

A Fingerprint Method for Scientific Data Verification¹

Version: 09/14/2009

Micah Altman
Institute for Quantitative Social Science, Harvard University
1737 Cambridge St. # 325
Cambridge, MA 02138

Micah_Altman@harvard.edu

¹ Originally published in *Advances in Systems, Computing Sciences and Software Engineering*, 2008. (Proceedings of the International Conference on Systems, Computing Sciences and Software Engineering 2007), Springer Verlag. This version contains clarifications and comments developed in conjunction with the 09/07/2009 DVN 2.0 software release. These changes are marked with change tracking indicators to distinguish them from the original article as published.

Abstract - This article discusses an algorithm (called “UNF”) for verifying digital data matrices. This algorithm is now used in a number of software packages and digital library projects. We discuss the details of the algorithm, and offer an extension for normalization of time and duration data.

INTRODUCTION

In digital library systems, migrating digital objects among formats is necessary both for effective use and preservation. However, a consequence of such reformatting is that we are often required to re-verify the intellectual content of the object, in order to avoid data loss and misinterpretation.

The Universal Numeric Fingerprints (UNF’s) is an algorithmic tool used to verify that a data matrix or related digital object that is produced in one software environment and/or format has been correctly interpreted when moved to a different environment and/or format.

In outline, the algorithm uses three steps to compute a fingerprint: First, an approximation algorithm is used to compute a reduced-fidelity version of the target digital object. Second, this reduced version is put into a normalized serial form. Third, a hash function is used to compute a unique fingerprint from the resulting serialized object.

While these steps could in theory be applied to any digital object, in practice it is necessary to create a specific normal form and approximation method must be defined for each type of object to be processed. The UNF implementation is specifically tailored for use with scientific data. We describe the approach, implementation, and algorithmic details below.

RELATED APPROACHES

The UNF approach shares some similarities to some published approaches to audio and video fingerprinting [1,2] The UNF algorithm however is specifically tailored to scientific and statistical data vectors. Audio and video fingerprinting methods typically rely on type-specific feature extractions, and are not directly applicable to scientific databases. Furthermore, unlike audio and video fingerprints, which typically compute a long sequence, UNF’s use a more compact representation, suitable for use in scholarly citations.

Other approaches have been developed that attempt to verify that one object is a derivative of another object, regardless of file format. These methods operate through insertion or alteration of data in unused or unnoticed portions of the object, forming a digital watermark. Research into digital watermarks have produced algorithms that are designed to be robust to lossy transformations of the object. And hence some types of image objects can be identified as a derivative of another even when the derivative is manifested in a different file format. (For a survey see [3].)

Watermarks have significant shortcomings when used to establish the semantic equivalence of two arbitrary digital

objects. Watermarking algorithms cannot be used to establish that two independently created objects are semantically equivalent, since these will not share the same watermark. Conversely, two objects could have identical watermark information added, but contain completely different semantic content. Nor can watermarks be used to verify that a derivative is identical to a watermarked digital object, if the derivative was created from the original digital object before the watermark was applied to that original digital object. Furthermore, watermarks are not practical for some objects, such as numeric data and source code files, where the alterations created by the watermarking process tend to alter the semantic content of the digital object.

IMPLEMENTATION AND USE

The UNF was developed in the course of research into improving the numerical accuracy of statistical software [4]. It was incorporates type-specific algorithms for scientific data such as numeric vectors, matrices, and related objects.

Where possible, UNF algorithms are implemented as plug-ins for the software applications being used to manipulate or analyze the digital objects of interest. This strategy is used so that the UNF can be computed directly from the internal data structure used by the software product performing the data display and manipulation. This allows the UNF to be used not just to verify format migration, but also to verify that the data has been correctly interpreted by a software application. (Some of the errors we frequently encountered in [4], and which the UNF detects, included: omitted records, omitted variables, recoding of missing values, and loss of numeric precision.)

