getdns API Design Review

Bob Steagall

VERISIGN

22-Jan-2014

1 Overview

This document provides a preliminary design review of the `getdns` API, version 0.320, as published on the VPNC website at http://www.vpnc.org/getdns-api. It does not evaluate or comment on the underlying object model implied by the API, nor does it address how well that model would fulfill its intended purpose. Instead, the review focusses purely upon the "mechanics" of the interface, looking for possible problem areas and opportunities for improvement.

2 API Refinement

This section is primarily an effort to identify areas where problems may exist, and suggest ways to address those problems. The underlying assumption is that the API is almost fully baked, and large changes, such as those that would result from a major refactoring, are not feasible.

2.1 Function Return Types and Integer Parameters

I am puzzled by the use of `unit16_t` (typedef'd as `getdns_return_t`) as the return type for all functions in the API, the use of `uint16_t` as the parameter type chosen to convey the many constants defined by the API (as macros), and the use of `uint16_t` and `uint8_t` as the types of certain numerical parameters. I'm not sure I see a benefit in using narrower integer types for these purposes, while I can see some disadvantages:

- There is a greater chance of generating annoying compiler warnings when using these narrow types for comparisons and assignments in code outside the API.
- Casting parameters to `uint16_t` or `uint8_t` from wider integer types increases the chances of accidental value truncation errors. It appears that the intent is to use narrower types in order to constrain parameters to a smaller range of "always-correct" values and avoid validation. I don't believe there is any efficiency to be gained, and in the worst case, this could be a source of difficult-to-find errors.
- The effectiveness of symbolic debuggers is reduced a little when examining return values and parameters corresponding to API constants. These items will appear as simple integers with "magic" values.

Instead of using a native integer type to represent return status codes and parameter constants, the API could define several enumerations with corresponding type aliases, with each enumeration-alias pair dedicated to a related group of constants. For example:

```
typedef enum getdns_return_code
{
    GETDNS_RETURN_GOOD                          = 0,
    GETDNS_RETURN_GENERIC_ERROR                 = 1,
    GETDNS_RETURN_BAD_DOMAIN_NAME               = 300,
    GETDNS_RETURN_BAD_CONTEXT                   = 301,
    GETDNS_RETURN_CONTEXT_UPDATE_FAIL           = 302,
    GETDNS_RETURN_UNKNOWN_TRANSACTION           = 303,
    GETDNS_RETURN_NO_SUCH_LIST_ITEM             = 304,
    GETDNS_RETURN_NO_SUCH_DICT_NAME             = 305,
    GETDNS_RETURN_WRONG_TYPE_REQUESTED          = 306,
    GETDNS_RETURN_NO_SUCH_EXTENSION             = 307,
    GETDNS_RETURN_EXTENSION_MISFORMAT           = 308,
    GETDNS_RETURN_DNSSEC_WITH_STUB_DISALLOWED = 309,
    GETDNS_RETURN_SET_REP_WIDTH_DO_NOT_USE    = 0x7FFFFFFF
}   getdns_return_t;
```

```
typedef enum getdns_dnssec_value
{
    GETDNS_DNSSEC_SECURE        = 400,
    GETDNS_DNSSEC_BOGUS         = 401,
    GETDNS_DNSSEC_INDETERMINATE = 402,
    GETDNS_DNSSEC_INSECURE      = 403,
    GETDNS_DNSSEC_NOT_PERFORMED = 404,
    GETDNS_DNSSEC_SET_REP_WIDTH_DO_NOT_USE = 0x7FFFFFFF
}   getdns_dnssec_t;
```

Note that the last value in each enumeration is a dummy that serves to specify the minimum width for all values of that enumeration. I've chosen a value which will lead to enumerated value sizes of 32 bits, which is a natural word size on many platforms.

Using enumerations instead of native integers and macros provides a few advantages:

- The C language provides no type safety when using enumerations, freely and quietly converting back and forth from enum to integer types. However, C++ treats enums as distinct types, and so some type safety is possible when the API header is included in C++ translation units. I believe any type safety the API can provide is worth having, even if it accrues only to C++ code.
- Using enumerated values can aid debugging when using symbolic debuggers. These days, most debuggers have the ability to show an enumeration's symbol as well as its value.
- Storage requirements for enumerations as described above is predictable and alignment is natural (e.g., on Intel HW, being of word size and falling on word boundaries).

