Programación en C moderno

Álvaro Neira Ayuso <alvaro@soleta.eu>

b) Libjansson (JSON): Biblioteca para exporta e importar datos en formato JSON.

- Introducción: Historia, tipos de objetos y software necesario.
- Estructura estándar de JSON.
- Ejemplo 1: Procesado, recorrido e impresión de un árbol sencillo.
  - Uso de funciones para procesar cadenas.
  - Uso de funciones para recorrer el árbol
  - Impresión del contenido del árbol
- Ejemplo 2: Procesado, recorrido e impresión de arboles con listas.
- Ejemplo 3: Lectura y escritura de ficheros en formato JSON.
- Ejemplo 4: Herramientas para validar la estructura de JSON.

Historia

- JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos.
- Cada vez hay más soporte de JSON mediante el uso de paquetes escritos por terceras parte.
- En diciembre de 2005 Yahoo! comenzó a dar soporte opcional de JSON en algunos de sus servicios web.

Historia

- El formato de JSON fue especificado por Douglas Crockford.
- Douglas Crockford es un programador Americano, el cuál es desarrollador de Javascript
- Actualmente es un diseñador y desarrollador Senior de Javascript para Paypal

Douglas Crockford

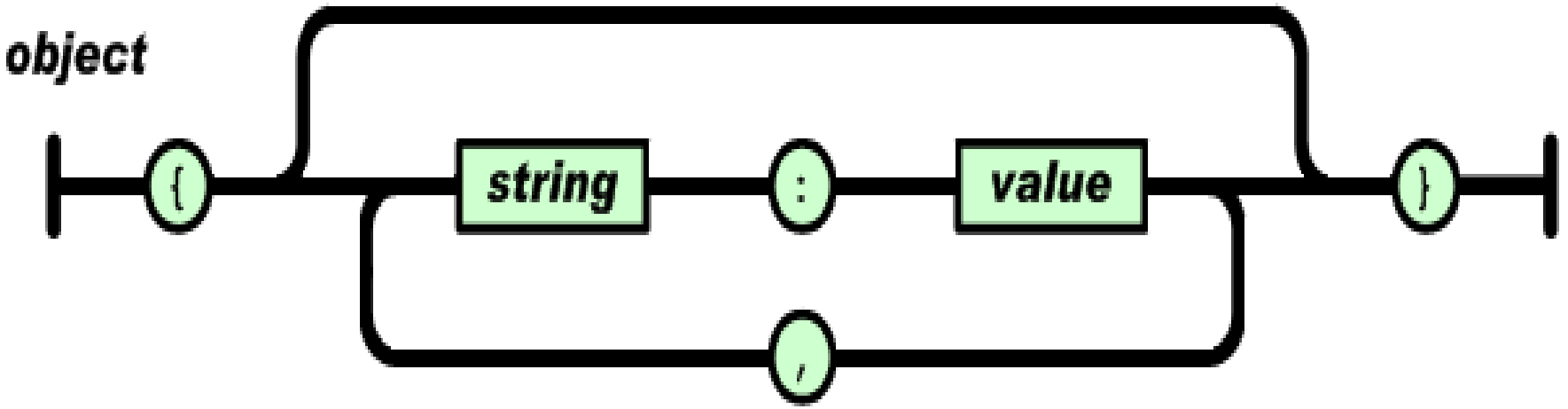
Historia: Jansson

- Jansson, biblioteca en C para codificar, decodificar y manipular datos en formato JSON.
- Sus principales características son:
  - Simple e intuitiva API y modelo de datos.
  - Documentación clara y accesible
  - Ninguna dependencia con otras bibliotecas
  - Soporte total con formato unicode (UTF-8)
- Jansson esta licenciado bajo licencia MIT.

