Jansson Documentation

Release 1.0.4

Petri Lehtinen

August 20, 2013

CONTENTS

This is the documentation for Jansson 1.0.4, last updated August 20, 2013.

Contents:

CHAPTER

ONE

API REFERENCE

1.1 Preliminaries

All declarations are in `jansson.h`, so it's enough to

```
#include <jansson.h>
```

in each source file.

All constants are prefixed `JSON_` and other identifiers with `json_`. Type names are suffixed with `_t` and `typedef`'d so that the `struct` keyword need not be used.

1.2 Value Representation

The JSON specification (RFC 4627) defines the following data types: *object*, *array*, *string*, *number*, *boolean*, and *null*. JSON types are used dynamically; arrays and objects can hold any other data type, including themselves. For this reason, Jansson's type system is also dynamic in nature. There's one C type to represent all JSON values, and this structure knows the type of the JSON value it holds.

json_t
: This data structure is used throughout the library to represent all JSON values. It always contains the type of the JSON value it holds and the value's reference count. The rest depends on the type of the value.

Objects of `json_t` are always used through a pointer. There are APIs for querying the type, manipulating the reference count, and for constructing and manipulating values of different types.

1.2.1 Type

The type of a JSON value is queried and tested using the following functions:

enum **json_type**
: The type of a JSON value. The following members are defined:

JSON_OBJECT
JSON_ARRAY
JSON_STRING
JSON_INTEGER
JSON_REAL
JSON_TRUE
JSON_FALSE
JSON_NULL

These correspond to JSON object, array, string, number, boolean and null. A number is represented by either a value of the type `JSON_INTEGER` or of the type `JSON_REAL`. A true boolean value is represented by a value of the type `JSON_TRUE` and false by a value of the type `JSON_FALSE`.

int **json_typeof**(const json_t **json*)
Return the type of the JSON value (a `json_type` cast to `int`). This function is actually implemented as a macro for speed.

json_is_object(const json_t **json*)
json_is_array(const json_t **json*)
json_is_string(const json_t **json*)
json_is_integer(const json_t **json*)
json_is_real(const json_t **json*)
json_is_true(const json_t **json*)
json_is_false(const json_t **json*)
json_is_null(const json_t **json*)
These functions (actually macros) return true (non-zero) for values of the given type, and false (zero) for values of other types.

json_is_number(const json_t **json*)
Returns true for values of types `JSON_INTEGER` and `JSON_REAL`, and false for other types.

json_is_boolean(const json_t **json*)
Returns true for types `JSON_TRUE` and `JSON_FALSE`, and false for values of other types.

1.2.2 Reference Count

The reference count is used to track whether a value is still in use or not. When a value is created, it's reference count is set to 1. If a reference to a value is kept (e.g. a value is stored somewhere for later use), its reference count is incremented, and when the value is no longer needed, the reference count is decremented. When the reference count drops to zero, there are no references left, and the value can be destroyed.

The following functions are used to manipulate the reference count.

json_t ***json_incref**(json_t **json*)
Increment the reference count of *json*.

void **json_decref**(json_t **json*)
Decrement the reference count of *json*. As soon as a call to `json_decref()` drops the reference count to zero, the value is destroyed and it can no longer be used.

Functions creating new JSON values set the reference count to 1. These functions are said to return a **new reference**. Other functions returning (existing) JSON values do not normally increase the reference count. These functions are said to return a **borrowed reference**. So, if the user will hold a reference to a value returned as a borrowed reference, he must call `json_incref()`. As soon as the value is no longer needed, `json_decref()` should be called to release the reference.

Normally, all functions accepting a JSON value as an argument will manage the reference, i.e. increase and decrease the reference count as needed. However, some functions **steal** the reference, i.e. they have the same result as if the user called `json_decref()` on the argument right after calling the function. These are usually convenience functions for adding new references to containers and not to worry about the reference count.

In the following sections it is clearly documented whether a function will return a new or borrowed reference or steal a reference to its argument.

1.3 True, False and Null

json_t ***json_true**(void)
Return value: New reference. Returns a value of the type `JSON_TRUE`, or *NULL* on error.

json_t ***json_false**(void)
Return value: New reference. Returns a value of the type `JSON_FALSE`, or *NULL* on error.

json_t ***json_null**(void)
Return value: New reference. Returns a value of the type `JSON_NULL`, or *NULL* on error.

1.4 String

json_t ***json_string**(const char **value*)
Return value: New reference. Returns a new value of the type `JSON_STRING`, or *NULL* on error. *value* must be a valid UTF-8 encoded Unicode string.

const char ***json_string_value**(const json_t **json*)
Returns the associated value of a `JSON_STRING` value as a null terminated UTF-8 encoded string.

1.5 Number

json_t ***json_integer**(int *value*)
Return value: New reference. Returns a new value of the type `JSON_INTEGER`, or *NULL* on error.

int **json_integer_value**(const json_t **json*)
Returns the associated integer value of values of the type `JSON_INTEGER`, or 0 for values of other types.

json_t ***json_real**(double *value*)
Return value: New reference. Returns a new value of the type `JSON_REAL`, or *NULL* on error.

double **json_real_value**(const json_t **json*)
Returns the associated real value of values of the type `JSON_INTEGER`, or 0 for values of other types.

In addition to the functions above, there's a common query function for integers and reals:

double **json_number_value**(const json_t **json*)
Returns the value of either `JSON_INTEGER` or `JSON_REAL`, cast to double regardless of the actual type.

