The C4 Identification System

Universally Consistent Identification Without Communication

Joshua Kolden
joshua@studiopyxis.com

ETC@USC
Robert Zemeckis Center for Digital Arts
3131 S Figueroa St., RZC 201
Los Angeles, CA 90089

(Updated September 24th, 2015)

ABSTRACT

The Cinema Content Creation Cloud or C4 is a framework for media production using globally distributed resources. A key problem of organizing and managing work across such a wide domain is the accurate identification of data.

There are many identification systems in use today such as file names, URIs, UUIDs, UMIDs and others. However, existing systems focus on a different set of problems, and as such, prove inadequate to the task of unambiguous identification of all assets worldwide regardless of origin, ownership, location, or storage system. These problems include uniqueness, consistency, usability and security for every single digital asset in the world.

The C4 ID system provides an unambiguous, universally unique ID for any file or block of data. However, not only is the C4 ID universally unique, it is also universally consistent. This means that given identical files in two different organizations, both organizations would independently agree on the C4 ID, without the need for a central registry or any other shared information. This allows organizations to communicate with others about assets consistently and unambiguously, while managing assets internally in any way they choose. This can be done without prior agreement or communication of any kind.

This '*agreement without communication*' is an essential feature of C4 IDs and a key differentiator between it and other identification systems. It enables interoperability between human beings, organizations, databases, software applications, and networks, and it is essential to the globally distributed workflows of media production.

This paper discusses the features, implementation details, motivations, and ideas behind the C4 ID system. It compares C4 IDs with other popular identification systems, and provides examples of how to use C4 IDs to communicate about files, coordinate workflows, as well as some additional valuable side effects**,** such as for storage and network optimization.

INTRODUCTION

"There are only two hard problems in Computer Science: cache invalidation and naming things."
Phil Karlton

Identifying data is so common we take it completely for granted, until we run into the limitations of our identification systems. In computer systems it is difficult to think of data identification that isn't framed in the mental model of files and folders. Yet, this way of identifying data is not an inherent property of data or computers. It was an intentionally imposed model designed to help people identify digital data in a familiar way. However, identification by filename and path pose a severe limitation on our ability to communicate about data in an unambiguous way.

Conceptually, identification of digital assets is a simple process. One puts information into a file, names the file, and puts the file into folders for organization. However, this easy-to-understand model breaks down almost immediately as we try to interact with these files. Even as casual users, we run into these kinds of problems all the time.

For example, simply attempting to copy files with identical names into the same folder presents an ambiguity that even modern operating systems are unable to handle without user interaction or loss of data. The problems with filesystems grow exponentially as we try to manage data across organizations. With distributed systems like the Internet, keeping track of specific data becomes effectively impossible.

What is needed is a way to identify data that does not depend on a single person's choice of file names. Even file-naming conventions are inadequate since they are prone to human error, and the larger the organizational scale, the less likely a convention can be enforced.

The first step in creating a universal identification system is to outline the properties of an imaginary ideal identification system, and then incorporate as many of those qualities as is technically and ergonomically possible.

PERFECT IDENTIFICATION

Fundamentally, an ideal universal ID is unique and unambiguous for every file in the world. This includes the files and formats we have, as well as the ones we haven't created yet. The future wouldn't break the perfect ID system. Any block of data should be identifiable, and filenames are already inadequate for this task. In fact, we shouldn't be constrained by the concept of files, at all.

Globally, everyone would agree on the name of a file in the ideal identification system. We could still have names for files that are meaningful within our local context, like a laptop, for example, but if we spoke with people on the other side of the world about files we have in common, everyone would call them by the same ideal names.