Furthermore, this implement strategy obviates the need to write specific parsers for each data format – since the host software application parses the object before calculating UNF. Instead, we need only supply standard interfaces for computing UNF’s of vectors, matrices, and other related structures.

The general algorithm was first described in [4] along with a sample implementation. Version 3 was the first version to be implemented in publicly available software: The UNF package for R was made available through the Comprehensive R Archive Network (CRAN), a code archive developed part of the R Project [5]. (This version of the software also introduced the name “Universal Numeric Fingerprint”.)

UNF plug-ins are currently available for the statistical software packages “R”, “SAS” and “Stata”; along with a stand-alone application, and C++ libraries. The UNF technology has been incorporated in a number of other digital library and preservation practices: UNF’s are used to support format migration in the “Virtual Data Center” (VDC) digital library system for federated data sharing. [6] The successor of the VDC, the Dataverse Network Project (DVN) [7] which

provides digital library software and virtual-archiving services, uses UNF's for data verification.

Both systems now generate scholarly citations for all data collections, based on the standard for the citation of quantitative data developed in [8]. In this citation standard, the UNF is an integral part of the citations, and is used to verify that a given data collection matches the version cited. The citation's combination of a global unique identifier, UNF, and bridge service URL combine to support permanence, verifiability, and accessibility.

For use in citations to data, when displayed, the UNF is formatted so that it is self-identifying and can easily be printed. An example of a printed UNF is

UNF:3:ZNR114053UZq389x0Bffg?==, where UNF: identifies the rest of the string as a UNF, :3 means that the fingerprint uses version 3 of the UNF and hash algorithm, and everything after the next colon is the actual fingerprint. For a particular algorithm and number of significant digits, the fingerprint is always the same length. Thus, the UNF includes enough self-identifying information so that the algorithm used may be updated to newer versions over time without disturbing old citations.

UNF's are also used by the The Data Preservation Alliance for the Social Sciences (Data-PASS), and . This is a partnership of six major U.S. archival institutions collaborating to acquire and preserve data at-risk of being lost to the research community. To support this goal, Data-PASS developed shared technical infrastructure and best practices for metadata, confidentiality, security, processing and other elements of the archival management process. UNF's are used for citation and verification in the Data-PASS shared catalog infrastructure, and are part of the best practices for metadata identified by the alliance. (See [9,10]).

In these systems, standards, and practices, UNF's are used as a replacement for or supplement to file-level cryptographic hashes and summary statistics. (In a sense, the UNF is a uniquely tailored summary statistic for the entire object.) Like a cryptographic hash, the UNF detects changes to the data object. Unlike a standard hash, it is sensitive only to semantically important changes – changes of format or insignificant changes of precision will not yield differing UNF's.

ALGORITHMIC CONSIDERATIONS

To summarize, the abstract algorithm consists of the following steps: An approximation algorithm is used to compute the approximated semantic content of the digital object. This approximated content is then put into a normalized form. A hash function is used to compute a unique fingerprint for the resulting normalized, approximated object, and the hash is stored. When the object is reloaded into the same or another application, this process is repeated, and the value generated at load time is compared to the stored value. These steps are discussed in more detail in the remainder of this section.

The first part of what is needed is a normalization function $f()$ that maps each sequence of numbers (or more commonly, blocks of bits) to a single value:

$$f \{i_0, i_1, \dots, i_n\} \rightarrow c \quad (1)$$

To verify the data, we would need to compute f once when initially creating the matrix. Then recompute it after the data has been read into our statistics package. For robust verification, we should choose $f()$ such that small changes to the sequence are likely to yield different values in $f()$.

Normalization of objects alone, while it can be used as a basis of establishing identity across formats in limited cases, is inapplicable when reformatting of the object changes the precision, accuracy, or level of detail of an object in trivial ways. This is a well known issue in video and audio formats, in reformatting complex text documents, and surprisingly occurs commonly even in reformatting purely numerical databases.