2.2 Type Aliases for Opaque Types

The API should provide type aliases for all of its opaque types. This will promote a consistent conceptual model and improve readability in both client code and the API itself. For example, the `struct getdns_dict` and `struct getdns_list` data structures both appear to be opaque types that will not provide direct access to their members, and thus there is no reason for client code to work directly with pointers to these types.

In order to be consistent with `getdns_context_t`, as well as in the interest of promoting const correctness when `struct getdns_dict` and `struct getdns_list` are defined in the API header, they could be treated as handle types:

```
typedef struct getdns_dict const*  getdns_dict_t;
typedef struct getdns_list const*  getdns_list_t;
```

From what I can tell, client code will use accessor functions for retrieving data from `dict` and `list` objects, so the `const` is warranted in this scenario. Internal implementation functions can cast away the const in order to perform modification and deallocation of `dict` and `list` objects, presuming that the client has respected their const-ness and not directly modified their internal state.

On the other hand, if the API header contains only forward declarations of `struct getdns_dict` and `struct getdns_list`, then the const is unnecessary, and the handles could be defined as:

```
typedef struct getdns_dict*    getdns_dict_t;
typedef struct getdns_list*    getdns_list_t;
```

2.3 Const Correctness

The API as presented in version 0.320 does not really provide const correctness, an important safety mechanism that C and C++ code should implement wherever possible. In particular:

- an input parameter that is a pointer should be a pointer-to-const if the argument is not changed by the function; and,
- an output parameter that is a pointer should be a pointer-to-pointer-to-const if the client cannot change the pointed-to return value.

For example, in the `getdns_dict_get_*` and `getdns_dict_set_*` sets of helper functions, the second parameter in each function could be changed to `char const*`:

```
getdns_return_t getdns_dict_get_int
(
    getdns_dict_t   this_dict,
    char const*     name,
    uint32_t*       answer
);

getdns_return_t getdns_dict_set_int
(
    getdns_dict_t   this_dict,
    char const*     name,
    uint32_t        child_uint32
);
```

2.4 Provide Lengths of Mutable String Buffers

To avoid possible problems with buffer overruns, the API should supply lengths to all functions that modify mutable character buffers. For example:

```
char*   getdns_convert_dns_name_to_fqdn
(
    char*   name,
    size_t  len
);

char*   getdns_convert_fqdn_to_dns_name
(
    char*   name,
    size_t  len
);

char*   getdns_convert_ulabel_to_alabel
(
    char*   label,
    size_t  len
);

char*   getdns_convert_alabel_to_ulabel
(
    char*   label,
    size_t  len
);
```

2.5 Additional String Manipulators

A second version of the string buffer manipulation functions listed above could be created with both source and destination parameters, and a size parameter specifying the size of the destination buffer. If a left-to-right source-to-destination convention is used:

```
char*   getdns_convert_dns_name_to_fqdn_2
(
    char const* src_name,
    size_t      src_len,
    char*       dst_name,
    size_t      dst_len
);

char*   getdns_convert_fqdn_to_dns_name_2
(
    char const* src_name,
    size_t      src_len,
    char*       dst_name,
    size_t      dst_len
);

char*   getdns_convert_ulabel_to_alabel_2
(
    char const* src_ulabel,
    size_t      src_len,
    char*       dst_alabel,
    size_t      dst_len
);

char*   getdns_convert_alabel_to_ulabel_2
(
    char const* src_alabel,
    size_t      src_len,
    char*       dst_ulabel,
    size_t      dst_len
);
```

Note that the size of the source array is provided in the examples above. While one could make an argument against including this extra parameter, it does have the advantages of allowing for early parameter validation and permitting the use of embedded substrings as source arguments.

2.6 Remove Unnecessary Header Dependencies

If no macro, type, variable, or function declaration or definition from `<netinet/in.h>` is referenced by the `getdns` header, then `<netinet/in.h>` should not be included.

2.7 C++ Compilation Support

The header containing the C API should be modified with the usual conditional compilation directives in order to support compilation with C++:

```
/* Created at 2013-04-02-16-59-04*/
#ifndef GETDNS_H
#define GETDNS_H

#include <stdint.h>
#include <stdlib.h>
#include <stdbool.h>

#ifdef __cplusplus
extern "C" {
#endif
```

```
/*  BODY OF THE HEADER GOES HERE...
*/

#ifdef __cplusplus
}
#endif
#endif
```

3 API Refactoring

This section of the review is an attempt to refactor and improve the usability of the API. As such, the comments and recommendations in this section assume that much larger changes in the API are possible. They reflect my personal tastes as well as the principles I've employed over the years to build numerous libraries, specifically:

- Provide an interface that concisely and elegantly expresses a coherent conceptual model.
- Promote understandability, readability, and mnemonic integrity.
- Eliminate or minimize opportunities for ambiguity and/or confusion.
- Eliminate or minimize annoying and/or repetitive work that must be performed by client code.