We can almost do this on the Internet with the URL infrastructure backed by a central registry, the domain registration system managed by ICANN.[1] However, every file in the world is not identifiable

---

[1] *The Internet Corporation for Assigned Names and Numbers is a nonprofit organization that is responsible for coordinating the maintenance and methodologies of several databases, with unique identifiers, related to the namespaces of the Internet - and thereby, ensuring the network's stable and secure operation. — https://en.wikipedia.org/wiki/ICANN*

by a URL, because not every file is on the Internet, and we don't always have an Internet connection or bandwidth to access our files via http. An ideal system would not need a central registry. It would be able to identify any data anywhere it was encountered, so it would work just as well in total isolation on a computer in a secure facility, as it would on a Backbone-connected web server. Filenames work on an isolated computer, and we'd expect the perfect ID to work at least as well. We might want to talk over the phone about a file or simply copy it between folders without confusing the operating system.

This brings up the issue of security, which is of particular concern since some existing ID mechanisms imbed potentially sensitive information within the IDs themselves. In most identification systems, we must use metadata to help identify the data we want, but the ideal identification system could separate metadata from identity so that changes in metadata would not affect the identification of the data, and with stronger identification, better and clearer associations could be made. Intentional associations would be stronger, while a finer level of control could be exerted over what data needed to be shared. So for someone with both the data and associated metadata, we'd expect the two to be linked together unambiguously, perhaps even more tightly than a filename is linked to the data of a file now. On the other hand, someone with only an ID would know nothing about the data or metadata related to the ID.

There are also practical issues of existing infrastructures based on filename identification to consider. There are a lot of systems that have been constructed to interact with data in the form of files, and while these current systems may not have optimal identification mechanisms, we'd hope the ideal system wouldn't break them since that would have huge cost implications. In other words, the ideal identification system would have IDs that are normal strings of regular characters that could be used as a file name, in a URL, or anywhere we might normally use a file name. This is worth noting since many ID systems include symbols like dash '-', colon ':', forward slash '/', plus '+', and other characters that have special meaning within filesystems and URLs.

Although we typically think of data assets like a video file as the key value, metadata also has significant value. The perfect ID system would allow organizations to retain ownership of metadata separate from the assets, while assets themselves might be shared with other organizations or distributed publicly. In the entertainment industry, in particular, data is often "siloed" within different organizations. An ideal identifier would help link information together between data silos without forcing organizations to provide complete copies of their databases.

Finally, an ideal system would be a system that everyone uses. For that to be true, it would have to be easy. Easy means it's easy to use, it's easy to implement, and it's easy to integrate into existing processes.

From this idealized identification system we can derive the following ten properties of perfect identification. We would expect any good identification system to have some of these properties; an excellent system would have most of them; and the perfect system would have all of them and perhaps others.

1. **It would be safe to use in filenames, URLs, and database records.**

2. **It would be immune to local property changes like filename, path or date.**

3. **It would be format agnostic and able to represent any kind of data.**

4. **It would be unique throughout the world for a given piece of data.**

5. **It would not depend on any external information.**

6. **It would be easily recognized by humans or machines.**
7. **It would be simple to implement in software and easy to use.**
8. **It would not leak sensitive information.**
9. **It would be unchangeable for a given asset.**
10. **The same file would always have the same ID to everyone.**

Of the existing identification systems commonly found in media production today, filenames and URLs have only the first and perhaps seventh property.

UUIDs or GUIDs[2] have only the first 4 properties, and #4 only partially depending on how they are generated.

UMID[3], a SMPTE standard identifier, conspicuously has none of these properties.

MD5 hash as an ID is a considerable improvement over the others, but still only exhibits about half of the properties of an ideal identifier.

C4 IDs have all 10 properties.

C4 ID

C4 IDs are safe to use as filenames and in URLs, or wherever one might normally use a filename, because they use a filename- and URL-safe alphabet that only includes capital letters, lowercase letters, and numbers; no symbols or special characters are required.

C4 IDs are immune to local property changes on files because only the bytes within a file or block of data are considered, not the file name or any other file attributes.

C4 IDs can be used to identify any data, whether it be in a file or an arbitrary block of data, because they are generated from the bytes directly without regard to any data format those bytes represent.