Any type-specific approximation function $A()$, may be employed.² This approximation process, $A()$, accepts as input a digital object, O , of specified type, and an approximation-level parameter, k .

$A()$ should be chosen to capture *semantic* of approximation.³ That is $O' = A(O, k)$, then O' should be *semantically* close to O , since the UNF does not otherwise make direct use of the semantic content of the object.

Two other conditions on $A()$ are notable:

1. [Monotonicity] if $k < k_0$ then for some measure of semantic distance, if $A(O, k_j) \triangleleft A(O', k_j)$ then $A(O, k_i) \triangleleft A(O', k_i)$, $j > i$
2. [Nesting] If $k \geq k_0$ then $A(A(O, k), k_0) = A(O, k_0)$

When $A()$ satisfies the monotonicity condition, greater values of k yield less accurate approximations. When $A()$ satisfies the nesting condition, UNF's can then be used to determine if two derivative objects O' and O'' are approximations of an object O , without knowing the UNF of O . To do this, generate a sequence of UNF's for both objects, at increasing levels of approximation. If the resulting UNF's match for a particular approximation level, the two objects are approximately semantically identical. (Alternatively, if the conditions on $A()$ above are not satisfied, then it is possible that no UNF computed on O' will match the UNF computed on O'' at any approximation level.)

² For audio and video type-specific approximation is used, such as decimation, spatial or frequency downsampling, and/or numerical cutoff filtering. For examples of decimation, downsampling and cutoff algorithms see: [11], [12].

³ Note that this does not necessarily imply the inverse, that for two semantically similar objects, $A(O, k) = A(O', k)$ for some k .

TYPE-SPECIFIC NORMALIZATIONS AND APPROXIMATIONS FOR NUMERIC VECTORS

Versions three and four of the UNF algorithm operate on data-vectors⁴ and involve the following steps:

1. Each element in the numeric vector is rounded to k significant digits, using the IEEE 754 'round toward zero' rounding mode. The default value of k is 7, the maximum expressible in single-precision floating point calculations.
2. Each element is then converted to a character string. Unless the element is missing or not a finite value, it is represented in exponential notation, in which non-informational zeros are omitted. This notation is constructed such that if step 1 satisfies the nesting above, the resulting character string will also satisfy the nesting property.

For example, the number pi, at five digits, is represented as "-3.1415e+" and the number 300 is represented as the string "+3.e+2".

Specifically, this notation comprises:

- (a) A sign character in {+, -}
 - (b) A single leading period.
 - (c) A decimal point, represented by a period character: '.'
 - (d) Up to $k-1$ digits following the decimal, comprised of the remaining $k-1$ digits of the number, omitting trailing zeros.
 - (e) A lower case 'e'
 - (f) A sign character.
 - (g) The digits of the exponent, omitting trailing zeros.
3. If the element is missing it is represented as a string of two null characters. If the element is a IEEE 754 non-finite floating point special value, it is represented as the signed lower-case IEEE minimal printable equivalents (i.e., +inf, -inf, +nan).
 4. Character strings representing non-missing values are terminated with a POSIX end-of-line character (i.e., a [Unicode Line Feed, U+000A](#)).
 5. Each character string is encoded in a Unicode bit encoding. (see [13]) (Versions 1-4 use UTF-32BE versions 4.1 and greater uses UTF-8 for increased performance.)

⁴ A vector is defined as an ordered sequence of values of uniform type. Vectors may include missing values.

6. The vector of character strings is combined into a single sequence, with each character string (including the final string) followed by a null byte. A hash is computed on the resulting sequence. Version 4 uses SHA256 [14] as the hashing algorithm. (Versions one through three use a 64-bit checksum, a 64-bit CRC, and MD5, respectively, as the hashing algorithm.)
7. The resulting hash is then base64 encoded (see [15]), using big-endian byte-order, for printing.

