in compressed data. The forensic path for the email addresses found indicate that it originated in an email, that was GZIP compressed and found in an HTTP stream. The forensic path of the email addresses features found might be represented as shown in the example feature file in Listing 2. It is worth nothing that the *hashdb* scanner can recognize a matching block embedded in part of another file. No other existing digital forensic tool can do this; other tools find only completely unembedded files.

3 Installation Guide

hashdb is a command line tool that can be run on Linux, MacOS or Windows systems. Here we describe the installation procedures for those systems. Steps include how to install the required dependencies as well as download *hashdb* and compile the release or run the executable.

3.1 Installing on Linux or Mac

Before compiling *hashdb* for your platform, you may need to install other packages on your system which *hashdb* requires to compile cleanly and with a full set of capabilities.

Dependencies for Linux

The following commands should add the appropriate packages:

- `sudo yum update`
- `sudo yum groupinstall development-tools`
- `sudo yum install gcc-c++`
- `sudo yum install libxml2-devel openssl-devel tre-devel boost-devel`

Dependencies for Mac Systems
Mac users must first install Apple's Xcode development system. Other components should be downloaded using the MacPorts system. If you do not have MacPorts, go to the App store and download and install it. It is free. Once it is installed, try:

- `sudo port install autoconf automake libxml2`

Download and Install *hashdb*
Next, download the latest version of *hashdb*. The software can be downloaded from `http://digitalcorpora.org/downloads/hashdb/`. The file to download is `hashdb-x.y.z.tar.gz` where x.y.z is the latest version. As of publication of this manual, the latest version of *hashdb* is 1.1.1.

After downloading the file, un-tar it by either right-clicking on the file and choosing "extract to...' or typing the following at the command line:

- `tar -xvf hashdb-x.y.z.tar.gz`

Then, in the newly created *hashdb-x.y.z* directory, run the following commands to install *hashdb* in */usr/local/bin* (by default):

- `./configure`
- `make`
- `sudo make install`

hashdb is now installed on your system and can be run from the command line.

Note: sudo is not required. If you do not wish to use sudo, build and install *hashdb* and **bulk_extractor** in your own space at "$HOME/local" using the following commands:

- `./configure --prefix=$HOME/local/ --exec-prefix=$HOME/local CPPFLAGS=-I$HOME/local/include/ LDFLAGS=-L$HOME/local/lib/`
- `make`
- `make install`

3.2 Installing on Windows

Windows users should download the Windows Installer for *hashdb*. The file to download is located at `http://digitalcorpora.org/downloads/hashdb` and is called `hashdb-x.y.z-windowsinstaller.exe` where x.y.z is the latest version number (1.1.1 as of publication of this manual).

You should close all Command windows before running the installation executable. Windows will not be able to find the *hashdb* tools in a Command window if any are open during the installation process. If you do not do this before installation, simply close all Command windows after installation. When you re-open, Windows should be able to find *hashdb*.

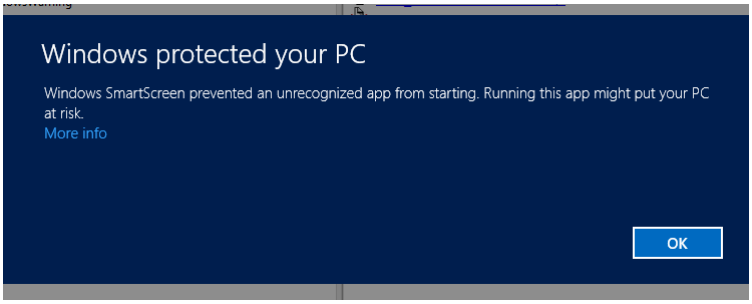

Figure 4: Windows 8 warning when trying to run the installer. Select "More Info" and then "Run Anyway."

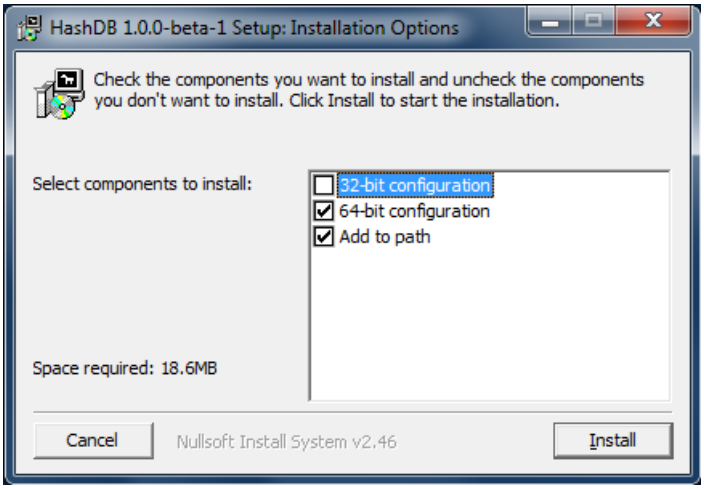

Figure 5: Dialog appears when the user executes the Windows Installer. Select the default configuration.

Next run the `hashdb-x.y.z-windowsinstaller.exe` file. This will automatically install *hashdb* on your machine. Some Windows safeguards may try to prevent you from running it. Figure 4 shows the message Windows 8 displays when trying to run the installer. To run anyway, click on "More info" and then select "Run Anyway."

When the installer file is executed, the installation will begin and show a dialog like the one shown in Figure 5. Users should select the default configuration, which will be the 64-bit configuration for 64-bit Windows systems, or the 32-bit configuration for 32-bit Windows systems. Click on 'Install' and the installer will install *hashdb* on your system and then notify you when it is complete. *hashdb* is now installed on your system can be run from the command line.

3.3 Installing Other Related Tools

Download and Install bulk_extractor

The **bulk_extractor** *hashdb* scanner provides the capability to import block hashes into a new hash database and to scan for hashes against an existing hash database. This scanner is included in **bulk_extractor** version 1.4.5 or later. For detailed instructions on downloading and installing **bulk_extractor**, please refer to the Users Manual

found at `http://digitalcorpora.org/downloads/bulk_extractor/BEUsersManual.pdf`. Note: *hashdb* must be installed first for **bulk_extractor** to build properly with *hashdb*. **bulk_extractor** will automatically install the *hashdb* scanner but only if the *hashdb* library has been installed. Otherwise, **bulk_extractor** will build without the *hashdb* scanner. To check that the *hashdb* scanner is enabled, observe that is enabled in the output of running `./configure` or type bulk_extractor -h and look for *hashdb* setting options.

Download and Install md5deep
md5deep is available at `https://github.com/jessek/hashdeep/releases/tag/release-4.4`. Additional platform-specific installation structions are provided at `https://github.com/simsong/hashdb/wiki/Installing-md5deep`.

Download and Install fiwalk
Please see `http://www.forensicswiki.org/wiki/Fiwalk`. **fiwalk** is part of The Sleuth Kit® (TSK), available from `https://github.com/sleuthkit/sleuthkit`.

4 Running *hashdb*

The core capabilities provided by *hashdb* involve creating and maintaining a database of hash values and scanning media for those hash values. To perform those tasks, *hashdb* users need to start by building a database (if an existing database is not available for use). Users then import hashes using a DFXML file or by using the **bulk_extractor** *hashdb* scaner, and then possibly merge or subtract hashes to obtain the desired set of hashes to scan against. Users then scan for hashes that match. Additional commands are provided to support statistical analysis, performance tuning and performance analysis.

This section describes *hashdb* commands, along with examples, for performing these tasks. For more examples of command usage, please see **section 5**. For a *hashdb* quick reference summary, please see **Appendix A** or `http://digitalcorpora.org/downloads/hashdb/hashdb_quick_reference.pdf`.

4.1 General Usage

Some options apply generically to multiple commands, see Table 1.

Option `-q` suppresses progress output for commands that print progress.

Option `-f` allows control of flags that impact run-time performance when opening the B-Trees that store the hashes inside hash databases. Mutlple flags may be provided by separating them with colon character ":". Flags include:

- `preload` - read the existing file to preload O/S file cache.
- `cache_branches` - enable permanent cache of all branch nodes touched. Otherwise make branch node available when use count becomes 0, just like leaf nodes.
- `least_memory`

Table 1: Options for General Usage

Option	Verbose Option	Specification
`-q`	`--quiet`	Requests quiet mode for commands that print progress status
`-f`	`--flags`	Flags controlling mode: `preload` `cache_branches` `least_memory` `low_memory balanced` `fast fastest`

- `low_memory`
- `balanced`
- `fast`
- `fastest`

4.2 Creating a Hash Database

A hash database must be created before hashes can be added to it. The command to create a hash database is shown in Table 2.

Table 2: Command for Creating Hash Databases

Command	Usage	Description
create	`create [-p <hash block size>] [-m<maximum duplicates>] [--bloom <state>] [--bloom_n <n>] [--bloom_kM <k:M>] hashdb.hdb`	Creates a new hash database with the given configuration parameters.

Tables 3 and 4 show the optional parameters that can be used to specify database settings.

Hash Block Size
This setting specifies the hash block size used to generate hashes. The hash block size must be greater than or equal to the sector size of 512, and must be divisible by 512 in order to be byte aligned, as discussed in **subsection 2.1**.

Maximum Duplicates
This setting specifies the maximum number of duplicates of a hash value that *hashdb* may put into the database. A default value of 0 means unlimited, but this may be unreasonable. For example if a block is repeated many times and is thus not interesting, limit storing its duplicates using this setting.

Bloom
This setting controls whether the Bloom filter will be enabled or disabled.