C4 IDs are unique for a given file throughout the world and will remain so for the foreseeable future because they are generated using a Secure Hashing Algorithm. Secure hashing algorithms produce a short unique number for a given block of data of any size similar to a MD5 hash. However, MD5 is not secure because in addition to other issues modern hardware makes it possible to create a document that produce the same MD5 output as a different document[4]. This is known as a 'collission', secure hashing algorithems have no known colissions or any practical way to creat a

---

[2] The term GUID or Globally Unique Identifier usually refers to Microsoft's implementation of the Universally Unique Identifier (UUID) standard.

[3] UMID stands for Unique Material Identifier and is defined by the SMPTE standard. The UMID identification system is described in SMPTE 330M-2004 and revised in SMPTE 330M-2011. These standards documents are available for purchase from SMPTE at https://www.smpte.org/standards.

[4] More information about the security implications of MD5 is available at https://en.wikipedia.org/wiki/MD5#Security.

colission intentionally. Different files regardless of how similar they appear will produce completely different C4 IDs. The ID produced will not be incrementally or predictably different but by all appearances completely random and unique. In addition to this, C4 IDs have an enormous address space. They contain 64 bytes of 'uniqueness' while MD5 by comparison has only 16. To help understand how staggeringly large this address space is, consider that by some estimates there are roughly 2^{266} atoms in the entire universe C4 IDs can uniquely identify 2^{256} of them.

C4 IDs are generated using only the bytes of the file or data being identified. No external information or registry is used, because none is needed to ensure uniqueness and consistency.

C4 IDs are easy to identify by humans and machines alike even out of context because all C4 IDs start with "c4". Other notable attributes are that they are always exactly 90 characters long, do not contain zero "0", the capital letters "O" and "I", or lower case "l" (ell), because these characters can easily be confused with each other in certain fonts.

C4 IDs are easy to implement because there are only two simple steps. First, the very common SHA-512 algorithm is used to generate a 512-bit hash or "message digest" of the data to be identified. Second, the resulting hash is encoded to a simple Base58 character set and prefixed with "c4". These two steps are easily implemented because the SHA-512 algorithm has many open source implementations, and the details of encoding to Base58 are described in pseudocode provided below. Also full open source implementations in various programming languages are freely available online.[5]

C4 IDs are secure and do not leak any sensitive information because they do not contain any metadata about the file being identified or the computer on which the ID was generated. Computer information such as system clock or MAC address is not used in any way when calculating the ID. Each ID has every appearance of being completely random and unique, and the hashing algorithm has been thoroughly cryptographically analyzed, having been widely available for more than a decade. There is no known meaningful cryptographic attack on the SHA-512 algorithm. SHA-512 is a member of the SHA-2 family of secure hashing algorithms, which was published in FIPS[6] PUB 180-4 in 2001 by NIST[7] as a United States federal standard for information processing.

C4 IDs are unchangeable and inseparable from the data they represent. This is due to the fundamental concept of basing the ID on the data itself rather than assigning an ID. All assignment-based identification systems suffer the same weakness of changeability. It is trivial to lose track of a file accidentally by simply changing its name, while it is impossible even to intentionally hide a file from its C4 ID regardless of how it is renamed or moved.

C4 IDs will always be the same for the same file, and will always be different for different files. Two organizations that each have a file with the same C4 ID can be absolutely confident that they have the same file without needing to send the copies to each other for comparison.

---

[5] Documentation and links to source code are available at http://www.cccc.io. Code contributions are welcome and encouraged.

[6] FIPS stands for Federal Information Processing Standards which are publicly announced standards developed by the United States federal government for use in computer systems by non-military government agencies and government contractors. https://en.wikipedia.org/wiki/Federal_Information_Processing_Standards

[7] NIST stands for The National Institute of Standards and Technology, previously known as the National Bureau of Standards, is a measurement standards laboratory. http://www.nist.gov

COST

The C4 ID system is open source, and free to use as is, or incorporated into other software including closed source software. C4 IDs are unique, authoritative, already exist within every digital file ever created, are easy to share, easy to recognize, and integrate well with existing systems. There is a cost for these features even though the software is free, but breaking them down we see that those costs compare quite favorably with current systems. C4 IDs also bring immediate benefits that can more than compensate for the initial costs of adoption.