ALGORITHMIC VARIATIONS

Character vectors. Vectors of character values are treated similarly: In step 1, each character string is converted to Unicode Normal Form C, and then truncated to k characters rather than rounded to k digits (The default k for characters is 128.) Step 2 is skipped. Step 3 through 7 are the same.

Mixed vectors. UNF's can be computed on vectors of mixed type by normalizing each element as per its type, and then computing the hash on the resulting byte stream.

Unordered sets. To compute a UNF for an unordered set we convert it to a vector by sorting the approximated elements as follows: (1) First apply the type specific approximation to each element in the vector. (2) Second, sort the approximated elements in POSIX order. (3) Third, apply type specific normalization to each element. (4) Compute the hash.

Composite objects. UNF's can be computed on composite objects that comprise a hierarchy of other objects, by computing a UNF of the UNF of each component recursively. This proceeds as follows:

1. Form a vector of mixed primitive values, by iterating across each component and transforming it according to its type:
 - (a) If the component is a single primitive type, convert it to its approximate canonical representation as above, but do not compute a UNF.
 - (b) If the component is a vector or set, compute its UNF as above.
 - (c) If the component is a compound object, compute a UNF for it, using steps 1-3.
2. For each UNF in the resulting vector, convert it to a base-64 encoded character string.
3. If there is no explicitly intrinsically defined ordering to the object components, sort the resulting values using a POSIX locale sort order.
4. Compute the UNF of the resulting vector of character strings.

For example, the UNF's for multiple variables can be combined to form a UNF for an entire data frame, and UNF's for a set of data frames can be combined to form a single UNF representing an entire research study. Specifically, UNF's any hierarchical composite objects: For simplicity in the self-documenting representation, one should use a consistent

Micah Altman 8/31/09 12:11 PM

Deleted: separated

Micah Altman 8/31/09 12:11 PM

Deleted: POSIX end-of-line characters and a

Micah Altman 9/2/09 11:42 AM

Deleted: three

version and level of precision across the individual UNF's being combined.

EXTENSIONS IN VERSION 5

In this section we introduce improvements to the current UNF algorithm, which together comprise version 5 of the algorithm. To summarize briefly, these changes comprise changes to the following steps in the algorithm:

- Allowing the hash to be truncated after computing.
- Providing a new default rounding mode, rounding to significant digits.
- Normalization forms for date, time, duration, bitstring, and logical values.
- Extensions to the printable form to compactly indicate any non-default options used.

Replacement Hash Function

Version three of the UNF used the MD5 algorithm in its digest step. This produced a relatively manageable printable hash, however the algorithm has since been found to have significant flaws [16]. Version 4 of the UNF algorithm replaced MD5 with SHA256. For automated applications, this was a desirable trade-off: a great increase in security was gained by a minor decrease in performance. However, when printing citations, the length of the SHA256 hash may have a significant impact on readability and usability. Version 5 of the UNF strikes a more balanced trade-off among security, efficiency, and human usability by allowing truncation of the SHA256 using the method described in [17]:

- A full hash (in this case SHA256) is computed.
- To truncate to N bits, the N leftmost (most significant) are used, the least significant $256-N$ are discarded

Unlike the MD5 algorithm, truncated SHA-256 is not subject to any known attacks. (The discovery of near-collisions in SHA-256 would cast doubt on the security of the truncated hash. None so far have been reported.) Finally, SHA-256 always computes a 256 bit hash internally for security, but this resulting can be truncated to either 192, 196 or 128 bits for printing and storage. This means that truncated hashes may be compared to their untruncated counterparts to determine that the truncated hash represents the same object.⁵

Thus a truncated SHA256 yields a substantial benefit for human usability in printed citations, and significantly improves security, at the cost of a small reduction in performance relative to MD5.