1.6 Array

A JSON array is an ordered collection of other JSON values.

json_t ***json_array**(void)
Return value: New reference. Returns a new value of the type `JSON_ARRAY`, or *NULL* on error. Initially, the array is empty.

unsigned int **json_array_size**(const json_t **array*)
Returns the number of elements in *array*.

json_t ***json_array_get**(const json_t **array*, unsigned int *index*)
Return value: Borrowed reference. Returns the element in *array* at position *index*, or *NULL* if *index* is out of range. The valid range for *index* is from 0 to the return value of `json_array_size()` minus 1.

int **json_array_set** (json_t **array*, unsigned int *index*, json_t **value*)
Replaces the element in *array* at position *index* with *value*. Returns 0 on success, or -1 if *index* is out of range. The valid range for *index* is from 0 to the return value of `json_array_size()` minus 1.

int **json_array_append** (json_t **array*, json_t **value*)
Appends *value* to the end of *array*, growing the size of *array* by 1. Returns 0 on success and -1 on error.

1.7 Object

A JSON object is a dictionary of key-value pairs, where the key is a Unicode string and the value is any JSON value.

json_t ***json_object** (void)
Return value: New reference. Returns a new value of the type `JSON_OBJECT`, or *NULL* on error. Initially, the object is empty.

json_t ***json_object_get** (const json_t **object*, const char **key*)
Return value: Borrowed reference. Get a value corresponding to *key* from *object*. Returns *NULL* if *key* is not found and on error.

int **json_object_set** (json_t **object*, const char **key*, json_t **value*)
Set the value of *key* to *value* in *object*. *key* must be a valid terminated UTF-8 encoded Unicode string. If there already is a value for *key*, it is replaced by the new value. Returns 0 on success and -1 on error.

int **json_object_del** (json_t **object*, const char **key*)
Delete *key* from *object* if it exists. Returns 0 on success, or -1 if *key* was not found.

The following functions implement an iteration protocol for objects:

void ***json_object_iter** (json_t **object*)
Returns an opaque iterator which can be used to iterate over all key-value pairs in *object*, or *NULL* if *object* is empty.

void ***json_object_iter_next** (json_t **object*, void **iter*)
Returns an iterator pointing to the next key-value pair in *object* after *iter*, or *NULL* if the whole object has been iterated through.

const char ***json_object_iter_key** (void **iter*)
Extract the associated key from *iter*.

json_t ***json_object_iter_value** (void **iter*)
Return value: Borrowed reference. Extract the associated value from *iter*.

1.8 Encoding

This sections describes the functions that can be used to encode values to JSON. Only objects and arrays can be encoded, since they are the only valid "root" values of a JSON text.

Each function takes a *flags* parameter that controls some aspects of how the data is encoded. Its default value is 0. The following macros can be ORed together to obtain *flags*.

JSON_INDENT(n) Pretty-print the result, indenting arrays and objects by *n* spaces. The valid range for *n* is between 0 and 255, other values result in an undefined output. If `JSON_INDENT` is not used or *n* is 0, no pretty-printing is done and the result is a compact representation.

The following functions perform the actual JSON encoding. The result is in UTF-8.

char *__json_dumps__(const json_t *_root_, uint32_t _flags_)
Returns the JSON representation of *root* as a string, or *NULL* on error. *flags* is described above. The return value must be freed by the caller using `free()`.

int __json_dumpf__(const json_t *_root_, FILE *_output_, uint32_t _flags_)
Write the JSON representation of *root* to the stream *output*. *flags* is described above. Returns 0 on success and -1 on error.

int __json_dump_file__(const json_t *_json_, const char *_path_, uint32_t _flags_)
Write the JSON representation of *root* to the file *path*. If *path* already exists, it is overwritten. *flags* is described above. Returns 0 on success and -1 on error.

1.9 Decoding

This sections describes the functions that can be used to decode JSON text to the Jansson representation of JSON data. The JSON specification requires that a JSON text is either a serialized array or object, and this requirement is also enforced with the following functions.

The only supported character encoding is UTF-8 (which ASCII is a subset of).

__json_error_t__
This data structure is used to return information on decoding errors from the decoding functions. Its definition is repeated here:

```
#define JSON_ERROR_TEXT_LENGTH  160

typedef struct {
    char text[JSON_ERROR_TEXT_LENGTH];
    int line;
} json_error_t;
```

line is the line number on which the error occurred, or -1 if this information is not available. *text* contains the error message (in UTF-8), or an empty string if a message is not available.

The normal usef of `json_error_t` is to allocate it normally on the stack, and pass a pointer to a decoding function. Example:

```
int main() {
    json_t *json;
    json_error_t error;

    json = json_load_file("/path/to/file.json", &error);
    if(!json) {
        /* the error variable contains error information */
    }
    ...
}
```

Also note that if the decoding succeeded (`json != NULL` in the above example), the contents of `error` are unspecified.

All decoding functions also accept *NULL* as the `json_error_t` pointer, in which case no error information is returned to the caller.

The following functions perform the actual JSON decoding.

json_t *__json_loads__(const char *_input_, json_error_t *_error_)
Return value: New reference. Decodes the JSON string *input* and returns the array or object it contains, or

NULL on error, in which case *error* is filled with information about the error. See above for discussion on the *error* parameter.

json_t ***json_loadf**(FILE **input*, json_error_t **error*)

Return value: New reference. Decodes the JSON text in stream *input* and returns the array or object it contains, or *NULL* on error, in which case *error* is filled with information about the error. See above for discussion on the *error* parameter.

json_t ***json_load_file**(const char **path*, json_error_t **error*)

Return value: New reference. Decodes the JSON text in file *path* and returns the array or object it contains, or *NULL* on error, in which case *error* is filled with information about the error. See above for discussion on the *error* parameter.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *search*