Many assignment-based IDs have effectively no computational cost since they are assigned and not computed. However, the cost associated with managing data by assigned ID is significant. Whatever savings is gained by not computing a hash on the data, it is immediately lost by the need to communicate and coordinate with a registry, usually remote, which is required in assignment-based identification systems to enforce consistency. Also there are many expensive enterprise-level tools such as digital asset management systems that are primarily used to mitigate the problems created by ephemeral identification like filenames and UUIDs.

A C4 ID's computational cost is primarily within the hashing algorithm. The SHA-512 algorithm used by C4 is one of the highest performance hashing algorithms of any kind, and, on 64-bit hardware, even surpasses other SHA2 algorithms such as SHA-256 as the highest performance secure hashing algorithm. The following table shows a performance comparison between MD5, SHA1, and SHA2.

Algorithm	Megabytes per Second	Cycles Per Byte	Ratio of MD5s performance
MD5	382	6.3	100%
SHA1	192	10.9	50.26%
SHA-512	154	13.6	40.31%
SHA-256	139	15	36.39%

Table. 1. Comparison of hashing performance between MD5, SHA1, and SHA2.

This table does not represent a rigorous test, but simply an anecdotal comparison based on commodity hardware. These tests were run on a 64 bit 2.194 GHz CPU.

As discussed already, MD5 is inadequate for the purposes of globally unique identification due to its weak security properties and inadequate address space, so its relatively high performance is not very useful. SHA1 is an improvement, but not a significant one, given that it is a weaker algorithm compared to the SHA2 family (SHA-256, SHA-512, etc.), and SHA-512 is only slightly more computationally expensive for a significant improvement in security and address space.

All this considered, the computational performance of any hashing algorithm is almost always above the threshold of Input/Output (IO) performance. This means there are no effective differences between these algorithms when taking disk or network IO into account. Any system having better then 130 MB/sec of IO performance, i.e. faster than the throughput of the slowest algorithm, would

almost definitely also have better CPU performance than the commodity CPU used in this test. In addition to that, these numbers represent single core performance. Parallelization can easily improve performance by spreading multiple files across CPU cores in the unlikely event a single core becomes saturated due to overwhelming IO performance.

In the end, hashing performance does not have a significant impact on overall performance and value of any hash based identification system. The cost lies entirely within the fact that the contents of a file must be read to produce an ID, and this cost is the same for MD5 or C4 IDs.

To mitigate this IO cost, we can place C4 ID generation at strategic points within workflows. For example, when IO costs are already being paid for file copying and network communication. Software tools that generate data could help by computing C4 IDs before saving, and hardware manufacturers can compute and report C4 IDs in the course of data transmission and verification.

Processes such as these can reduce the effective cost of C4 ID generation to nearly zero. In addition, value can be extracted almost immediately by adopting the C4 ID system due to some of its unique qualities.

ADOPTION

Using C4 IDs does not call for significant infrastructure change, and needn't supersede any other identification system. Initially C4 IDs can be used for disambiguation, network optimization, and storage deduplication at a net negative cost. In other words adopting the C4 ID system is likely to immediately pay dividends in reduced infrastructure costs that are well in excess of the cost of adoption.

C4 IDs for Data Deduplication

Deduplication is the process of removing unneeded copies of files in file systems. Simply identifying all files in a file system with C4 IDs is enough to reveal which files are identical copies regardless of file names, location or permission. One can easily then decide which copies are needed and which can be removed.

The storage recovered by this process is often quite significant and leads to an immediate real value in addition to the initial acquisition of C4 IDs for all files.

C4 IDs for Archive

With deduplication, often the next question is if files have been archived. Processing archives to compute C4 ID provides unequivocal confirmation of a successful backup. Archives can also be easily evaluated for reliability by re-running C4 ID extraction on an archive and comparing that with the record of C4 IDs that should appear on the archive. Data degradation can be identified and managed during health checks due to the C4 IDs sensitivity to data alteration.

