Xcode DerivedData Hashes

Xcode (by default) uses “Unique” build locations for each project and stores them in a folder called “DerivedData”. This folder is located (by default) at `~/Library/Developer/Xcode/DerivedData/`. Within that folder each folder that is labeled with the following format:

```
[name of the root project/workspace]-[28 character identifier]
```

There seemed to be no way to resolve what that identifier was in relation to the root project file that was open. This seemed to be barely mentioned online and only refered to as “application hash”. After some exploration in Hopper, I found the function call that creates the unique identifier. Below is a reconstructed of the function `hashStringForPath` in DevToolsCore.framework:

```
#import <Foundation/Foundation.h>
#import <CommonCrypto/CommonCrypto.h>

// this function is used to swap byte ordering of a 64bit integer
uint64_t swap_uint64(uint64_t val) {
    val = ((val << 8) & 0xFF00FF00FF00FF00ULL ) | ((val >> 8) & 0x00FF0
    val = ((val << 16) & 0xFFFF0000FFFF0000ULL ) | ((val >> 16) & 0x000
    return (val << 32) | (val >> 32);
}

/*!
    @function hashStringForPath

    Create the unique identifier string for a Xcode project path

    @param path (input) string path to the ".xcodeproj" or ".xcworkspac

    @result NSString* of the identifier
*/
NSString * hashStringForPath(NSString *path) {
    // using uint64_t[2] for ease of use, since it is the same size as
    uint64_t digest[2] = {0};

    // char array that will contain the identifier
    unsigned char resultStr[28] = {0};

    // setup md5 context
    CC_MD5_CTX md5;
    CC_MD5_Init(&md5);

    // get the UTF8 string of the path
    const char *c_path = [path UTF8String];

    // get length of the path string
    unsigned long length = strlen(c_path);

    // update the md5 context with the full path string
    CC_MD5_Update (&md5, c_path, length);

    // finalize working with the md5 context and store into the digest
    CC_MD5_Final (digest, &md5);

    // take the first 8 bytes of the digest and swap byte order
    uint64_t startValue = swap_uint64(digest[0]);

    // for indexes 13->0
    int index = 13;
    do {
        // take 'startValue' mod 26 (restrict to alphabetic) and add ba
        resultStr[index] = (char)((startValue % 26) + 'a');
```

```
        // divide 'startValue' by 26
        startValue /= 26;

        index--;
    } while (index >= 0);

    // The second loop, this time using the last 8 bytes
    // repeating the same process as before but over indexes 27->14
    startValue = swap_uint64(digest[1]);
    index = 27;
    do {
        resultStr[index] = (char)((startValue % 26) + 'a');
        startValue /= 26;
        index--;
    } while (index > 13);

    // create a new string from the 'resultStr' char array and return
    return [[[NSString alloc] initWithBytes:resultStr length:28 encodin
}
```