Table 3: Settings for New Databases

Option	Verbose Option	Specification
`-p`	`--hash_block_size=`*hash_ block_ size*	Specifies the block size (hash_block_size) in bytes used to generate the hashes that will be stored in the database. Default is 4096 bytes.
`-m`	`--max_duplicates=`*maximum*	Specifies the maximum number of hash duplicates allowed. 0 value indicates there is no limit. Default is 0.

Table 4: Bloom Filter Settings

Option	Verbose Option	Specification
`-A`	`--bloom=`*enabled*|*disabled*	Enables or disables the Bloom filter.
`-B`	`--bloom_n=`*n*	Configures Bloom filter to work well for a database of (n) hashes.
`-C`	`--bloom_kM=`*k*:*M*	Available for advanced or experimental use. Configures Bloom filter with (k) hash functions and (M) bits per hash.

Bloom_n
This setting controls the size of the Bloom filter by using the number of hashes expected in the database as input.

Bloom_kM
This setting directly controls the number of hash functions and the number of bits per hash to use in the Bloom filter and is only recommended for experimentation. We recommend using `bloom_n` instead.

Example
To create an (empty) hash database named `mock_video.hdb`, type the following command:

■ `hashdb create mock_video.hdb`

The above command will create a database with all of the default hash database settings. Most users will not need to change those settings. Our DFXML file was created with a default block size of 4096 bytes. Users can specify either the option and value or the verbose option value for each parameter along with the create command, as in:

- `hashdb create --max_duplicates=20 mock_video.hdb`
- `hashdb create -m 20 mock_video.hdb`

The above two commands produce identical results, creating the database `mock_video.hdb` that will accept a maximum of 20 hash duplicates.

4.3 Importing and Exporting using DFXML

Hash databases may be imported from and exported to files in DFXML format. Once a database has been created, it may be populated with hash values from a DFXML file. Commands to import and export hashes are shown in Table 5. Note that there are other

Table 5: Commands for Importing and Exporting between DFXML Files and Hash Databases

Command	Usage	Description
import	`import [-r <repository name>] <DFXML file> <hashdb>`	Imports values from the DFXML file into the hash database. Command can optionally include a specific repository name to use for the set of hashes being imported
export	`export <hashdb.hdb> <DFXML file.xml>`	Exports hash database to the DFXML file

ways to populate a database besides importing from a DFXML file, including using other hash databases (discussed in **subsection 5.4**), by using the **bulk_extractor** *hashdb* scanner (discussed in **subsection 5.3**), and through the use of the import capability provided by the *hashdb* library API (discussed in **subsection 5.2**).

Using the DFXML file created in the previous section, type the following command:

- `hashdb import -r mock_video_repository mock_video.xml mock_video.hdb`

In the above command the option **-r** is used along with the repository name `mock_video_repository` to indicate the repository source of the block hashes being imported into the database. The repository name is used to keep track of the sources of hashes. Hash blocks contained in one database often originate from many different sources and the fileme may be the same. For example, if we add two separate but similar databases with partial overlap to a database, this will result in some duplicate hashes from multiple sources with the same filename. The repository name can be used with those duplicates to allow users to track all hashes back to their original sources. By default, the repository name used is the text `repository_` with the filename of the file being imported from appended after it.

The **import** command in the above example imports the contents of `mock_video.xml` into the database `mock_video.hdb`. *hashdb* prints the following to the command line to indicate that the hashes have been inserted into the database successfully:

```
hashdb changes (insert):
    hashes inserted: 2595
```

Listing 3: Excerpt of the `log.xml` indicating hash blocks were inserted

```
...
   <repository_name>mock_video_repository</repository_name>
    <timestamp name='begin import' delta='0.024016' total='0.024016'/>
    <timestamp name='end import' delta='0.015009' total='0.039025'/>
    <hashdb_changes>
      <hashes_inserted>2595</hashes_inserted>
    </hashdb_changes>
...
```

The database `mock_video.hdb` now holds 2595 hash values. If curious, you may navigate into the directory `mock_video.hdb` to observe its set of database files. The following lists the contents:

```
 4097 Mar  9 21:52 Bloom_filter_1
90112 Mar  9 21:56 hash_store
 3788 Mar  9 21:52 history.xml
 3573 Mar  9 21:56 log.xml
 3105 Mar  9 21:52 settings.xml
   47 Mar  9 22:21 source_filename_store.dat
 8192 Mar  9 21:56 source_filename_store.idx1
 8192 Mar  9 21:56 source_filename_store.idx2
   25 Mar  9 22:21 source_lookup_store.dat
 8192 Mar  9 21:56 source_lookup_store.idx1
 8192 Mar  9 21:56 source_lookup_store.idx2
 8192 Mar  9 21:56 source_metadata_store
   37 Mar  9 22:21 source_repository_name_store.dat
 8192 Mar  9 21:56 source_repository_name_store.idx1
 8192 Mar  9 21:56 source_repository_name_store.idx2
```

The file `log.xml` shows that a set of hash blocks have just been inserted. Listing 3 shows the excerpt of the log file that tracks this statistic. Users will prefer to run statistical commands such as this to get information about the contents of the database (and confirm that values were inserted):

- `hashdb size mock_video.hdb`

4.4 Database Manipulation

Databases may need to be merged together or common hash values may need to be subtracted out in order for them to be more suitable for scanning against. Commands that manipulate hash databases are outlined in Table 6. Destination databases are created if they do not exist yet.

4.4.1 Tracking Changes in Hash Databases

Statistics about hash database changes are reported on the console and to the log file and history file inside the hash database. These statistics show the number of hashes inserted or removed as a result of a command, and also show the number of hashes not inserted or not removed because specific conditions were not met. These statistics are shown in Table 7.

4.5 Scan Services

hashdb can be used to determine if a file, directory or disk image has content that matches previously identified content. This capability can be used, for example, to de-

Table 6: Commands to Manipulate Hash Databases

Command	Usage	Description
add	`add <source db> <destination db>`	Copies all of the hashes from *source db* to *destination db*
add_multiple	`add_multiple <source db1> <source db2> <destination db>`	Performs the union of *source db1* and *source db2* and copies all of the hash values from the union into *destination db*
add_repository	`add_repository <source db1> <destination db2> <repository namedb>`	Adds *source db1* to *destination db2* but only when the repository name matches
intersect	`intersect <source db1> <source db2> <destination db>`	Copies hash values common to both *source db1* and *source db2* into *destination db* where sources match
intersect_hash	`intersect_hash <source db1> <source db2> <destination db>`	Copies hash values common to both *source db1* and *source db2* into *destination db* even when source repository name and filename do not match
subtract	`subtract <source db1> <source db2> <destination db>`	Copies hash values found in *source db1* but not in *source db2* into *destination db* where sources match
subtract_hash	`subtract <source db1> <source db2> <destination db>`	Copies hash values found in *source db1* but not in *source db2* into *destination db* even when source repository name and filename do not match
deduplicate	`deduplicate <source db> <destination db>`	Copies all non-duplicate hash values from *source db* into *destination db*

termine if a set of files contains a specific file excerpt or if a media image contains a video fragment. Forensic investigators can use this feature to search for blacklisted content. To scan media for hash values, run using the **bulk_extractor** *hashdb* scanner on a media image file and provide a hash database created by *hashdb* as input. Scan services are shown in Table 8.

Table 7: Database Changes Resulting from Commands that Manipulate Hash Databases

Statistic	Meaning
`hashes_inserted`	Number of hashes inserted.
`hashes_not_inserted_ mismatched_hash_block_size`	Number of hashes not inserted because the hash block size of the block requested for insert was incorrect. For example if the database requires a hash block size of 4096, and the file size is 5096 bytes, the last block hash size will be an (invalid) 100 bytes, so it will not be inserted. NOTE: this will occur almost every time hash blocks are added to the database since the remaining bytes of every file are not likely to be comprised of the exact hash size. This is not an error.
`hashes_not_inserted_ invalid_byte_alignment`	Number of hashes not inserted because the file offset was not byte aligned. If the database expects a byte alignment of 512 and the *hashdb* user tries to add a hash at byte 80, *hashdb* will detect that 80 does not fall on a 512 byte boundary ($80 \bmod 512 \neq 0$).
`hashes_not_inserted_ exceeds_max_duplicates`	Number of hashes not inserted because they exceed the max duplicates value. For example, user sets max duplicates with `-m 20` and the run attempts to import 30 hashdigests calculated from 30 NULL blocks of input, so we see 20 max duplicates.
`hashes_not_inserted_ duplicate_element`	Number of hashes not inserted because they are duplicate elements. The user attempts to import a hash where the hash value, repository name, filename, and its file offset are all the same.
`hashes_removed`	Number of hashes removed.
`hashes_not_removed_ mismatched_hash_block_size`	Number of hashes not removed because the hash block size of the block requested for removal did not match the hash block size the database was configured to accept.
`hashes_not_removed_ invalid_byte_alignment`	Number of hashes not removed because the file offset was not byte aligned.
`hashes_not_removed_ no_hash`	Number of hashes not removed because the hash blocks requested for removal did not exist in the database.
`hashes_not_removed_ no_element`	Number of hashes not removed because the hashes, specifically identified by hash value, repository name, filename, and its file offset do not exist in the database, indicating a possible mistake in database management.
`source_metadata_inserted`	Number of metadata items inserted.
`source_metadata_inserted_ already_present`	Number of metadata items not inserted because the specified metadata is already present.