Although not the focus of this paper, the C4 framework has processes for improved archive handling. A useful component of that is the C4 ID tree (*Fig. 1.*). A C4 ID tree is a form of Merkle tree[8], which

---

[8] *In cryptography and computer science, a hash tree or Merkle tree is a tree in which every non-leaf node is labelled with the hash of the labels of its children nodes. Hash trees are useful because they allow efficient and secure verification of the contents of large data structures.*
- https://en.wikipedia.org/wiki/Merkle_tree

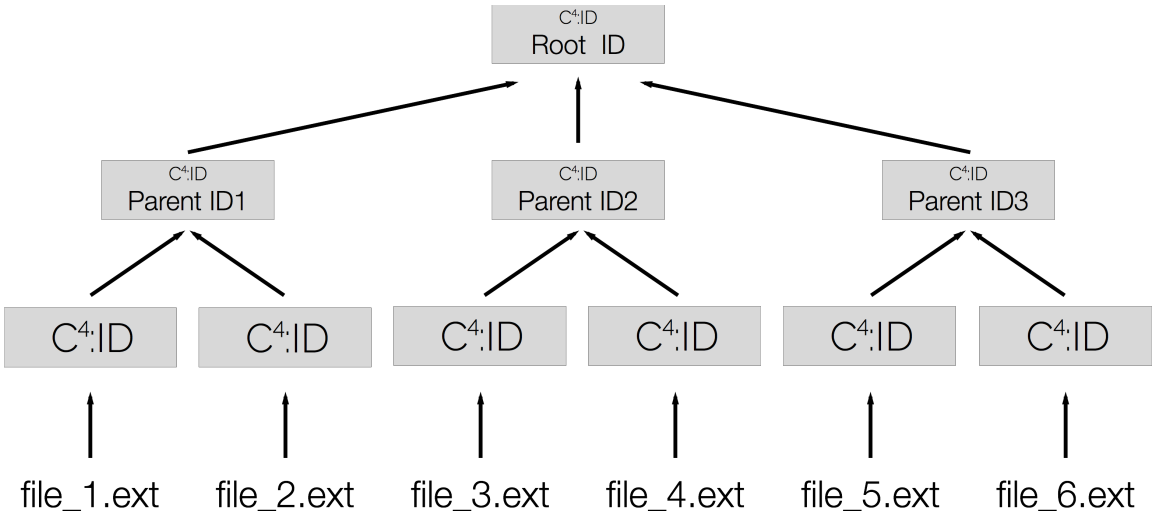

Fig. 1. A C4 ID tree is a hierarchy of C4 IDs used to efficiently identify small changes in files.

is a tree of IDs in which a C4 ID is generated for a string that is the alphabetical listing of all child IDs concatinated togeather end to end (no newline or space). In this way, a single C4 ID efficiently represents a tree of files. If one bit is changed in one file, the root C4 ID will change. One can then examine the children of the root ID to see which of those have changed. Following the changes down the tree makes it easy to quickly identify a single change in a single file without having to compare all file contents or even every single C4 ID one by one.

C4 IDs for Network Optimization.

Just like with data deduplication, C4 IDs can reduce the transmission costs of data across a network. It can do this in at least two important ways.

One, keeping C4 IDs for all files on remote and local systems, and reconfirming IDs after transmission allow systems to detect transmission errors and avoid re-transmitting data that already exists on the remote system.

Second, deterministic computational results provide a way to produce a remote file that does not exist from source data via computation when that is a more cost-effective approach than transmitting data across the wire. For example, source video that can be transcoded to a desired result faster than transmitting the resulting file. If you know the C4 ID of the result, and you have a record of how it is produced from the source, you can reproduce the result and ensure you've done it correctly without comparing the files directly.

