GEDCOM Unique Identifiers

by Gordon Clarke — last modified 2007-06-08 16:06

Guidelines and sample code for UIDs for GEDCOM

The need for Globally Unique Identifiers

We face a challenge as GEDCOM files are passed between users and imported into the system multiple times. We believe the standardize use of a Globally Unique ID (GUID) within GEDCOM files would make it easier for FamilySearch and other family history applications to know if a record that is being imported is identical to some other existing record.

Guidelines

We have proofed the following guidelines for UIDs.

1. Each individual and family record in your database should be assigned a UID.

2. This UID should be included during export and preserved during import (even when the exporting and importing programs are from different vendors).

3. The UID should be a 16-byte integer.

4. For windows applications we recommend using the CoCreateGuid() API to generate the UID. For other platforms please use a method to generate the UID which guarantees that the number will be globally unique.

5. During merge all UIDs should be preserved. This implies that a record could have n number of UIDs.

6. During export the UID should be associated with the _UID tag and represented as a textual hex number. See below an example of C++ code that converts to 16-byte UID to a text representation of a hex number and back. The example also shows the creation and checking of a check digit. The check digit has been implemented in PAF since version 5.0 and is encouraged but not required.

Here's the sample code:

```
 Uid.h:
/***********************************************************************
*******
 *   Copyright (c) 2000   Intellectual Reserve, Inc.  All rights
```

```
reserved.
 *                        Unauthorized reproduction of this software is
 *                        prohibited and is in violation of United
States
 *                        copyright laws.
 **********************************************************************
*******/
#pragma once
// uid.h
BOOL UidToString(PBYTE pBinary, CString& sResult, BOOL bAddChecksum =
FALSE);
BOOL StringToUid(PWSTR pString, PBYTE pBinary);
```

Uid.cpp:

```
/**********************************************************************
*******
 *   Copyright (c) 2000   Intellectual Reserve, Inc.  All rights
reserved.
 *                        Unauthorized reproduction of this software is
 *                        prohibited and is in violation of United
States
 *                        copyright laws.
 **********************************************************************
*******/
// uid.cpp
#include "stdafx.h"
#include "uid.h"

// convert a binary Unique ID to a string
// return TRUE if converted, FALSE if there wasn't a UID
BOOL UidToString(
       PBYTE pBinary,                    // pointer to 16 bytes of data
(a GUID)
       CString& sResult,          // place to return the string
       BOOL bAddChecksum)   // TRUE if a checksum should be added to
the end
{
       unsigned char checkA = 0;
       unsigned char checkB = 0;
       CString sDigits;
       BOOL bEmpty = TRUE;

       // clear result to start
       sResult.Empty();

       for (int i = 0; i < 16; i++, pBinary++)
       {
              // keep track of whether we really have a uid
              if (*pBinary)
                     bEmpty = FALSE;

              // build ongoing checksum
              checkA += *pBinary;
              checkB += checkA;

              // add next set of digits
              sDigits.Format(L"%02X", *pBinary);
```

```
                sResult += sDigits;
        }

        if (bAddChecksum)
        {
                sDigits.Format(L"%02X", checkA);
                sResult += sDigits;
                sDigits.Format(L"%02X", checkB);
                sResult += sDigits;
        }

        if (bEmpty)
        {
                sResult.Empty();
                return FALSE;
        }

        return TRUE;
}

// convert a string version of an ID to its binary value
// return FALSE if string was not valid or had an invalid checksum
BOOL StringToUid(
        PWSTR pString,             // pointer to string UID
        PBYTE pBinary)             // pointer to buffer (16 bytes) where
binary UID will be returned
{
        PBYTE pBinaryOrg = pBinary;
        BOOL bValidString = TRUE;

        // see if we have a valid length (with or without a checksum)
        int nLen = wcslen(pString);
        if (nLen != 32 && nLen != 36)
                bValidString = FALSE;

        if (bValidString)
        {
                unsigned char checkA = 0;
                unsigned char checkB = 0;
                int nNibble[2];

                for (int i = 0; i < 16 && bValidString; i++, pBinary++)
                {
                        for (int j = 0; j < 2; j++, pString++)
                        {
                              if (*pString >= '0' && *pString <= '9')
                                       nNibble[j] = *pString - '0';
                              else if (*pString >= 'A' && *pString <= 'F')
                                      nNibble[j] = *pString - 'A' + 10;
                              else
                                      bValidString = FALSE;
                        }

                        *pBinary = (nNibble[0] << 4) + nNibble[1];

                        // compute ongoing checksum
                        checkA += *pBinary;
```

```
            checkB += checkA;
        }

        // verify the checksum
        if (bValidString && nLen == 36)
        {
            unsigned char checkVerify[2];

            for (int i = 0; i < 2 && bValidString; i++)
            {
                for (int j = 0; j < 2; j++, pString++)
                {
                    if (*pString >= '0' && *pString <=
'9')
                        nNibble[j] = *pString - '0';
                    else if (*pString >= 'A' && *pString
<= 'F')
                        nNibble[j] = *pString - 'A' +
10;
                    else
                        bValidString = FALSE;
                }

                checkVerify[i] = (nNibble[0] << 4) +
nNibble[1];
            }
            if (checkVerify[0] != checkA || checkVerify[1] !=
checkB)
                bValidString = FALSE;
        }
    }

    if (!bValidString)
    {
        memset(pBinaryOrg, 0, 16);
        return FALSE;
    }

    return TRUE;
}
```