Table 8: Commands that Provide Scan Services

Command	Usage	Description
scan	`scan <path_or_socket> <DFXML file>`	Scans the hashdb for hashes that match hashes in the DFXML file and prints out matches
scan_hash	`scan_hash <path_or_socket> <hash value>`	Scans the hashdb for the specified hash value and prints out whether it matches
scan_expanded	`scan <hashdb> <DFXML file>`	Scans the hashdb for hashes that match hashes in the DFXML file and prints out the repository name, filename, and file offset for each hash that matches
scan_expanded_hash	`scan_expanded_hash <hashdb> <hash value>`	Scans the hashdb for the specified hash value and prints out the repository name, filename, and file offset for each hash that matches
server	`server <hashdb> <port number>`	Starts a scan service at the given port number

First, identify the media that you would like to scan. For this example, we download and use video file `mock_video_redacted_image` available at `http://digitalcorpora.org/downloads/hashdb/demo`.

Second, identify the existing hash database that will be used to search for hash value matches. We'll use the database `mock_video.hdb` that we created in the previous section. That database contains all of the block hash values from a media image.

Finally, run **bulk_extractor** from the command line and send the required parameters to the *hashdb* scanner using the **-S** option. Run the following command:

■ `bulk_extractor -e hashdb -o outdir -S hashdb_mode=scan -S hashdb_scan_path_or_socket=mock_video.hdb mock_video_redacted_image`

This command tells **bulk_extractor** to enable the *hashdb* scanner and to run it in "scan" mode to try to match the values found in the local database `mock_video.hdb`. Note: other run options using **bulk_extractor** are discussed further in **subsection 5.3**.

Listing 4 shows the output printed to the command line as a result of the above **bulk_extractor** *hashdb* scan command.

Listing 4: Output from **bulk_extractor** *hashdb* scan

```
bulk_extractor version: 1.4.1
Input file: mock_video_redacted_image
Output directory: outdir1
Disk Size: 12596738
Threads: 4
All data are read; waiting for threads to finish...
Time elapsed waiting for 1 thread to finish:
     (timeout in 60 min .)
Time elapsed waiting for 1 thread to finish:
    6 sec (timeout in 59 min 54 sec.)
Thread 0: Processing 0

All Threads Finished!
Producer time spent waiting: 0 sec.
Average consumer time spent waiting: 4.69167 sec.
Phase 2. Shutting down scanners
Phase 3. Creating Histograms
   ccn histogram...   ccn_track2 histogram...   domain histogram...
   email histogram...   ether histogram...   find histogram...
   ip histogram...   telephone histogram...   url histogram...
   url microsoft-live...   url services...   url facebook-address...
   url facebook-id...   url searches...
Elapsed time: 6.33812 sec.
Total MB processed: 125
Overall performance: 1.98746 MBytes/sec (0.496864 MBytes/sec/thread)
Total email features found: 0
```

All hash block matches discovered in the hash database are printed to the **bulk_extractor** output file `identified_blocks.txt`. Listing 5 shows the contents of that file after the **bulk_extractor** run. Each line of the file corresponds to one hash block from the input data provided that was matched in the database. The number at the beginning of the line is the Forensic Path.

The second column of the `identified_blocks.txt` file shows the actual block hash value. The final column is the number of times this block hash value has been added to the hash database. It is a count of hash duplicates. Hash duplicates occur when the hash value is the same but any part of the source information including repository name, filename or offset, is unique. In this case, each hash values shown has only been added to the database once.

Users may be put off by the quantity of matches incurred by low-entropy data in their databases such as blocks of zeros or metadata header blocks from files that are otherwise unique. Database manipulation commands, **subsection 4.4** , can mitigate this, for example:

- Use the "subtract" command to remove known whitelist data created from sources such as "brand new" operating system images and the NSRL.
- Alternatively, use the "deduplicate" command to copy all hash values that have been imported exactly once.

These commands are provided to manage false positives.

Listing 5: The `identified_blocks.txt` file produced by **bulk_extractor**'s *hashdb* scanner. First column is the forensic path, second is the hash value, and third is the number of times the hash value occurs in the database

```
# BANNER FILE NOT PROVIDED (-b option)
# BULK_EXTRACTOR-Version: 1.5.5-dev ($Rev: 10844 $)
# Feature-Recorder: identified_blocks
# Filename: /home/bdallen/demo8/demo_video_redacted_image
# Feature-File-Version: 1.1
12452352        3b6b477d391f73f67c1c01e2141dbb17        {"count":1}
12456448        89a170b6b9a948d21d1d6ee1e7cdc467        {"count":1}
12460544        f58a09656658c6b41e244b4a6091592c        {"count":1}
12464640        1d0abbddf1344ac751d17604bdd9ebe8        {"count":1}
12468736        16d75027533b0a5ab900089a244384a0        {"count":1}
12472832        97068927ff7ca0c4d27ac527474065bc        {"count":1}
12476928        80a403ea48854676501a02e390a69699        {"count":1}
12481024        7de953ea563c4df1f8369d8dd2cfb4d9        {"count":1}
12485120        1b803bd6e014d1855e6f8413041c2b07        {"count":1}
12489216        cf49adf3285b983d9f8d60497290bfd2        {"count":1}
12493312        4cc415709e205ac0ef5b5dcfb77936b6        {"count":1}
12497408        0c5c611edc8dfd34f85c6cbf88702e51        {"count":1}
12501504        4a93e65fb187d71c2b8b5697f1460e3d        {"count":1}
12505600        a667f79e6446222092257af1780f6a9f        {"count":1}
12509696        aec94ab99f591f507b3c27424a0b52c5        {"count":1}
12513792        c6361fe0eb4f7b13bac6529e1cdd8ea4        {"count":1}
```

Listing 6: The `identified_sources.txt` file produced by post-processing the `identified_blocks.txt` file. First column is the forensic path, second is the hash value, and third is the repository name, filename, and file offset

```
12464640        1d0abbddf1344ac751d17604bdd9ebe8        {"count":1,"source
_id":1,"repository_name":"temp1","filename":"/home/bdallen/demo8/demo_vide
o.mp4","filesize":10630146,"hashdigest":"a003483521c181d26e66dc09740e939d"
}
```

4.6 Statistics

Various statistics are available about a given hash database including the size of a database, where its hashes were sourced from, a histogram of its hashes, and more. Table 9 describes available statistics.

Table 9: Commands that provide Statistics about Hash Databases

Command	Usage	Description
size	`size <hashdb>`	Prints out size information relating to the database
sources	`sources <hashdb>`	Provides a top-level view of the repository names and filenames in the database. It prints out all repositories and files that have contributed to this database
histogram	`histogram <hashdb>`	Prints a hash distribution for the hashes in the *hashdb*
duplicates	`duplicates <hashdb> <number>`	Prints out hashes in the database that are sourced the given number of times
hash_table	`hash_table <hashdb> <repository name> <filename>`	Prints out the hashes and their offsets for the specified source
expand_identified_blocks	`expand_identified_blocks <hashdb> <identified_blocks.txt>`	Prints out the repository name, filename, and file offset of each hash in the *hashdb* for each hash feature in the `identified_blocks.txt` input file
explain_identified_blocks	`expand_identified_blocks [-m <number>] <hashdb> <identified_blocks.txt>`	Prints out hash and source tables for sources with hashes observed no more than the maximum number of times, default max 20, for each hash feature in the `identified_blocks.txt` input file

To identify the source information associated with the hash values found in `identified_blocks.txt`, type the following command using the hash database and `identified_blocks.txt` file as

input (command should be run from the same directory in which you ran **bulk_extractor**):

- `hashdb expand_identified_blocks mock_video.hdb outdir/identified_blocks.txt > identified_sources.txt`

The above command pipes the output directly into the file `identified_sources.txt`. Each line of the file will provide the source information for one of the identified hash blocks. An example line from this file is shown in Listing 6, which shows that the block at Forensic path 12464640 matches the block 10498048 bytes into the `mock_video.mp4` files in the hash database, indicating a positive match.

A table of more relevant hashes and source information may be generated using the `explain_identified_blocks` command. To generate these tables for `identified_blocks.txt`, type the following command using the hash database and `identified_blocks.txt` file as input (command should be run from the same directory in which you ran **bulk_extractor**):

- `hashdb explain_identified_blocks mock_video.hdb outdir/identified_blocks.txt > explained_hashes_and_sources.txt`

The above command pipes the output directly into the file `explain_hashes_and_sources.txt`. The output of this command is shown in Listing 7. The first table in file `explain_hashes_and_sources.txt` shows block hashes, their source ID, and the offset into the file where they were sourced from. The second table in the file shows sources, in this case, just one source corresponding to source ID 1. Each line of source information describes the repository name, filename, file size, and file hash for a given source ID.

4.7 Tuning

Tuning commands are provided to improve performance or allow upgrade capability, see Table 10. For Bloom filter settings, see Table 4.

Table 10: Commands that Tune Hash Databases

Command	Usage	Description
rebuild_bloom	`rebuild_bloom[--bloom <state>] [--bloom_n <n>] [--bloom_kM <k:M>] <hashdb.hdb>`	Rebuilds the Bloom filter using the provided settings, see Table 4
upgrade	`upgrade <hashdb.hdb>`	Upgrades the *hashdb* v1.0.0 database to be compatible with *hashdb* v1.1.1

4.8 Performance Analysis

Performance analysis commands for analyzing *hashdb* performance are available, see Table 11.