Here is an example of a simple YAML[9] object that describes a source file by C4 ID, a process to run on that input, and the C4 ID of the expected result (only known after the process has run at least once).

```
source: c44WhDKyZ9udieWXYC9S2cSmnRyrVQuNMxQxv4rMavaxuWvT4…
process: ffmpeg -i ${source}.avi ${result}.mpg
result: c432UP9Xwc91hJhiHtdpFLK7A9Az647M16WarqgHcYMND1bzn…
```

[9] *YAML is a markup language that is a strict superset of JSON. YAML is chosen here for it's readability over JSON, and XML as well as for it's similarity to the C4 Domain Language.*
- http://yaml.org

C4 IDs for Disambiguation

Since existing identification systems often encounter ambiguous conditions, one can use C4 IDs to provide disambiguation on a case by case basis without regard to how heavily C4 IDs are used overall. For example, if you encounter a file that has the same name as another file, you can compute the C4 IDs on only those two files to immediately tell if they are the same. If you are speaking with someone over the phone about a file, you can both identify the files in question to confirm you both are talking about the same thing.

This "as needed" approach is also a very low impact way to start using and getting comfortable with C4 IDs.

C4 IDs for Version Control

A core quality of C4 IDs is that there is no concept of modifying a file. A modified file is a different file for the purposes of unique identification. This is also exactly what version control systems need to differentiate between versions.

As a file changes, C4 IDs and a copy of the file can be retained as a version. Large projects, with hundreds of files of any size or type, can be easily tracked by version by keeping copies of the files and C4 IDs for each version. The C4 IDs for a version can be stored in a text file or database and used to restore the entire state of a particular version of the project at any time. Branching, tagging and other features of version control systems are done by simply adding tags or branch names to any particular version set of C4 IDs.

C4 IDs for Virtual File Systems

As discussed, filenames and folders do not serve well as unambiguous identification, but these systems do serve well as metadata, or as a way to 'tag' data with meaning. The problem comes from how limiting they can be. Consider the following path:

```
/prod/tdk/tf/020/ele/eye_comp/v23/shadow_pass.0127.dpx
```

The path has meaning to those who understand the naming convention, but very little to anyone else. If you were looking for this file but didn't know this exact list of folders in this exact order you would likely be unable to find it. In reality these folders are more akin to tags. They are intended to associate information with the file, but are unnecessarily reducing the accessibility of the file.

A virtual file system implemented using C4 IDs gives one the flexibility to have a file appear at the above path AND all the combinations of that path.

```
/eye_comp/v23/020/tf/tdk/ele/prod/shadow_pass.0127.dpx
```

Better still, depending on how many different shows at this company have an 'eye_comp' version 23, perhaps only the first few tags are adequate to uniquely identify that particular file.

```
/eye_comp/v23/shadow_pass.0127.dpx
```

This kind of system is easy to implement with C4 IDs using normal file system tools like symbolic links. You can do this by keeping all files by their C4 ID in a flat or very shallow path, then use symbolic links to create many alternate views from the regular file system.

```
/
 c4/
    c4D1bzn…
    c46Warq…
    …
 prod/tdk/tf/020/ele/eye_comp/v23/shadow_pass.0127.dpx -> /c4/c4D1bzn…
 eye_comp/v23/020/tf/tdk/ele/prod/shadow_pass.0127.dpx -> /c4/c4D1bzn…
 eye_comp/v23/shadow_pass.0127.dpx                     -> /c4/c4D1bzn…
 …
```

A more robust system could dynamically construct a filesystem based on tags or even innate file attributes like resolution or file type. This gets to the core of what we actually want from file systems: we want to associate these attributes and descriptions with our files, but we don't want to be limited by these descriptions to a single way of accessing files. Many points of view exist within an organization and each point of view has different priorities and different metadata that is considered relevant. While a visual effects person might consider element name, frame number, and sequence label most relevant, a producer might consider take, actor name, and scene number more appropriate.

Having a database with all of this information pointing to the unambiguous C4 IDs gives everyone access to the same files from whatever point of view they choose. Automated tools can always access files from a fixed predictable path, while users may use different paths at different times for different reasons. This is a much better use of the file system concept that enables rather than restricts discoverability.

C4 IDs for Relationships

Once files can be differentiated unambiguously, we have the ability to establish immutable relationships between those files. Just as a C4 ID immutably identifies a specific file, relationships between files can also be defined immutably. The C4 domain language includes the ability to describe the connections between files, but a similar mechanism is easy to accomplish with simple data structures.