3.1 Names

Note: In the interest of brevity, I have left out the parameters from the function declarations listed in this section. An ellipsis simply means a function's parameters have been elided for purposes of discussion; it *does not* indicate a variable-length parameter list.

3.1.1 Functions and Macros - General

The first opportunity for improvement I see lies in the naming conventions used by the API for function and macro names. Because the C programming language has no concept of namespaces, it is a common practice to use a unique prefix to disambiguate the function names and macro names that belong a given library. In the case of the `getdns` API, functions are prefixed by "`getdns_`" and macros are prefixed by "`GETDNS_`". To me, including the verb "get" as part of the prefix in the API's accessor functions violates the DRY (don't repeat yourself) principle. Consider, for example:

```
getdns_return_t getdns_list_get_list(...);
```

When building a C API that employs prefixes in this way, I try to make the prefixes to be as neutral as possible – usually names, rarely adjectives or nouns, and never verbs. In the case of the `getdns` API, I recommend removing the verb from the prefix by changing the function and macro name prefixes to "`vdns_`" and "`VDNS_`", respectively, where the leading "v" or "V" derives from VPNC, e.g.:

```
typedef enum vdns_nametype_value
{
    VDNS_NAMETYPE_DNS   = 800,
    ...
}   vdns_nametype_t;

#define VDNS_NAMETYPE_DNS_TEXT    Normal DNS (RFC 1035)

vdns_return_t   vdns_list_get_length(...);
```

Another alternative is to change the function and macro name prefixes to "gdns_" and "GDNS_", respectively, preserving some of the heritage of getdns. The remainder of this section uses the "v"/"V" names for purposes of discussion.

3.1.2 Synchronous and Asynchronous Lookup Functions

The second opportunity to improve the getdns API naming conventions lies in the distinction between the synchronous and asynchronous (event-driven) function names, and in the internal structure of those names. The API appears to make the assumption that the default mode of programming will be asynchronous, and disambiguates the synchronous version with the "_sync" suffix.

Most programmers using the API for the first time will probably assume that the unadorned function names are the default mode, and also incorrectly assume that the default mode is a synchronous programming model. This mismatch in expectations is a potential source of confusion.

I recommend that the suffix "_async" be added to all of the asynchronous functions, and that an appropriate verb be added to the function names for both asynchronous and synchronous variants:

```
vdns_return_t   vdns_get_general_async(...);
vdns_return_t   vdns_get_address_async(...);
vdns_return_t   vdns_get_hostname_async(...);
vdns_return_t   vdns_get_service_async(...);
vdns_return_t   vdns_cancel_callback(...);

vdns_return_t   vdns_get_general_sync(...);
vdns_return_t   vdns_get_address_sync(...);
vdns_return_t   vdns_get_hostname_sync(...);
vdns_return_t   vdns_get_service_sync(...);
```

In this way, the purpose and execution model of these key functions are clearly and unambiguously documented in their names. In addition to improving clarity, this convention will ensure consistency with the other function name changes that I recommend below.

3.1.3 Verb-Object Inversion

The final opportunity for improving naming convections lies in the way helper function names are constructed. Almost all of the functions in the API have a name wherein an object type appears before the relevant verb, for example:

```
getdns_return_t getdns_context_create(...);
void            getdns_context_destroy(...);
```

In addition to changing the prefix, I recommend that the object and verb in these names be transposed:

```
vdns_return_t   vdns_create_context(...);
void            vdns_destroy_context(...);
```

In other words, the verb or command portion of the name appears immediately after the prefix, followed by the object of that command.