Listing 7: The `identified_hashes_and_sources.txt` file produced by post-processing the `identified_blocks.txt` file using the `explain_identified_blocks` command

```
# hashdb-Version: 1.1.0
# explain_identified_blocks-command-Version: 1
# hashes
["0c5c611edc8dfd34f85c6cbf88702e51",{"count":1},[{"source_id":1,"file_offs
et":10530816}]]
["16d75027533b0a5ab900089a244384a0",{"count":1},[{"source_id":1,"file_offs
et":10502144}]]
["1b803bd6e014d1855e6f8413041c2b07",{"count":1},[{"source_id":1,"file_offs
et":10518528}]]
["1d0abbddf1344ac751d17604bdd9ebe8",{"count":1},[{"source_id":1,"file_offs
et":10498048}]]
["3b6b477d391f73f67c1c01e2141dbb17",{"count":1},[{"source_id":1,"file_offs
et":10485760}]]
["4a93e65fb187d71c2b8b5697f1460e3d",{"count":1},[{"source_id":1,"file_offs
et":10534912}]]
["4cc415709e205ac0ef5b5dcfb77936b6",{"count":1},[{"source_id":1,"file_offs
et":10526720}]]
["7de953ea563c4df1f8369d8dd2cfb4d9",{"count":1},[{"source_id":1,"file_offs
et":10514432}]]
["80a403ea48854676501a02e390a69699",{"count":1},[{"source_id":1,"file_offs
et":10510336}]]
["89a170b6b9a948d21d1d6ee1e7cdc467",{"count":1},[{"source_id":1,"file_offs
et":10489856}]]
["97068927ff7ca0c4d27ac527474065bc",{"count":1},[{"source_id":1,"file_offs
et":10506240}]]
["a667f79e6446222092257af1780f6a9f",{"count":1},[{"source_id":1,"file_offs
et":10539008}]]
["aec94ab99f591f507b3c27424a0b52c5",{"count":1},[{"source_id":1,"file_offs
et":10543104}]]
["c6361fe0eb4f7b13bac6529e1cdd8ea4",{"count":1},[{"source_id":1,"file_offs
et":10547200}]]
["cf49adf3285b983d9f8d60497290bfd2",{"count":1},[{"source_id":1,"file_offs
et":10522624}]]
["f58a09656658c6b41e244b4a6091592c",{"count":1},[{"source_id":1,"file_offs
et":10493952}]]
# sources
{"source_id":1,"repository_name":"temp1","filename":"/home/bdallen/demo8/d
emo_video.mp4","filesize":10630146,"hashdigest":"a003483521c181d26e66dc097
40e939d"}
```

Table 11: Commands that Support *hashdb* Performance Analysis

Command	Usage	Description
add_random	`add_random -r [<repository name>] <hashdb.hdb> <count>`	Adds count random hashes to the given database, creating timing data in the `log.xml` file
scan_random	`scan_random <hashdb.hdb> <hashdb copy.hdb>`	Scans the given database, creating timing data in the `log.xml` file

4.9 Importing and Scanning Using the bulk_extractor *hashdb* Scanner

The **bulk_extractor** *hashdb* scanner may be used to import hashes and to scan for hashes. Example syntax for this scanner is shown in Table 12. Scanner options are shown in Table 13.

Table 12: bulk_extractor *hashdb* Scanner Commands

Command	Usage	Description
import	`bulk_extractor -E hashdb -S hashdb_mode=import -o outdir1 my_image1`	Import hashes from image into outdir1/hashdb.hdb
scan	`bulk_extractor -E hashdb -S hashdb\_mode=scan -S hashdb_scan_path_or_socket =outdir1/hashdb.hdb -o outdir2 my_image`	Scan image for hashes matching hashes in outdir1/hashdb.hdb

Table 13: bulk_extractor *hashdb* Scanner Options

Option	Default	Specification
`hashdb_mode`	`none`	The mode for the scanner, one of `[none|import|scan]`. For "none", the scanner is active but performs no action. For "import", the scanner imports block hashes. For "scan", the scanner scans for matching block hashes.
`hashdb_block_size`	4096	Block size, in bytes, used to generate hashes.
`hashdb_ignore_empty_blocks`	`YES`	Selects to ignore empty blocks, one of `[YES|NO]`.
`hashdb_scan_path_or_socket`		The file path to a hash database or socket to a hashdb server to scan against. Valid only in scan mode. No default provided. Value must be specified if in scan mode.
`hashdb_scan_sector_size`	512	Selects the scan sector size. Scans along sector boundaries. Valid only in scan mode.
`hashdb_scan_max_features`	0	The maximum number of feature lines to record or 0 for no limit. Valid only in scan mode.
`hashdb_import_sector_size`	4096	Selects the import sector size. Imports along sector boundaries. Valid only in import mode.
`hashdb_import_repository_name`	`default_repository`	Selects the repository name to attribute the import to. Valid only in import mode.
`hashdb_import_max_duplicates`	0	The maximum number of duplicates to import for a given hash value. Valid only in import mode. Use default of 0 for no limit.

5 Use Cases for *hashdb*

There are many different ways to utilize the functionality provided by the *hashdb* tool. In this section, we highlight some of the most common uses of the system.

5.1 Querying for Source or Database Information

Users can scan a hash database directly using various querying commands. Those commands are outlined in Table 8. The "scan" command allows users to search for hash blocks in a DFXML file that match hash blocks in a database. This can be used to determine if content from raw media matches fragments of previously encountered data contained in a database. For example, a forensic investigator may have a disk image in evidence. Using that disk image and third party tool such as **md5deep**, the investigator can generate a DFXML file of sector block hashes. The investigator can then run

the "scan" command with the DFXML file to see if any content from the disk image matches hash blocks of known fragments of previously encountered data. The "sources" and "statistics," commands provide information about the source of the hash blocks and statistics about the database itself.

Each hash block stored in the database is stored with three separate pieces of source information. This complete source information is provided for each source record in the hash database, including hash duplicates. The "expand_identified_blocks" command prints out this information for hashes identified in `identified_blocks.txt` feature files. The source information includes:

- Repository Name: The repository name indicates the provenance of the dataset. It is its description information, such as "Company X's intellectual property files". The DFXML file generated by **md5deep** does not include a repository name. To specify your own repository name when importing, use the `-r <repository name>` option, specifically, `import -r <repository name>`. Otherwise, a default repository name will be generated, consisting of the text `repository_` followed by the filename of the DFXML file, including its full path.
- Filename: The file from which the block hash was sourced. Typically, hash values are sourced from files or directories of file using **md5deep** with the recursive directory "-r" option. If hash values are source from raw media using the **bulk_extractor** *hashdb* scanner in `import` mode, then the Forensic Path is used as the source information.
- File Offset: the offset, in bytes, into the file where the block hash was calculated.

5.1.1 Querying a Remote Hash Database

hashdb also provides the capability to set up a remote socket to "scan" an existing database. Users can set up a database on the socket and then access the "scan" command via that socket. To set up the scan service, users need to provide the name of the hash database and the TCP socket that will be available for clients. For example, the following command starts hashdb as a server service for the hash database at path `my_hashdb.hdb` at socket endpoint `tcp://*:14500`:

■ `hashdb server my_hashdb.hdb tcp://*:14500`

This example searches the *hashdb* server service available at socket tcp://localhost:14500 for hashes that match those in the DFXML file `my_dfxml.xml`:

■ `hashdb scan tcp://localhost:14500 my_dfxml.xml`

The only socket service *hashdb* provides is for scanning. The *hashdb* "scan" command, the *hashdb* library API constructor for scanning and the **bulk_extractor** *hashdb* scanner in scan mode all accept a path or a socket and are the only place where sockets are used.

A note of caution, when a socket server service is opened, its associated hash databased is opened. Do not make changes to a database when it is opened as a socket server service. Although this will not corrupt the hash database, it is likely to cause the server service to perform incorrectly.

It is likely that the TCP port number you choose to use will need to be enabled by your firewall on the Server side.

There is no security in the current protocol. It should only be used on a private network.

5.2 Writing Software that works with *hashdb*

hashdb provides an API that other software programs can use to access two important database capabilities. The file `hashdb.hpp` found in the *src* directory contains the complete specification of the API. That complete file is also contained in Appendix C of this document. The two key features provided by the API include the ability to import values into a hash database and the ability to scan media for any values matching those in a given hash database. The **bulk_extractor** program uses the *hashdb* API to implement both of these capabilities.

5.3 Scanning or Importing to a Database Using bulk_extractor

The **bulk_extractor** *hashdb* scanner allows users to query for fragments of previously encountered hash values and populate a hash database with hash values. Options that control the *hashdb* scanner are provided to **bulk_extractor** using the "-S name=value" command line parameters. When **bulk_extractor** executes, the parameters are sent directly to the scanner.

For example, the following command runs the **bulk_extractor** *hashdb* scanner in import mode and adds hash values calculated from the disk image `my_image` to a hash database:

- `bulk_extractor -e hashdb -o outputDir -S hashdb_mode=import my_image`

Note, **bulk_extractor** will place feature file and other output not relevant to the *hashdb* application in the "outputDir" directory. When using the import command, the output directory will contain a newly created hash database called `hashdb.hdb`. That database can then be copied or added to a hash database in another location.

5.4 Updating Hash Databases

hashdb provides users with the ability to manipulate the contents of hash databases. The specific command line options for performing these functions are described in Table 6. *hashdb* databases are treated as sets with the add, subtract and intersect commands basically using add, subtract and intersect set operations. For example, the following command will copy all non-duplicate values from `mock_video.hdb` into `mock_video_dedup.hdb` :

- `hashdb deduplicate mock_video.hdb mock_video_dedup.hdb`

Whenever a database is created or updated, *hashdb* updates the file `log.xml`, found in the database's directory with information about the actions performed.

