perm.pub/dsi:1wFGhvmv8XZfPx0O5Hya2e9AyXo/2.1
Additional formats and editions available online.

Edition 2.1

Obsolete

Author date: 2024-02-11

Archive date: 2024-02-18

Citation:
E. Castedo Ellerman (2024) "Document Succession Identifiers" perm.pub https://perm.pub/dsi:1wFGhvmv8XZfPx0O5Hya2e9AyXo/2.1

Document Succession Identifiers

E. Castedo Ellerman (castedo@castedo.com)

Abstract

DOCUMENT TYPE: Living Technical Specification

A Document Succession Identifier (DSI) in a bibliographic reference facilitates the long-term retrieval of a cited document from any compatible website. A DSI can refer to either a succession of document snapshots or a specific document snapshot. This lets readers discover document updates while preserving access to earlier snapshots.

This technical specification formally defines the textual format of a DSI and outlines the concept of a document succession.

Alternative Documents

- As of 2023, the website try.perm.pub provides documentation, tutorials, and guides on resources that support Document Succession Identifiers (DSIs).
- For a non-technical overview of DSIs and their purpose, visit Why Publish Digital Successions.

Specification

Document Successions

A document succession contains document snapshots, each a static collection of bytes identifiable by a Git [1] hash or an equivalent Software Hash Identifier (SWHID). A snapshot can be either a file or a directory encoded for compatibility with Git, SWHIDs, and the Software Heritage Archive [2].

Each snapshot in a document succession is assigned at least one fixed edition number. Edition number assignments are immutable. Document successions are updated only by adding new edition numbers.

An edition number can be a single non-negative integer or a multilevel tuple of two to four non-negative integers.

In a document succession, no fixed edition number can be a prefix of another. For example, if `1.2.3` is a fixed edition number, then `1.2` cannot be a fixed edition number. However, a non-fixed edition number can identify a dynamic sequence of editions, such as `1.2` identifying the sequence `1.2.1`, `1.2.2`, and `1.2.3`.

Multilevel edition numbers resemble software package release numbers (for example, software release 2.19.2). Larger integers represent newer editions within the same level of an edition number. This specification does not assign semantic meaning to specific levels in edition numbers.

The integer `0` has a special meaning in edition numbers. An edition number with any zero component is considered *unlisted*. Furthermore, the last component integer of a fixed edition number must not be zero.

Textual Representation of a DSI

The textual representation of a Document Succession Identifier (DSI) consists of a base identifier, which may be optionally followed by a slash and an edition number. The edition number is composed of one to four non-negative integers, each less than one thousand, and separated by periods.

Examples

Base DSI of this specification:
`1wFGhvmv8XZfPx0O5Hya2e9AyXo`

DSI of the first edition:
`1wFGhvmv8XZfPx0O5Hya2e9AyXo/1`

DSI of the first subedition of the first edition:
`1wFGhvmv8XZfPx0O5Hya2e9AyXo/1.1`

Base DSI as a Git Hash

As of 2023, DSIs are implemented using Git [1]. Future implementations may use different hashing mechanisms, as long as the risk of identifier collision remains acceptably low. In a Git-based implementation, the base DSI is calculated from the initial commit of a document succession. Git-compatible software calculates a 20-byte binary hash, which is typically represented as a 40-digit hexadecimal string. For DSIs, this hash is represented by a 27-character string using the standard base64url format (RFC 4648)[3].

Formal Definition in Extended Backus—Naur Form

Note that `N * [ x ]` matches zero to N repetitions of `x`.

```
dsi ::= [ prefix ] base_dsi [ "/" [ edition_number ] ] ;
base_dsi ::= ( 26 * b64u_digit ) b64u_digit27 ;
edition_number ::= int_number ( 3 * [ "." int_number ] ) ;
int_number ::= "0" | pos_dec_digit ( 3 * [ dec_digit ] );
pos_dec_digit := "1" ... "9" ;
dec_digit := "0" | pos_dec_digit;
b64u_digit ::= "A" ... "Z" | "a" ... "z" | dec_digit | "-" | "_" ;
b64u_digit27 ::= "A" | "E" | "I" | "M" | "Q" | "U" | "Y" | "c" |
                 "g" | "k" | "o" | "s" | "w" | "0" | "4" | "8" ;
```

The optional `prefix` is unspecified and is described in the discussion section that follows.

Discussion

Optional DSI Prefix

Users may choose to use a DSI with or without a prefix, depending on the application context. An intuitive choice for a prefix is `dsi:`, which mirrors the acronym "DSI". For added convenience, some websites provide a URL that serves as a DSI prefix.

As of 2023, the Hidos tool supports DSIs both with and without the `dsi:` prefix in its `find` subcommand. For example:

```
$ hidos find dsi:1wFGhvmv8XZfPx0O5Hya2e9AyXo
gh-703611066 https://github.com/digital-successions/
1wFGhvmv8XZfPx0O5Hya2e9AyXo.git
```

As of 2023, the website `perm.pub` supports a URL-based prefix `https://perm.pub/`, as demonstrated in the following example:

```
$ firefox https://perm.pub/1wFGhvmv8XZfPx0O5Hya2e9AyXo
```

Future Extensions

To accommodate future enhancements, there are three methods to extend the textual representation of a DSI:

- Use a character that is neither a slash (`/`) nor one of the 64 base64url characters.
- Vary the number of characters from 27.
- Make the 27th character one of the 48 base64url characters that never appear as the last character in a base64url encoding of 40 bytes (that is, any base64url character other than `A`, `E`, `I`, `M`, `Q`, `U`, `Y`, `c`, `g`, `k`, `o`, `s`, `w`, `0`, `4`, or `8`).