Alternative Approximation Method for Numeric Vectors

⁵ Note that this method of truncating SHA256 is not identical to the official SHA224 implementation, since the official SHA224 uses different initial values.

Rounding toward zero ensures nesting holds, but can produce more rounding error than rounding toward nearest. The maximum log relative error (LRE) for the former is $(\text{digits}-1)$ while the maximum LRE for the latter is digits . Hence, you may wish to use one more significant digit when computing UNF's than when reporting rounding significant digits for presentation or storage. More important, rounding toward zero also does not guarantee inverse semantic similarity, since two numbers that are close numerically can yield different approximate values. In particular, numeric imprecision that occurs near whole-number boundaries can yield different approximations.

Version 5 by default, uses rounding to k significant digits. This proceeds by rescaling (dividing by a power of 10) the value so that the k th digit is immediately to the left of the decimal, applying IEEE round to nearest rounding, and rescaling by the original scale factor. This violates nesting, but assures that two values numerically close will have the same approximate representation, which makes it more robust to numeric imprecision. As an option, the previous rounding mode may still be used.

Normalization of Date, Time, Duration, Bitfield and Logical Values

Previous implementations of the UNF algorithm addressed vectors of numeric and character data. While dates and times are can be represented in either form, additional normalization is required to ensure that semantically equivalent vectors of other types yield the same UNF. This section describes additional normalization forms.⁶

In version 5, *boolean values* are represented by converting to one of three numeric values: {0,1,missing}. No rounding is applied.

Bit fields are normalized by: (1) converting to big endian-form, (2) truncating all leading empty bits, (3) aligning to a byte boundary by re-padding with leading zero bits, (4) base-64 encoding to form a character string representation. No

⁶ ASN.1 [19] defines a 'distinguished' encoding form for sets and sequences of values which resembles somewhat the UNF normalization representation for numbers (which was developed independently). However we did not find ASN normalization forms suitable for scientific data representation for a number of specific and general reasons: ASN.1 does not have a way of representing missing values nor infinite values. Nor are the ASN normalization forms not designed to be compatible with partial value representation, such as partial dates. More important, ASN.1 normalization forms also are made on the basis of format rather than semantic content (e.g. Integers and reals have different normalizations type, but may represent the same value). Generally, ASN.1 is a complex standard most suitable for systems data exchange, it does not address construction of an approximate form of an object, nor the construction and representation of fingerprints.

Micah Altman 9/2/09 11:38 AM

Deleted:

Micah Altman 8/31/09 12:29 PM

Comment: After conversion, the UNF is computed according to the rule for numeric vectors/sets as above.

Micah Altman 8/31/09 12:27 PM

Comment: Design note: the padding is for robustness, but involves an implicit assumption is that the length of the bitfield is semantically either implied, fixed, or unimportant.

rounding is applied. Missing values are represented by three null bytes.

Time, date and duration normalization is more complex. Time and date standards, such as ISO 8601 [18] are helpful in determining whether two elements are semantically equivalent, but are not sufficient for use in the UNF process since they fail to define a single canonical representation. (ISO 8601 permits many representations of time and date values, including variation in separators, time zone representation, fractional times in different bases, week dates and ordinal dates. Thus it is possible to represent the same moment in time in many different ways, and still be compliant with the standard).

We now describe a unique character-based representation for times, calendar dates, and durations. This representation can be used in step 2 of the UNF algorithm described in the previous section, in order to form UNF's of vectors of time/date elements. This normalization method is essentially a single unambiguous representation selected from the many described in the ISO 8601 standard, and is thus compliant with the standard, and can easily be implemented using standard libraries for manipulating ISO date/time values.

To form a UNF for a data element comprising a calendar date the date is converted to a character string in the following form: "YYYY-MM-DD". This comprises zero-padded numbers representing year, month (in [01,12]) and day. Partial (imprecise) dates where only the year, or the year and month are formed as "YYYY" and "YYYY-MM" are permitted.