I've also noticed that the names of some support functions in the API for manipulating list and dict objects can be somewhat repetitious. For example, the two accessor functions

```
getdns_return_t getdns_list_get_list(...);
```

and

```
getdns_return_t getdns_dict_get_dict(...);
```

both seem a little awkward. In order to provide self-documentation of purpose and eliminate intra-name repetition, I recommend that this entire set of functions be renamed as follows:

```
vdns_list_t     vdns_create_list();
void            vdns_destroy_list(...);

vdns_dict_t     vdns_create_dict();
void            vdns_destroy_dict(...);

vdns_return_t   vdns_get_list_length(...);
vdns_return_t   vdns_get_list_element_data_type(...);
vdns_return_t   vdns_get_list_element_as_dict(...);
vdns_return_t   vdns_get_list_element_as_list(...);
vdns_return_t   vdns_get_list_element_as_bindata(...);
vdns_return_t   vdns_get_list_element_as_int(...);

vdns_return_t   vdns_get_dict_names(...);
vdns_return_t   vdns_get_dict_data_type(...);
vdns_return_t   vdns_get_dict_element_as_dict(...);
vdns_return_t   vdns_get_dict_element_as_list(...);
vdns_return_t   vdns_get_dict_element_as_bindata(...);
vdns_return_t   vdns_get_dict_element_as_int(...);

vdns_return_t   vdns_set_list_element_as_dict(...);
vdns_return_t   vdns_set_list_element_as_list(...);
vdns_return_t   vdns_set_list_element_as_bindata(...);
vdns_return_t   vdns_set_list_element_as_int(...);

vdns_return_t   vdns_set_dict_element_as_dict(...);
vdns_return_t   vdns_set_dict_element_as_list(...);
vdns_return_t   vdns_set_dict_element_as_bindata(...);
vdns_return_t   vdns_set_dict_element_as_int(...);
```

With these changes, a function's name clearly documents its operation (get or set), and whether it operates upon an entire `list`/`dict` or upon a single element. To my mind, it also makes the function name read in a more natural way.

A similar recommendation applies to those support functions that modify context attributes:

```
vdns_return_t   vdns_set_context_update_callback(...);
vdns_return_t   vdns_set_context_resolution_type(...);
vdns_return_t   vdns_set_context_namespaces(...);
vdns_return_t   vdns_set_context_dns_transport(...);
vdns_return_t   vdns_set_context_limit_outstanding_queries(...);

vdns_return_t   vdns_set_context_timeout(...);
vdns_return_t   vdns_set_context_follow_redirects(...);
vdns_return_t   vdns_set_context_dns_root_servers(...);
vdns_return_t   vdns_set_context_append_name(...);
vdns_return_t   vdns_set_context_suffix(...);

vdns_return_t   vdns_set_context_dnssec_trust_anchors(...);
vdns_return_t   vdns_set_context_dnssec_allowed_skew(...);
vdns_return_t   vdns_set_context_stub_resolution(...);
vdns_return_t   vdns_set_context_edns_maximum_udp_payload_size(...);
vdns_return_t   vdns_set_context_edns_extended_rcode(...);

vdns_return_t   vdns_set_context_edns_version(...);
vdns_return_t   vdns_set_context_edns_do_bit(...);
```

```
vdns_return_t   vdns_set_context_memory_allocator(...);
vdns_return_t   vdns_set_context_memory_deallocator(...);
vdns_return_t   vdns_set_context_memory_reallocator(...);
```

3.2 Modifying `bindata` Elements of `list` or `dict` Objects

The functions for modifying `bindata` elements of a `list` or `dict` require the client code to supply a fully-formed `bindata` parameter:

```
vdns_return_t
vdns_set_list_element_as_bindata
(
    vdns_list_t     this_list,
    size_t          index,
    vdns_bindata_t  child_bindata
);

vdns_return_t
vdns_set_dict_element_as_bindata
(
    vdns_dict_t     this_dict,
    char const*     name,
    vdns_bindata_t  child_bindata
);
```

While this is clearly useful if one desires to copy the contents of one `bindata` element into another, it also makes the *ad hoc* modification of a `list` or `dict` element a little more awkward than it needs to be. I recommend that a second variant of each of these functions be added so that the size and content of the `bindata` may be set directly:

```
vdns_return_t
vdns_set_list_element_as_bindata_2
(
    vdns_list_t     this_list,
    size_t          index,
    uint8_t*        bindata_content,
    size_t          bindata_size
);

vdns_return_t
vdns_set_dict_element_as_bindata_2
(
    vdns_dict_t     this_dict,
    char const*     name,
    uint8_t*        bindata_content,
    size_t          bindata_size
);
```

3.3 Platform Independence

The API must ensure that no assumptions are made, nor are any constructs used, that may restrict its platform independence. This comment is more reminder than substantive at this point.