Use of Base64url Over Hexadecimal

The textual identifier uses "base64url" (Base 64 with a URL and filename safe alphabet) as specified in RFC 4648 [3]. Both base64url and hexadecimal have their advantages and disadvantages.

The main downside to base64url is its susceptibility to copy errors when the copying process relies on human sight. Certain fonts make a poor distinction between some characters. For example, some popular sans serif fonts make no visual distinction between capital 'I' and lowercase 'l'.

However, creators of a new document succession are not obligated to use a specific DSI. If a DSI is deemed unsuitable, generating a new one is straightforward.

The main advantage of base64url is its brevity, requiring only 27 characters compared to 40 in hexadecimal. Since DSIs are used in contexts similar to DOIs, a 27-character identifier is more likely to be acceptable, as it is comparable to the length of a long DOI. Additionally, a shorter ID is more suitable for display on mobile devices, reducing the likelihood of truncation, horizontal scrolling, or the need for very small fonts.

The choice of base64url is partly made on the belief that the following technology trends mitigate the copy-by-human-sight issue:

1. Use of hyperlinks, copy-and-paste, and QR codes.
2. Tools that generate websites and PDFs with customizable fonts.
3. Human-to-computer interfaces incorporating features like autocomplete and typo correction to mitigate input errors.

Document Successions Encoded with Git

When a document succession is first created with Git, it begins as an initial Git commit without any document snapshots. A *Document Succession Identifier* is an intrinsic identifier [2] [4] of the initial Git commit. Subsequent Git commits add document snapshots to the document succession. However, the Git tree of each commit does not represent a single document snapshot; instead, it records all snapshots in the succession. The top-level directory contains subdirectories named with non-negative integers. Each subdirectory contains either an entry named `object` or further subdirectories also named with non-negative integers. An entry named `object` encodes a document snapshot, which may be a file (Git blob) or a directory (Git tree). For example, adding a single file as edition 1 results in a directory path `1/object` that corresponds to a Git blob for edition 1.

Digital Signing

For testing purposes, a document succession in Git can be unsigned. However, for public distribution, it must be signed. Digital signatures are applied using SSH signing keys through Git [1]. Each commit in a signed document succession must be signed and contain a `signed_succession` subdirectory that includes an `allowed_signers` file listing the public keys authorized to extend the document succession.

Implementation Choice Rationales

Separation of Git History from Edition History

Representing the history of editions through means other than Git commit history is a deliberate design choice. Git commit history records all Git actions, which can lead to inflexible and complicated non-linear histories. Software Heritage automatically preserves Git commits, increasing the risk that a Git commit history could become an unintended complicated non-linear history. Non-linear Git commit histories and merge commits might be useful in certain scenarios.

Separating edition history from Git commit history also allows for future enhancements, such as retracting specific editions.

Use of Git Tree Paths Instead of Git Tags

In document successions, edition numbers are akin to software release versions, which are typically identified using Git tags. However, this specification adopts a different approach. Edition numbers are recorded with file paths in Git trees rather than Git tags. With this approach, a single latest Git commit captures a complete record of a document succession. This means copying document successions is as easy as copying Git branches. This is especially useful when consolidating records from multiple sources into a single Git repository.

In contrast, software projects, which often include release tags, are copied by cloning the entire repository. Using Git tags for edition numbers would introduce the complexity of keeping a branch and edition number tags in sync, thereby increasing the risk of problems during copying.

While branches in Git repositories are useful for managing document successions, branch names do not constitute a part of the document succession record.

Acknowledgments

Thank you to Valentin Lorentz for raising questions about design choices and pointing out an important shortcoming in how GPG digital signatures were used in the initial implementation of the Hidos library (version 0.3) [5].

Further Reading

- This specification is heavily influenced by the concept of *intrinsic identifier* and related concepts discussed in [2] [4].
- For a discussion on various concepts and proposed terminology regarding persistent identifiers, see [6]. According to this proposed terminology, a DSI is a persistent identifier (PID) that is "frozen" and "waxing" with "intraversioned" and "extraversioned" PIDs depending on the edition number.

Changes

From Edition 1 to 2

- The term "digital succession" has been updated to "document succession."

From Edition 1.2

- References to SSH signing keys have replaced mentions of GPG/PGP signing keys.

References

1. Git — Wikipedia, the free encyclopedia. 2023. Available: https://en.wikipedia.org/w/index.php?title=Git&oldid=1177307938
2. Cosmo RD, Gruenpeter M, Zacchiroli S. Referencing Source Code Artifacts: A Separate Concern in Software Citation. Computing in Science & Engineering. 2020;22: 33–43. doi:10.1109/MCSE.2019.2963148
3. Josefsson S. The Base16, Base32, and Base64 data encodings. RFC Editor; Internet Requests for Comments; RFC Editor; 2006 Oct. doi:10.17487/RFC4648
4. Di Cosmo R, Gruenpeter M, Zacchiroli S. Identifiers for Digital Objects: the Case of Software Source Code Preservation. iPRES 2018 - 15th International Conference on Digital Preservation. Boston, United States; 2018. pp. 1–9. Available: https://hal.archives-ouvertes.fr/hal-01865790
5. Hidos 0.3. 2022. Available: https://archive.softwareheritage.org/swh:1:rev:b963e5d2366724df6e8c34d864a7984ce4a2e1be;origin=https://gitlab.com/perm.pub/hidos
6. Kunze J, Calvert S, DeBarry JD, Hanlon M, Janée G, Sweat S. Persistence Statements: Describing Digital Stickiness. Data Science Journal. 2017;16: 39–. doi:10.5334/dsj-2017-039