After each command to change a database, statistics about the changes are writen in the `log.xml` file and to `stdout`. Table 7 shows all of the statistics tracked in the log file along with their meaning. The value of each statistic is the number of times the event

happened during the command. For example, if 280 hashes are removed, the statistic "hashes_removed" will be marked with a value of 280.

5.4.1 Update Commands and "Duplicate" Hashes

Commands that add or import hashes of the same value will result in hash duplicates if the source information is unique. If hash and source values are identical (including repository name), no hash values are added with the **add** or **import** commands. The **intersect** and **subtract** commands do not require source information to match. An intersection occurs when hatches match, regardless of whether the source information matches. Similarly, hash values are also subtracted from the database regardless of whether or not their source information matches. The update statistics specified in the log file (shown in 7) will specify the results of each of these commands to help users track changes.

As discussed previously, users can only specify the repository name with the **import** command. As databases become large, the repository name for each hash value will help identify important source information. Users should plan on importing data with specific repository names whenever possible to avoid source confusion later.

Finally, we provide two philosophies for mitigating duplicate hash bloat:

- If you know you have imported the same blacklist data twice, and you do not want to manage a 'whitelist' database, `deduplicate` is a quick and easy way to get rid of low-entropy noise.
- If your database has blacklist data from more than one source or you just want tighter control about what you want to remove and are willing to use a 'whitelist' database to remove hashes to improve lookup speed or to reduce noise about uninteresting hashes found, use `subtract`.

5.5 Optimizing a Hash Database

For large databases, it takes a bit of time to look up a hash value to determine whether it is in the database. This time adds up when scanning for millions of hash values. As an optimization, *hashdb* provides the capability to utilize a Bloom filter to speed up performance during hash queries. A Bloom filter is a data structure that is used to determine if a member is not part of a set. In *hashdb*, a Bloom filter can be used to quickly indicate if a hash value is not part of the database. If the Bloom filter indicates a hash value is definitely not in the hash database, no actual hash database look up is necessary. If the Bloom filter says the hash value may be in the database, a look up is still required and no time is saved. The disadvantage of using a Bloom filter is that it can consume large amounts of disk space and memory. A Bloom filter that is too small fills up and then too often gives false positives that indicate the hash value might be in the database. A Bloom filter that is too large will take up too much memory and disk space.

hashdb has a Bloom filter. Users can enable or disable this Bloom filter and tune it using information about the hashes and hash functions. The optimal configuration for the Bloom filter depends on the size of the dataset. Although several tuning controls are available, we recommend only using "bloom1_n <n>," where "n" is the expected

Listing 8: Excerpt of a DFXML exported by *hashdb*

```
<fileobject>
  <repository_name>mock_video_repository</repository_name>
  <filename>/home/bdallen/demo/mock_video.mp4</filename>
  <byte_run file_offset='6938624' len='4096'>
    <hashdigest type='MD5'>0016aa775765eb7929ec06dea25b6f0e</hashdigest>
  </byte_run>
</fileobject>
<fileobject>
  <repository_name>mock_video_repository</repository_name>
  <filename>/home/bdallen/demo/mock_video.mp4</filename>
  <byte_run file_offset='3837952' len='4096'>
    <hashdigest type='MD5'>00183a37c80b3ee02cb4bdd3e7d7e9d2</hashdigest>
  </byte_run>
</fileobject>\
<fileobject>
  <repository_name>mock_video_repository</repository_name>
  <filename>/home/bdallen/demo/mock_video.mp4</filename>
  <byte_run file_offset='5652480' len='4096'>
    <hashdigest type='MD5'>00513c9484ebc957eb928adf30504bc9</hashdigest>
  </byte_run>
</fileobject>
```

number of hashes in the dataset. If users want to improve scan speed, they should tune Bloom 1 based on their database size using this option. The default setting for the Bloom filter in *hashdb* is enabled, is tuned for about 45,000,000 hashes, and takes up about 33MB of space.

5.6 Exporting Hash Databases

Users can export hashes from a hash database to a DFXML file using the "export" command. For example, the following command will export the `mock_video.hdb` database to the file `demoVideoHashes.xml`:

- `hashdb export mock_video.hdb demoVideoHashes.xml`

Note that the DFXML that *hashdb* exports is compatible but different from the DFXML created by `md5deep`. Listing 8 shows an example excerpt of a DFXML file exported from *hashdb*. The differences are:

1. The first offset is 6938624, not 0, because the output is sorted by hash value.
2. There is a `fileobject` tag wrapping every individual hash.
3. Every entry includes a `repository_name` tag.

6 Worked Example: Finding Similarity Between Disk Images

The worked example provided is intended to further illustrate how to use *hashdb* to answer specific questions and perform specific tasks. This example uses a publicly available dataset and can be replicated by readers of this manual. In this example, we walk

through the process of using *hashdb* (and **bulk_extractor**) to find the similarities between two separate disk images. We generate a hash database of block hashes from each media image and then obtain common block hashes by taking the intersection of the two databases.

First, we download two files to use for comparison. The disk images are called `jo-favorites-usb-2009-12-11.E01` and `jo-work-usb-2009-12-11.E01`. Both files are available at `http://digitalcorpora.org/corp/nps/scenarios/2009-m57-patents/drives-redacted/`. Specifically with this example, we will be comparing the contents of two fictional USB drives.

Then, we run **bulk_extractor** on each disk image separately:

■ `bulk_extractor -o workOutput -S hashdb_mode=import  jo-work-usb-2009-12-11.E01`

bulk_extractor writes the following output to the screen, indicating a successful run:

```
bulk_extractor version: 1.4.1
Input file: jo-work-usb-2009-12-11.E01
Output directory: workOutput
Disk Size: 131072000
Threads: 1
21:57:21 Offset 67MB (51.20%) Done in  0:00:24 at 21:57:45
All data are read; waiting for threads to finish...
Time elapsed waiting for 1 thread to finish:
    1 sec (timeout in 59 min 59 sec.)
All Threads Finished!
Producer time spent waiting: 38.5587 sec.
Average consumer time spent waiting: 1.85768 sec.
*****************************************
** bulk_extractor is probably CPU bound. **
**    Run on a computer with more cores  **
**      to get better performance.       **
*****************************************
Phase 2. Shutting down scanners
Phase 3. Creating Histograms
   ccn histogram...   ccn_track2 histogram...   domain histogram...
   email histogram...   ether histogram...   find histogram...
   ip histogram...   telephone histogram...   url histogram...
   url microsoft-live...   url services...   url facebook-address...
   url facebook-id...   url searches...
Elapsed time: 47.6743 sec.
Total MB processed: 1310
Overall performance: 2.74932 MBytes/sec (2.74932 MBytes/sec/thread)
Total email features found: 31
```

Next, run **bulk_extractor** on the other usb drive disk image:

■ `bulk_extractor -o favoritesOutput -S hashdb_mode=import jo-favorites-usb-2009-11.E01`

bulk_extractor runs, printing the following to the screen:

```
bulk_extractor version: 1.4.1
Input file: jo-favorites-usb-2009-12-11.E01
Output directory: favoritesOutput
Disk Size: 1048576000
Threads: 1
21:59:44 Offset 67MB (6.40%) Done in  0:05:07 at 22:04:51
22:00:08 Offset 150MB (14.40%) Done in  0:04:30 at 22:04:38
22:00:32 Offset 234MB (22.40%) Done in  0:03:59 at 22:04:31
22:00:40 Offset 318MB (30.40%) Done in  0:02:55 at 22:03:35
```

```
22:00:41 Offset 402MB (38.40%) Done in  0:02:05 at 22:02:46
22:00:42 Offset 486MB (46.40%) Done in  0:01:31 at 22:02:13
22:00:44 Offset 570MB (54.40%) Done in  0:01:07 at 22:01:51
22:00:45 Offset 654MB (62.40%) Done in  0:00:49 at 22:01:34
22:00:47 Offset 738MB (70.40%) Done in  0:00:35 at 22:01:22
22:00:48 Offset 822MB (78.40%) Done in  0:00:23 at 22:01:11
22:00:50 Offset 905MB (86.40%) Done in  0:00:13 at 22:01:03
22:00:51 Offset 989MB (94.40%) Done in  0:00:05 at 22:00:56
All data are read; waiting for threads to finish...
Time elapsed waiting for 1 thread to finish:
     (timeout in 60 min .)
All Threads Finished!
Producer time spent waiting: 76.8042 sec.
Average consumer time spent waiting: 1.79526 sec.
****************************************
** bulk_extractor is probably CPU bound. **
**    Run on a computer with more cores  **
**      to get better performance.       **
****************************************
Phase 2. Shutting down scanners
Phase 3. Creating Histograms
   ccn histogram...   ccn_track2 histogram...   domain histogram...
   email histogram...   ether histogram...   find histogram...
   ip histogram...   telephone histogram...   url histogram...
   url microsoft-live...   url services...   url facebook-address...
   url facebook-id...   url searches...
Elapsed time: 89.1399 sec.
Total MB processed: 10485
Overall performance: 11.7633 MBytes/sec (11.7633 MBytes/sec/thread)
Total email features found: 2
```

After **bulk_extractor** runs, two output directories are created. Each directory contains a hash database called *hashdb.hdb*. The hash databases each contain cryptographic block hashes produced from the disk images. Next we create database *intersection.hdb* with values that are common between the two databases using the following command:

■ `hashdb intersect workOutput/hashdb.hdb favoritesOutput/hashdb.hdb intersection.hdb`

hashdb prints the following indicating that 32 hashes were inserted successfully and 8 hashes were not inserted because they were considered to be duplicate elements (same hash and same source information):

```
hashdb changes (insert):
    hashes inserted=32
    hashes not inserted, duplicate element=8
```

Now, the database *intersection.hdb* contains hashes common to both disk images.