Similar to the process relationship example shown in YAML above, one can describe a set of files in a folder.

```
type: folder
name: joshua
path: /home/joshua
files:
  - name: .bashrc
    id: c458591YrKt...
  - name: .bash_profile
    id: c45FQJpSmoh...
```

Or similarly a shot composed of frames:

```
name: shot_1
timecode:
  start: 01:03:12:18
  end: 01:05:07:12
  frames:
    - c4a6iQxPP...
    - c4dpFLK7A...
    - c4wc91hJh...
    ...
```

The C4 ID of this data structure becomes the immutable pointer to this relationship. Retaining a relationship document anywhere ensures the relationships are maintained regardless of the actual disposition of files in storage. For example, perhaps it takes time to transfer a large file across a network, but a network of relationships points to a proxy that can be viewed while waiting for the larger file to arrive.

Consider the following set of documents:

```
---
c43gUUHy9f…:
  raw:
    format: raw
    size: 423MB
    filename: "shot.raw"
---
c4YX1uibzw...:
  dpx_files:
    source: c43gUUHy9f...
    process: ffmpeg -i ${source} … ${result}.%04d.dpx
    result:
      - c4a6iQxPP...
      - c4dpFLK7A...
      - c4wc91hJh...
      ...
---
c438DqXCQt...:
  proxy:
    filename: proxy.mv4
    source: c4YX1uibzw...
    process: ffmpeg -i ${source}.%04d.dpx … ${filename}
```

If a search using metadata returns the first record, we learn we are looking for a file identified by C4 ID `c43gUUHy9f….` But say that file is not available on the local machine. A search with that ID (`c43gUUHy9f…`) reveals a relationship record `c4YX1uibzw…` a set of dpx files, and from this we find `c438DqXCQt…` a local proxy of the larger raw file. This sequence of relationships is easily described and easily discovered. Relationship records can be created by a person or automation, they can also be created at any time with no knowledge of the other relationships. The resulting network of relationships is the same.

Even the infrastructure upon which this is implemented such as databases or storage and search mechanisms are all unimportant. Due to the immutability property of C4 IDs these relationships retain their meaning even if the files that described them are organized completely randomly on a file system. The infrastructure used to store and access this data can improve the performance, but it is not required to maintain the consistency of the relationships. Relationship records could just as easily be files on disk, SQL/NOSQL database records, graph database nodes, or notes on paper. Once the C4 ID of the relationship is defined, that ID definitively indicates the relationship between the assets described within. In turn, the C4 ID of any member of a relationship points unambiguously to the relationship.

While a given relationship document is definitive and immutable with respect to the relationships and files described, any given file can be a member of an unlimited number of relationships. This results in constrained well-defined relationships between files, but unlimited flexibility for how files are interpreted in the context of other relationships. Just because a file is a frame in a sequence container and called 'frame.0042.exr", does not mean it is not also the parent of a transcoded jpg file in your Documents folder named "hero_frame_for_poster.jpg". Without C4 IDs, one must choose one or the other. With C4 IDs, one can construct a tree of relationships that could conceivably include every single file used in the making of a film.

The result of defining relationships in this simple and unambiguous way is a self-evident self-organizing structure. This improves the recoverability of this information because no special knowledge or tools are required to enforce these structures. This is particularly valuable in archival applications where this self-evident, system independent approach is more stable over very long periods of otherwise rapid technological change.

Think of it like the black-and-white film of digital archive. Black-and-white film has been a fantastic archival format for more than a hundred years because of stable chemistry and the simplicity of holding a strip of film up to a light to know what it is. C4 IDs are the stable chemistry, and opening a relationship in a text editor is akin to holding it up to the light.

C4 IDs for Interoperability

One of the key goals of the Cinema Content Creation Cloud is to facilitate interoperability between organizations. By referring to files and data using C4 IDs, different organizations can communicate effectively while still remaining in control of their own data and resources. Even if an organization is willing to share a full copy of its data to partners, it's often not technically possible to do so.