The UNF method for representing time is based on the in the ISO 8601 extended format: "hh:mm:ss.ffff". The quantity "ffff" represents fractions of a second, it must contain no trailing (non-significant) zeroes, and must be omitted altogether if valued at zero (thus "12:01:01.0" is not legal for UNF's). Other fractional representations (such as fractional minutes and hours) are not permitted in the UNF representation.

The UNF representation permits times in one time zone only. If the time zone of the observation is known, the time value must be converted to the UTC time zone and a "Z" must be appended to the time representation: "02:01:04.003Z".

Elements that comprise a combined date and time are formed by concatenating the (full) date representation, "T", and the time representation, as in: "2004-12-28T02:01:04.003Z". Partial date representations must not be used for combined date/time values.

Type-specific approximation proceeds by deleting entire components of the time, date, or combined time/date. Components should be deleted in the following order: fractional seconds component, seconds, minutes, hours, day, time zone indicator (if any), and month.

Durations are represented by two date-time values, formatted as above, separated by a solidus (" / "), where each [n] represents the number of years, months, dates, hours, minutes, and seconds (respectively) in the duration. Fractional values of seconds (only) are permitted in the form of "nnn.ffff". Where n=0, the "0" is required. All other leading and trailing zeroes, fractional hours and minutes, and

truncated values are prohibited. Durations may be used only where the actual start time is not known, otherwise a time interval must be used.

Extensions to the Printed Representation

Version 5 extends the printed representation to include other ways of documenting alternate approximation levels, algorithmic options. This extension is forwards compatible and allows for a more compact printable representation when only a subset of approximation levels deviate from their defaults. These fields are required only where the approximation level, or algorithm parameter used differs from the default for that UNF version.

The following additional representations of approximation levels are permitted: (1) "###,###,###", (where ### is an integer value) indicating non-default levels of approximation for numeric, character, and time values (2) any combination and ordering of 'N###', 'X###', 'T###', 'H###', 'R###', separated by commas, indicating approximation values for numeric, character, time values, the hashing truncation, and the numeric rounding mode. Values associated with H may be in {256,192,196,128}, with 128 being the default, if H is not included, to indicate different levels of truncation. R may take on {1,2} representing the truncation and significant digit rounding mode.

SUMMARY

The abstract UNF procedure can be thought of as a "semantic checksum". Technically, it is formed by creating a cryptographic hash of the normalized content of a digital object that has been approximated at an acceptable level of semantic fidelity.

The UNF algorithm itself is adapted for statistical/ scientific data. It provides a method for generating normalized representations of approximated vectors (and more complex objects formed from vectors) comprising numeric, character, time, date and duration values.

UNF's are now used in statistical applications to prevent misinterpretation of data, in archiving and citation practices for verification of content, and in digital library systems to verify format migration. In these systems and standards UNF's are, typically, used a replacement for or supplement to file-level cryptographic hashes and simple descriptive statistics

A notable aspect of the UNF implementation is that UNF's are calculated from within the software application acting on the digital object. This use can detect misinterpretation of the object, as well as corruption. In a sense, the UNF acts as a uniquely tailored summary statistic for the entire object

ACKNOWLEDGMENTS

Micah Altman 9/14/09 7:48 PM
Comment: As in previous versions, these should appear after the version:e.g. UNF:5:H256,N3: BvH3Rd+VMW6BXUWN3qGqow==

Micah Altman 8/31/09 12:18 PM
Deleted: , and Tiger 128 correspondingly

Micah Altman 8/31/09 12:18 PM
Deleted: These

Micah Altman 8/31/09 12:31 PM
Comment: E.g., rounding to tens of minutes or fractional hours is not supported.

Micah Altman 8/31/09 12:33 PM
Deleted: the format P[n]Y[n]M[n]DT[n]H[n]M[n]S

Thanks to Gary King, Akio Sone, and Kevin Condon, Gustavo Durand, Ellen Kraffmiller, Merce Crosas, and Elena Villalon for comments and suggestions.