Here are some ways to gain knowledge from the common hashes identified:

- Constrain the matches further by using the `intersect` command to intersect the database with a blacklist database, and then use the `get_sources` command to find the blacklist filenames that these hash values correspond to.
- Use **bulk_extractor Viewer** to navigate to the data that these hashes were generated from to see if the raw data there is significant.
- If the scanned image contains a file system, try to use the **fiwalk** tool to carve the files from which the hash values were calculated.

7 Troubleshooting

All *hashdb* users should join the **bulk_extractor** users Google group for more information and help with any issues encountered. To join, send an email to **bulk_extractor-users+subscribe@ googlegroups.com**.

8 Related Reading

There are other articles related to block hashing, and its practical and research applications. Some of those articles are specifically cited throughout this manual. Other useful references include but are not limited to:

- Garfinkel, Simson, Alex Nelson, Douglas White and Vassil Rousseve. Using purpose-built functions and block hashes to enable small block and sub-file forensics. Digital Investigation. Volume 7. 2010. Page S13–S23. `http://www.dfrws.org/2010/proceedings/2010-302.pdf`.
- Foster, Kristina. Using Distinct Sectors in Media Sampling and Full Media Analysis to Detect Presence of Documents From a Corpus. Naval Postgraduate School Masters Thesis, September 2012. `http://calhoun.nps.edu/public/handle/10945/17365`.

References

[1] Bradley, J., and Garfinkel, S. bulk_extractor users guide, September 2013. `http://digitalcorpora.org/downloads/bulk_extractor/BEUsersManual.pdf`.

[2] Garfinkel, S. Digital forensics XML and the DFXML toolset. *Digital Investigation 8* (February 2012), 161–174. `http://www.sciencedirect.com/science/article/pii/S1742287611000910`.

[3] Young, J., Foster, K., Garfinkel, S., and Fairbanks, K. Distinct sector hashes for target file detection. *IEEE Computer* (December 2012). `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6311397`.

Appendices

A *hashdb* Quick Reference

General Usage

`hashdb <command> <options> <parameters>`	Run hashdb command, `-q` for quiet mode, `-f <flags>` to control B-Tree flags

New Database

`create [-p <hash block size>] [-m <maximum duplicates>] [<bloom settings>] <hashdb.hdb>`	Create a new hash database

Import/Export

`import [-r <repository name>] <hashdb.hdb> <dfxml.xml>`	Import from DFXML file into hash database
`export <hashdb.hdb> <dfxml.xml>`	Export hash database to DFXML file

Database Manipulation

`add <A.hdb> <B.hdb>`	$A + B \rightarrow B$ Add A into B
`add_multiple <A.hdb> <B.hdb> <C.hdb>`	$A + B \rightarrow C$ add A and B into C
`add_repository <A.hdb> <B.hdb> <repository name>`	$A + B \rightarrow B$ Add A into B but only when the repository name matches
`intersect <A.hdb> <B.hdb> <C.hdb>`	$A \cap B \rightarrow C$ add intersection of A and B into C
`intersect_hash <A.hdb> <B.hdb> <C.hdb>`	$A \cap B \rightarrow C$ intersect into C when hashes match
`subtract <A.hdb> <B.hdb> <C.hdb>`	$A - B \rightarrow C$ add A but not B into C
`subtract_hash <A.hdb> <B.hdb> <C.hdb>`	$A - B \rightarrow C$ add A but not hashes in B into C
`deduplicate <A.hdb> <B.hdb>`	Copy $A \rightarrow B$ except for hashes with duplicates

Scan Services

`scan <path or socket> <dfxml.xml>`	Scan DFXML file for matching hashes
`scan_hash <path or socket> <hash value>`	Scan for hash match
`scan_expanded <hashdb.hdb> <dfxml.xml>`	Scan DFXML file for matches showing all sources
`scan_expanded_hash <hashdb.hdb> <hash value>`	Scan for hash match showing all sources
`server <hashdb.hdb> <port number>`	Start scan service at port

Statistics

`size <hashdb.hdb>`	Print sizes of internal database tables
`sources <hashdb.hdb>`	Print source metadata
`histogram <hashdb.hdb>`	Print hash distribution
`duplicates <hashdb.hdb> <number>`	Print hashes sourced the given number of times
`hash_table <hashdb.hdb> <repository name> <filename>`	Print the hashes associated with this source
`expand_identified_blocks <hashdb.hdb> <identified_blocks.txt>`	Expand to include source information for each source
`explain_identified_blocks [-m <number>] <hashdb.hdb> <identified_blocks.txt>`	Print information about less frequently observed hashes

Tuning

`rebuild_bloom [<bloom settings>] <hashdb.hdb>`	Rebuild Bloom filter
`upgrade <hashdb.hdb>`	Make database from v1.0.0 compatible with v1.1.0

Performance Analysis

`add_random [-r <repository name>] <hashdb.hdb> <count>`	Add random hashes, log performance in `log.xml`
`scan_random <hashdb.hdb> <copy.hdb>`	Scan random hashes, log performance in `log.xml`

bulk_extractor Scanner

`bulk_extractor -E hashdb -S hashdb_mode=import -o outdir1 -R my_import_dir`	Import directory
`bulk_extractor -E hashdb -S hashdb_mode=scan -S hashdb_scan_path_or_socket= outdir1/hashdb.hdb -o outdir2 my_image2`	Scan image