Globally distributed resources implies shifting authority. Authority, in this context, means the location with the most current and correct information. Imagine a remote production unit shooting on location. This unit will be the authority of what has been shot, what changes have been made on set, and other production related information. A legal department elsewhere would be the authority with respect to the rights and clearances that have been completed, while a previs company would be the authority of what versions of previs are current, etc.

Here authority is not a title, it is a real world constraint that is usually a factor of proximity. For this reason it is necessary for any interoperability solution to accommodate or better enhance the ability to get information from the authority as needed (*pull*) rather than rely on a constant synchronization process. It can be useful to be notified that a remote update occurred, but it's generally better to get the latest from the authority as needed. Any local copy is guaranteed to be occasionally out of sync and so can never truly be authoritative. However, without C4 IDs, data must be managed locally to remain connected and consistent, and this requires aggressive synchronization.

C4 IDs make it much easer to architect solutions to these distributed information problems. Not only do C4 IDs make it obvious what information has changed and what information is the same, C4 IDs become the 'pivot point' between disconnected data sources and allow connections to be made between data sources reliably, regardless of location or type of database.

If one organization has a file they want to know more about, they can search any database they have access to with that file's C4 ID and retrieve any related data. Conversely, searching databases with metadata can lead to the related C4 IDs. Without C4 IDs, it is completely unclear and unreliable what information goes together from different data sources. With C4 IDs, it's trivial and unambiguous, and requires no prior agreement or coordination.

Implementation

The base 58 character set used for C4 IDs was originally created at Flickr[10] for URL shorting and readability. The encoding is similar to the common base 64 encoding but without the need for special characters ('/','_','+','='). Also confusingly similar characters are removed ('0','O', 'I', 'l'). The complete list of valid C4 ID characters numbers 0 through 57:

```
123456789abcdefghijkmnopqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXY
```

The C4 ID is easy to compute and validate. The steps are to compute a SHA-512 hash of the data, convert the result to base 58 given the character set below, pad with zero if needed and prefix "c4". The following pseudocode details how this is done.

[10] *https://www.flickr.com/groups/api/discuss/72157616713786392*

Pseudocode[11]

```
charset     = "123456789abcdefghijkmnopqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ"
base58      = 58   // The encoding basis
c4id_length = 90   // The guaranteed length
zero        = '1'  // '0' is not in the C4 ID alphabet so '1' is zero

string c4id(data)
{
    // Compute the SHA-512 hash.
    // The value is a very large unsigned int.
  BigUInt hash = Sha512(data)

  string id_string = “”

    // loop over the hash dividing each time until zero
  while hash != 0
  {
    BigUInt modulo

    hash = BigUInt::devide_with_modulo(hash, base58, modulo)

      // Append the character to the beginning of the string
    id_string = charset[modulo] + id_string
  }

    // We pad with zero if the id is less then 90 total characters
  int length_difference = c4id_length - 2 - length_of(id_string)

  string padding = repeat(zero, length_difference)

    //  "c4" + any padding + the encoded hash is the result
  return "c4" + padding + id_string
}
```

[11] *The pusdocode has been updated in theis draft. The first release of this whitepaper had the C4 ID string being assembled inncorrectly in left to right order. At the end of the while loop the inccorect code in the previous version was “id_string = id_string + charset[modulo]”. Reference implementaions where not effected.*

Acknowledgements

A great deal of credit for this paper goes to Erik Weaver, Abi Corbin, Daniel De La Rosa, Drew Diamond, Edie Meadows, Mat Ryer and Diann Benti. Speical thanks goes to Richard Goedeken of FotoKem for catching the error in the psudocode. I'd like to thank them for their instrumental advice and editorial support. I'd also like thank the various organizations who've directly or indirectly contributed research and technology in the advancement of universal identification, and in particular SMPTE for providing free access to the UMID standard for this paper. The C4 framework, this paper and associated open source software is made possible by the endorsement of the ETC and its major studio partners, as well as by their ongoing efforts to encourage the development of new and innovative cinematic technologies.