This research was supported in part by an award (PA#NDP03-1) from the Library of Congress through its National Digital Information Infrastructure and Preservation Program (NDIIPP).

REFERENCES

- [1.] P. Cano, E. Batle, T. Kalker, J. Haistma, "A Reivew of Algorithms for Audio Fingerprinting", *IEEE Workshop on Multimedia Signal Processing*, IEEE Press, 2002, pp. 169-173.
- [2.] J. Oostveen, T. Kalker, J. Haitsma, "Feature Extaction and a Databse Strategy for Video Fingerprinting", Lecture Notes in Computer Science, vol 2314, Heidelberg: Springer Berlin, 2002.
- [3.] P. Meerwald, and A. Uhl., "A Survey of Wavelet-Domain Watermarking Algorithms" in *Proceedings of SPIE, Electronic Imaging, Security and Watermarking of Multimedia Contents III*, vol. 4314, 2001, pp. 506-516.
- [4.] M. Altman M, J. Gill , M.P. McDonald, *Numerical Issues in Statistical Computing for the Social Scientist*. New York: John Wiley & Sons, 2003
- [5.] R development Core Team, "R: A language and environment for statistical computing." R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. 2004.
- [6.] M Altman, L. Andreev, M. Diggory, M. Krot., G. King, D. Kiskis, E. Kolster, A. Sone., S. Verba. "An Introduction to the Virtual Data Center Project and Software." in *Proceedings of the First ACM+IEEE Joint Conference on Digital Libraries*. ACM Press, New York. 2001.
- [7.] G. King, . 2007. "An Introduction to the Dataverse Network as an Infrastructure for Data Sharing", *Sociological Methods and Research*. Forthcoming 2007.
- [8.] Altman, M., King, G. 2007. "A proposed Standard for the Scholarly Citation of Quantitative Data", D-Lib 13(3/4).
- [9.] M. Altman , J. Crabtree., D. Donakowski., M. Maynard, , "Data Preservation Alliance for the Social Sciences: A Model for Collaboration". Paper presented at DigCCurr 2007, Chapel Hill, N.C. 2007.
- [10.]M. Vardigan, C. Whiteman, "ICPSR meets OAIS: applying the OAIS reference model in the social science archive context", *Archival Science*, 7(1) 2007. pp 73-87.
- [11.]K.J. Renze, J.H. Oliver JH, "Generalized Unstructured Decimation." IEEE Computer Graphics and Applications, vol. 16, Cole, Vardigan, Archival Science, ICPSR meets OAIS: applying the OAIS reference model to the social science archive contex. pp 24-32.
- [12.]IEEE, *Programs for Digital Signal Processing*, New York: IEEE Press, 1979.
- [13.]Unicode Consortium, *The Unicode Standard*, Version 4.0.0. Boston: Addison-Wesley, 2003.
- [14.]National Institute of Standards and Technology (NIST), (2002). "Secure Hash Algorithm." NIST FIPS 180-2.
- [15.]Josefsson, S , "The Base16, Base32, and Base64 Data Encodings" RFC 3548. 2003.
- [16.]A. Joux, "Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions."in *Lecture Notes in Computer Science* 3152. Springer Berlin: Heidelberg, 2004, pp. 206-317
- [17.] Q. Dang, Recommendation for Using Approved Hash Algorithms, NIST Special Publication 800-107.Nastional Institute of Standards and Technology. (Draft, July 2007)
- [18.]International Studies Organization. *Data elements and interchange formats Information interchange Representation of dates and times*. ISO Standard 8601, (3d ed.). 2004
- [19.] International Studies Organization. *Abstract Syntax Notation One (ASN.1)*, b

Micah Altman 8/31/09 12:17 PM

Deleted: ,

Micah Altman 8/31/09 12:17 PM

Deleted: and