B Output of *hashdb* Help Command

```
hashdb Version 1.1.1
Usage: hashdb -h | -H | -v | -V | <command>
  -h, --help         print this message
  -H, --Help         print this message plus usage notes and examples
  -v, -V, --version, --Version    print version number
  -q, --quiet        quiet mode
  -f, --flags=flags  set B-Tree flags, any of: preload:cache_branches:
                     least_memory:low_memoy:balanced:fast:fastest

hashdb supports the following commands:

New database:
  create [options] <hashdb>
    Create a new <hashdb> hash database.

    Options:
    -p, --hash_block_size=<hash block size>
      <hash block size>, in bytes, of hashes(default 4096)
      expected <hash block size>, in bytes, or 0 for no restriction
      (default 4096)
    -m, --max_duplicates=<maximum>
      <maximum> number of hash duplicates allowed, or 0 for no limit
      (default 0)
    --bloom <state>
      sets bloom filter <state> to enabled | disabled (default enabled)
    --bloom_n <n>
      expected total number <n> of distinct hashes (default 45634027)
    --bloom_kM <k:M>
      number of hash functions <k> and bits per hash <M> (default <k>=3
      and <M>=28 or <M>=value calculated from value in --bloom_n)

    Parameters:
    <hashdb>   the file path to the new hash database to create

Import/Export:
  import [-r <repository name>] <hashdb> <DFXML file>
    Import hashes from file <DFXML file> into hash database <hashdb>.

    Options:
    -r, --repository=<repository name>
      The repository name to use for the set of hashes being imported.
      (default is "repository_" followed by the <DFXML file> path).

    Parameters:
    <hashdb>       the hash database to insert the imported hashes into
    <DFXML file>   the DFXML file to import hashes from

  export <hashdb> <DFXML file>
    Export hashes from the <hashdb> to a <DFXML file>.

    Parameters:
    <hashdb>       the hash database containing hash values to be exported
    <DFXML file>   the new DFXML file to export hash values into

Database manipulation:
  add <source hashdb> <destination hashdb>
    Copies hashes from the <source hashdb> to the <destination hashdb>.

    Parameters:
```

```
  <source hashdb>       the source hash database to copy hashes from
  <destination hashdb>  the destination hash database to copy hashes into

add_multiple <source hashdb 1> <source hashdb 2> <destination hashdb>
  Perform a union add of <source hashdb 1> and <source hashdb 2>
  into the <destination hashdb>.

  Parameters:
  <source hashdb 1>     a hash database to copy hashes from
  <source hashdb 2>     a second hash database to copy hashes from
  <destination hashdb>  the destination hash database to copy hashes into

add_repository <source hashdb> <destination hashdb> <repository name>
  Copies hashes from the <source hashdb> to the <destination hashdb>
  when the <repository name> matches

  Parameters:
  <source hashdb>       the source hash database to copy hashes from
  <destination hashdb>  the destination hash database to copy hashes into
  <repository name>     the repository name to match when adding hashes

intersect <source hashdb 1> <source hashdb 2> <destination hashdb>
  Copies hashes that are common to both <source hashdb 1> and
  <source hashdb 2> into <destination hashdb>.  Hashes and their sources
  must match.

  Parameters:
  <source hashdb 1>     a hash databases to copy the intersection of
  <source hashdb 2>     a second hash databases to copy the intersection of
  <destination hashdb>  the destination hash database to copy the
                        intersection of exact matches into

intersect_hash <source hashdb 1> <source hashdb 2> <destination hashdb>
  Copies hashes that are common to both <source hashdb 1> and
  <source hashdb 2> into <destination hashdb>.  Hashes match when hash
  values match, even if their associated source repository name and
  filename do not match.

  Parameters:
  <source hashdb 1>     a hash databases to copy the intersection of
  <source hashdb 2>     a second hash databases to copy the intersection of
  <destination hashdb>  the destination hash database to copy the
                        intersection of hashes into

subtract <source hashdb 1> <source hashdb 2> <destination hashdb>
  Copy hashes that are in <souce hashdb 1> and not in <source hashdb 2>
  into <destination hashdb>.  Hashes and their sources must match.

  Parameters:
  <source hashdb 1>     the hash database containing hash values to be
                        added if they are not also in the other database
  <source hashdb 2>     the hash database containing the hash values that
                        will not be added
  <destination hashdb>  the hash database to add the difference of the
                        exact matches into

subtract_hash <source hashdb 1> <source hashdb 2> <destination hashdb>
  Copy hashes that are in <souce hashdb 1> and not in <source hashdb 2>
  into <destination hashdb>.  Hashes match when hash values match, even if
  their associated source repository name and filename do not match.
```

```
    Parameters:
    <source hashdb 1>     the hash database containing hash values to be
                          added if they are not also in the other database
    <source hashdb 2>     the hash database containing the hash values that
                          will not be added
    <destination hashdb>  the hash database to add the difference of the
                          hashes into

  deduplicate <source hashdb> <destination hashdb>
    Copy hashes in <source hashdb> into <destination hashdb> except
    for hashes defined multiple times.

    Parameters:
    <source hashdb>       the hash database to copy hashes from when source
                          hashes appear only once
    <destination hashdb>  the hash database to copy hashes to when source
                          hashes appear only once

Scan services:
  scan <hashdb> <DFXML file>
    Scans the <hashdb> for hashes that match hashes in the <DFXML file>
    and prints out matches.

    Parameters:
    <hashdb>          the file path to the hash database to use as the
                      lookup source
    <DFXML file>      the DFXML file containing hashes to scan for

  scan_hash <hashdb> <hash value>
    Scans the <hashdb> for the specified <hash value> and prints out matches.

    Parameters:
    <hashdb>          the file path to the hash database to use as the
                      lookup source
    <hash value>      the hash value to scan for

  scan_expanded <hashdb> <DFXML file>
    Scans the <hashdb> for hashes that match hashes in the <DFXML file>
    and prints out matches showing all sources.

    Parameters:
    <hashdb>          the file path to the hash database to use as the
                      lookup source
    <DFXML file>      the DFXML file containing hashes to scan for

  scan_expanded_hash <hashdb> <hash value>
    Scans the <hashdb> for the specified <hash value> and prints out matches
    showing all sources.

    Parameters:
    <hashdb>          the file path to the hash database to use as the
                      lookup source
    <hash value>      the hash value to scan for

  server <hashdb> <port number>
    Starts a query server service for <hashdb> at <port number> for
    servicing hashdb queries.

    Parameters:
    <hashdb>       the hash database that the server service will use
    <port number>  the TCP port to make available for clients, for
```

```
                  example '14500'

Statistics:
  size <hashdb>
    Prints out size information for the given <hashdb> database.

    Parameters:
    <hashdb>       the hash database to print size information for

  sources <hashdb>
    Prints out the repository name and filename of where each hash in
    the <hashdb> came from.

    Parameters:
    <hashdb>       the hash database to print all the repository name,
                   filename source information for

  histogram <hashdb>
    Prints out the histogram of hashes for the given <hashdb> database.

    Parameters:
    <hashdb>       the hash database to print the histogram of hashes for

  duplicates <hashdb> <number>
    Prints out the hashes in the given <hashdb> database that are sourced
    the given <number> of times.

    Parameters:
    <hashdb>       the hash database to print duplicate hashes about
    <number>       the requested number of duplicate hashes

  hash_table <hashdb> <repository name> <filename>
    Prints out the hashes and offsets from the given <hashdb> database
    for the <repository name> and <filename> source requested.

    Parameters:
    <hashdb>              the hash database to print duplicate hashes for
    <repository name>     the repository name to match
    <filename >           the filename to match

  expand_identified_blocks <hashdb> <identified blocks file>
    Prints out source information for each hash in <identified blocks file>
    by referencing source information in <hashdb>.  Source information
    includes repository name, filename, and file offset.

    Parameters:
    <hashdb>                  the hash database to use as the lookup source
                              associated with the identified blocks file
    <identified_blocks.txt>   the identified blocks feature file generated
                              by bulk_extractor

  explain_identified_blocks [-m <number>] <hashdb> <identified_blocks.txt>
    Prints source information from the <hashdb> database for hashes in the
    <identified_blocks.txt> file for sources containing hashes that are not
    repeated more than a maximum number of times.

    Options:
    -m <number>               the maximum number of repeats allowed before
                              a hash is dropped (default 20).

    Parameters:
```

```
    <hashdb>                  the hash database to use as the lookup source
                              associated with the identified blocks file
    <identified_blocks.txt>   the identified blocks feature file generated
                              by bulk_extractor

Tuning:
  rebuild_bloom [options] <hashdb>
    Rebuilds the bloom filters in the <hashdb> hash database.

    Options:
    --bloom <state>
      sets bloom filter <state> to enabled | disabled (default enabled)
    --bloom_n <n>
      expected total number <n> of distinct hashes (default 45634027)
    --bloom_kM <k:M>
      number of hash functions <k> and bits per hash <M> (default <k>=3
      and <M>=28 or <M>=value calculated from value in --bloom_n)

    Parameters:
    <hashdb>       the  hash database for which the bloom filters will be
                   rebuilt

  upgrade <hashdb>
    Make hashdb v1.0.0 compatible with hashdb v1.1.0.

    Parameters:
    <hashdb>       the hash database to upgrade

Performance analysis:
  add_random [-r <repository name>] <hashdb> <count>
    Add <count> randomly generated hashes into hash database <hashdb>.
    Writes performance data in the database's log.xml file.

    Options:
    -r, --repository=<repository name>
      The repository name to use for the set of hashes being added.
      (default is "repository_add_random").

    Parameters:
    <hashdb>       the hash database to add randomly generated hashes into
    <count>        the number of randomly generated hashes to add

  scan_random <hashdb> <hashdb copy>
    Scan for random hashes in the <hashdb> and <hashdb copy> databases.
    Writes performance data in the database's log.xml file.

    Parameters:
    <hashdb>       the hash database to scan
    <hashdb copy>  a copy of the hash database to scan

bulk_extractor hashdb scanner:
  bulk_extractor -e hashdb -S hashdb_mode=import -o outdir1 my_image1
    Imports hashes from my_image1 to outdir1/hashdb.hdb

  bulk_extractor -e hashdb -S hashdb_mode=scan
                 -S hashdb_scan_path_or_socket=outdir1/hashdb.hdb
                 -o outdir2 my_image2
    Scans hashes from my_image2 against hashes in outdir1/hashdb.hdb

Examples:
This example uses the md5deep tool to generate cryptographic hashes from
```

```
hash blocks in a file, and is suitable for importing into a hash database
using the hashdb "import" command.  Specifically:
"-p 4096" sets the hash block partition size to 4096 bytes.
"-d" instructs the md5deep tool to produce output in DFXML format.
"my_file" specifies the file that cryptographic hashes will be
generated for.
The output of md5deep is directed to file "my_dfxml_file.xml".
    md5deep -p 4096 -d my_file > my_dfxml_file.xml

This example uses the md5deep tool to generate hashes recursively under
subdirectories, and is suitable for importing into a hash database using
the hashdb "import" command.  Specifically:
"-p 4096" sets the hash block partition size to 4096 bytes.
"-d" instructs the md5deep tool to produce output in DFXML format.
"-r mydir" specifies that hashes will be generated recursively under
directory mydir.
The output of md5deep is directed to file "my_dfxml_file.xml".
    md5deep -p 4096 -d -r my_dir > my_dfxml_file.xml

This example creates a new hash database named my_hashdb.hdb with default
settings:
    hashdb create my_hashdb.hdb

This example imports hashes into hash database my_hashdb.hdb from DFXML input
file my_dfxml_file.xml, categorizing the hashes as sourced from repository
"my repository":
    hashdb import -r "my repository" my_hashdb.hdb my_dfxml_file.xml

This example exports hashes in my_hashdb.hdb to output DFXML file my_dfxml.xml:
    hashdb export my_hashdb my_dfxml.xml

This example adds hashes from hash database my_hashdb1.hdb to hash database
my_hashdb2.hdb:
    hashdb add my_hashdb1.hdb my_hashdb2.hdb

This example performs a database merge by adding my_hashdb1.hdb and my_hashdb2.hdb
into new hash database my_hashdb3.hdb:
    hashdb create my_hashdb3.hdb
    hashdb add_multiple my_hashdb1.hdb my_hashdb2.hdb my_hashdb3.hdb

This example removes hashes in my_hashdb1.hdb from my_hashdb2.hdb:
    hashdb subtract my_hashdb1.hdb my_hashdb2.hdb

This example creates a database without duplicates by copying all hashes
that appear only once in my_hashdb1.hdb into new database my_hashdb2.hdb:
    hashdb create my_hashdb2.hdb
    hashdb deduplicate my_hashdb1.hdb my_hashdb2.hdb

This example rebuilds the Bloom filters for hash database my_hashdb.hdb to
optimize it to work well with 50,000,000 different hash values:
    hashdb rebuild_bloom --bloom_n 50000000 my_hashdb.hdb

This example starts hashdb as a server service for the hash database at
path my_hashdb.hdb at port number "14500":
    hashdb server my_hashdb.hdb 14500

This example searches the hashdb server service available at socket
tcp://localhost:14500 for hashes that match those in DFXML file my_dfxml.xml
and directs output to stdout:
    hashdb scan tcp://localhost:14500 my_dfxml.xml
```

```
This example searches my_hashdb.hdb for hashes that match those in DFXML file
my_dfxml.xml and directs output to stdout:
    hashdb scan my_hashdb.hdb my_dfxml.xml

This example searches my_hashdb.hdb for hashes that match MD5 hash value
d2d95... and directs output to stdout:
    hashdb scan_hash my_hashdb.hdb d2d958b44c481cc41b0121b3b4afae85

This example prints out source metadata of where all hashes in my_hashdb.hdb
came from:
    hashdb sources my_hashdb.hdb

This example prints out size information about the hash database at file
path my_hashdb.hdb:
    hashdb size my_hashdb.hdb

This example prints out statistics about the hash database at file path
my_hashdb.hdb:
    hashdb statistics my_hashdb.hdb

This example prints out duplicate hashes in my_hashdb.hdb that have been
sourced 20 times:
    hashdb duplicates my_hashdb.hdb 20

This example prints out the table of hashes along with source information
for hashes in my_hashdb.hdb:
    hashdb hash_table my_hashdb.hdb

This example uses bulk_extractor to scan for hash values in media image
my_image that match hashes in hash database my_hashdb.hdb, creating output in
feature file my_scan/identified_blocks.txt:
    bulk_extractor -e hashdb -S hashdb_mode=scan
    -S hashdb_scan_path_or_socket=my_hashdb.hdb -o my_scan my_image

This example uses bulk_extractor to scan for hash values in the media image
available at socket tcp://localhost:14500, creating output in feature
file my_scan/identified_blocks.txt:
    bulk_extractor -e hashdb -S hashdb_mode=scan
    -S hashdb_scan_path_or_socket=tcp://localhost:14500 -o my_scan my_image

This example uses bulk_extractor to import hash values from media image
my_image into hash database my_scan/hashdb.hdb:
    bulk_extractor -e hashdb -S hashdb_mode=import -o my_scan my_image

This example creates new hash database my_hashdb.hdb using various tuning
parameters.  Specifically:
"-p 512" specifies that the hash database will contain hashes for data
hashed with a hash block size of 512 bytes.
"-m 2" specifies that when there are duplicate hashes, only the first
two hashes of a duplicate hash value will be copied.
"--bloom enabled" specifies that the Bloom filter is enabled.
"--bloom_n 50000000" specifies that the Bloom filter should be sized to expect
50,000,000 different hash values.
    hashdb create -p 512 -m 2 --bloom enabled --bloom_n 50000000
    my_hashdb.hdb

Using the md5deep tool to generate hash data:
hashdb imports hashes from DFXML files that contain cryptographic
hashes of hash blocks.  These files can be generated using the md5deep tool
or by exporting a hash database using the hashdb "export" command.
When using the md5deep tool to generate hash data, the "-p <partition size>"
```

```
option must be set to the desired hash block size.  This value must match
the hash block size that hashdb expects or else no hashes will be
copied in.  The md5deep tool also requires the "-d" option in order to
instruct md5deep to generate output in DFXML format.  Please see the md5deep
man page.

Using the bulk_extractor hashdb scanner:
The bulk_extractor hashdb scanner provides two capabilities: 1) scanning
a hash database for previously encountered hash values, and 2) importing
block hashes into a new hash database.  Options that control the hashdb
scanner are provided to bulk_extractor using "-S name=value" parameters
when bulk_extractor is invoked.  Please type "bulk_extractor -h" for
information on usage of the hashdb scanner.  Note that the hashdb scanner
is not available unless bulk_extractor has been compiled with hashdb support.

Please see the hashdb Users Manual for further information.
```

C *hashdb* API: hashdb.hpp

```
// Author:  Bruce Allen <bdallen@nps.edu>
// Created: 2/25/2013
//
// The software provided here is released by the Naval Postgraduate
// School, an agency of the U.S. Department of Navy.  The software
// bears no warranty, either expressed or implied. NPS does not assume
// legal liability nor responsibility for a User's use of the software
// or the results of such use.
//
// Please note that within the United States, copyright protection,
// under Section 105 of the United States Code, Title 17, is not
// available for any work of the United States Government and/or for
// any works created by United States Government employees. User
// acknowledges that this software contains work which was created by
// NPS government employees and is therefore in the public domain and
// not subject to copyright.
//
// Released into the public domain on February 25, 2013 by Bruce Allen.

/**
 * \file
 * Header file for the hashdb library.
 */

#ifndef HASHDB_HPP
#define HASHDB_HPP

#include <string>
#include <vector>
#include <stdint.h>

#ifdef HAVE_PTHREAD
#include <pthread.h>
#endif

/**
 * Version of the hashdb library.
 */
extern "C"
const char* hashdb_version();
```

```
// required inside hashdb_t__
class hashdb_manager_t;
class hashdb_changes_t;
class logger_t;
class tcp_client_manager_t;

/**
 * The hashdb library.
 *
 * Note: libhashdb must be compiled to support the same hash type
 * as the hash type provided in the template.
 */
template<typename T>
class hashdb_t__ {
  private:
  enum hashdb_modes_t {HASHDB_NONE,
                       HASHDB_IMPORT,
                       HASHDB_SCAN,
                       HASHDB_SCAN_SOCKET};
  std::string hashdb_dir;
  hashdb_modes_t mode;
  hashdb_manager_t *hashdb_manager;
  hashdb_changes_t *hashdb_changes;
  logger_t *logger;
  tcp_client_manager_t *tcp_client_manager;
  uint32_t block_size;
  uint32_t max_duplicates;

#ifdef HAVE_PTHREAD
  mutable pthread_mutex_t M;  // mutext protecting database access
#else
  mutable int M;              // placeholder
#endif

  public:
  // data structure for one import element
  struct import_element_t {
    T hash;
    std::string repository_name;
    std::string filename;
    uint64_t file_offset;
    import_element_t(const T p_hash,
                     const std::string p_repository_name,
                     const std::string p_filename,
                     uint64_t p_file_offset) :
                           hash(p_hash),
                           repository_name(p_repository_name),
                           filename(p_filename),
                           file_offset(p_file_offset) {
    }
    import_element_t() :
                           hash(),
                           repository_name(),
                           filename(),
                           file_offset(0) {
    }
  };

  /**
   * The import input is an array of import_element_t objects
```

```
   * to be imported into the hash database.
   */
  typedef std::vector<import_element_t> import_input_t;

  /**
   * The scan input is an array of hash values to be scanned for.
   */
  typedef std::vector<T> scan_input_t;

  /**
   * The scan output is an array of pairs of uint32_t index values that
   * index into the input vector, and uint32_t count values, where count
   * indicates the number of source entries that contain this hash value.
   * The scan output does not contain scan responses for hashes
   * that are not found (count=0).
   */
  typedef std::vector<std::pair<uint32_t, uint32_t> > scan_output_t;

  /**
   * Constructor.
   */
  hashdb_t__();

  /**
   * Open for importing, return true else false with error string.
   */
  std::pair<bool, std::string> open_import(const std::string&
      p_hashdb_dir,
                                           uint32_t p_block_size,
                                           uint32_t p_max_duplicates);

  /**
   * Import.
   */
  int import(const import_input_t& import_input);

  /**
   * Import source metadata.
   */
  int import_metadata(const std::string& repository_name,
                      const std::string& filename,
                      uint64_t filesize,
                      T hashdigest);

  /**
   * Open for scanning, return true else false with error string.
   */
  std::pair<bool, std::string> open_scan(const std::string&
      path_or_socket);

  /**
   * Scan.
   */
  int scan(const scan_input_t& scan_input,
           scan_output_t& scan_output) const;

#ifdef HAVE_CXX11
  hashdb_t__(const hashdb_t__& other) = delete;
#else
  // don't use this.
  hashdb_t__(const hashdb_t__& other) __attribute__ ((noreturn));
```

```
#endif

#ifdef HAVE_CXX11
  hashdb_t__& operator=(const hashdb_t__& other) = delete;
#else
  // don't use this.
  hashdb_t__& operator=(const hashdb_t__& other) __attribute__
      ((noreturn));
#endif

  ~hashdb_t__();
};

#endif
```

D bulk_extractor *hashdb* Scanner Usage Options

The **bulk_extractor** hashdb scanner provides two capabilities: 1) scanning a hash database for fragments of previously encountered hash values, and 2) importing block hashes into a new hash database. Options that control the hashdb scanner are provided to **bulk_extractor** using "-S name=value" parameters when **bulk_extractor** is invoked. Available options are:

```
-S hashdb_mode=none    Operational mode [none|import|scan]
     none    - The scanner is active but performs no action.
     import  - Import block hashes.
     scan    - Scan for matching block hashes. (hashdb)
-S hashdb_block_size=4096    Hash block size, in bytes, used to generate hashes (hashdb)
-S hashdb_ignore_empty_blocks=YES    Selects to ignore empty blocks. (hashdb)
-S hashdb_scan_path_or_socket=your_hashdb_directory    File path to a hash database or
   socket to a hashdb server to scan against.  Valid only in scan mode. (hashdb)
-S hashdb_scan_sector_size=512    Selects the scan sector size.  Scans along
   sector boundaries.  Valid only in scan mode. (hashdb)
-S hashdb_scan_max_features=0    The maximum number of features lines to record
   or 0 for no limit.  Valid only in scan mode. (hashdb)
-S hashdb_import_sector_size=4096    Selects the import sector size.  Imports along
   sector boundaries.  Valid only in import mode. (hashdb)
-S hashdb_import_repository_name=default_repository    Sets the repository name to
   attribute the import to.  Valid only in import mode. (hashdb)
-S hashdb_import_max_duplicates=0    The maximum number of duplicates to import
   for a given hash value, or 0 for no limit.  Valid only in import mode. (hashdb)
```