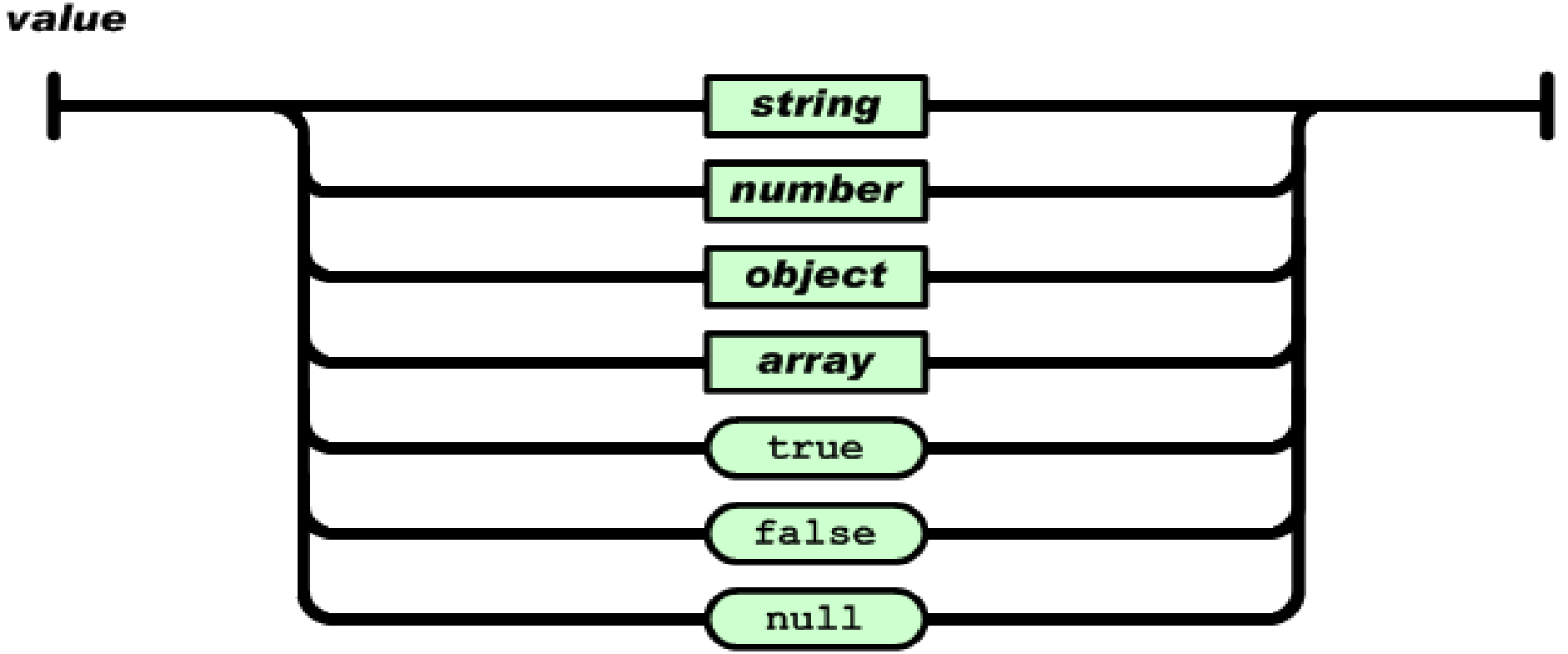
Software necesario

- libjansson4
- libjansson-dbg
- libjansson-dev
- libjansson-doc

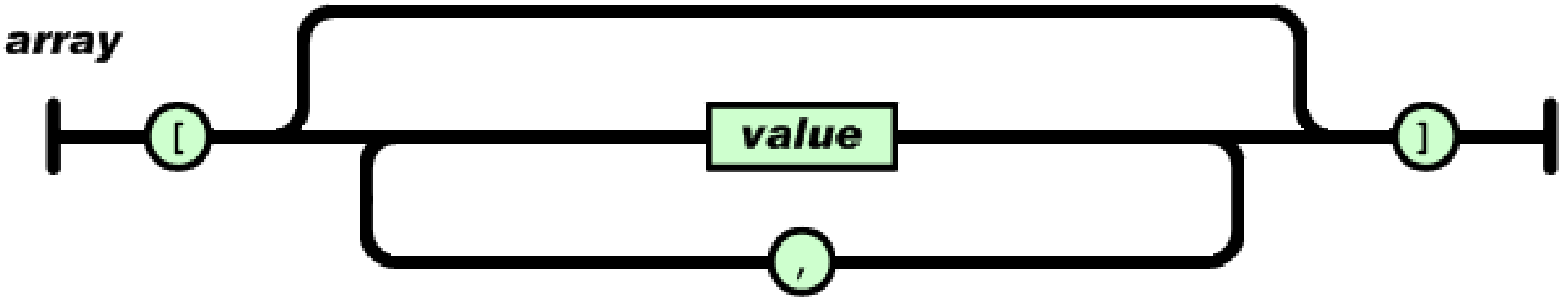
Formato JSON



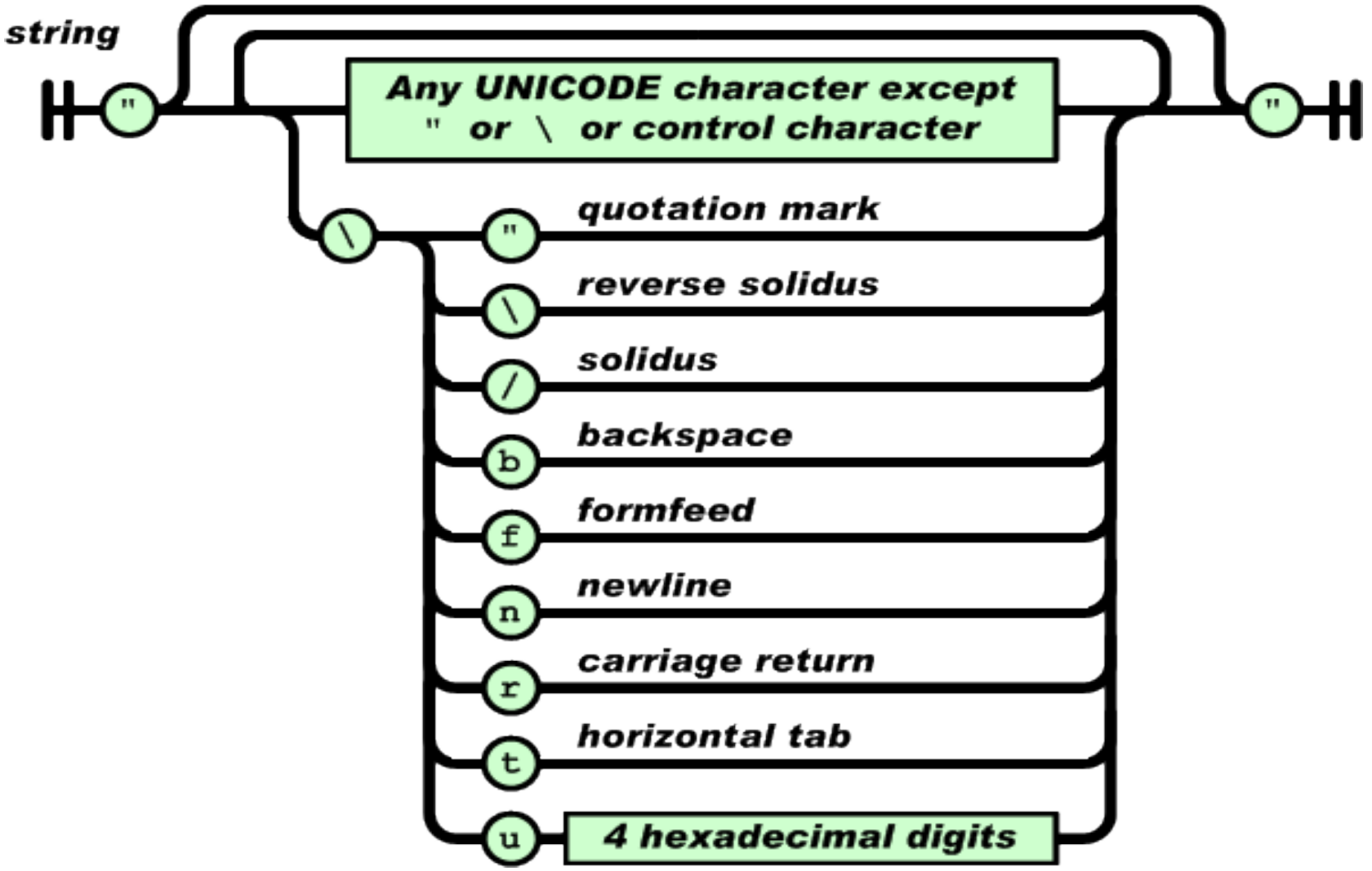
Formato JSON



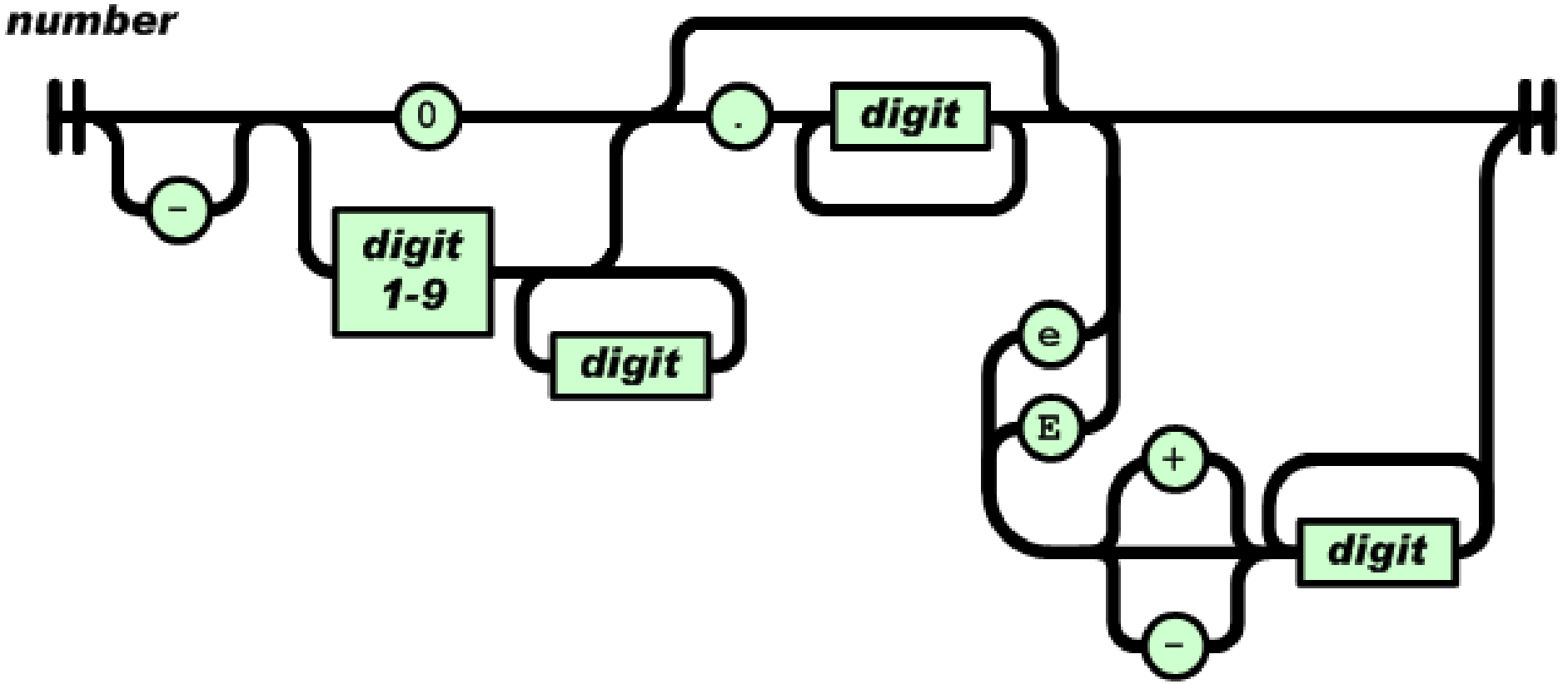
Formato JSON



Formato JSON



Formato JSON



Usar y compilar biblioteca jansson

- #include <jansson.h>

  Biblioteca que contiene todas las funciones y tipos de jansson

- gcc -o prog prog.c -ljansson

  Flag para activar el uso de libjansson

Jansson: String

Jansson usa UTF-8 como formato de codificación de caracteres

- json_t *json_string(const char *value)

  Devuelve un nuevo objeto JSON de tipo string o NULL en caso de error.

Jansson: String

- size_t json_string_length(const json_t *string)

  Devuelve el tamaño del string contenido en el objeto JSON de tipo string

- int json_string_set(const json_t *string, const char *value)

  Establece la cadena value en el objeto JSON de tipo string.

Jansson: String

- const char *json_string_value(const json_t *string)

  Devuelve la cadena contenida en el objeto JSON de tipo string

Jansson: Enteros

- Json solo tiene un tipo número “number”.
- Dicho tipo de número contiene tanto enteros como float
- El objeto de tipo JSON json_int_t contiene el número.

  json_t *json_integer(json_int_t value)

  Devuelve un tipo número o NULL en caso de error

Jansson: Enteros

- int json_integer_set(const json_t *integer, json_int_t value)

  Establece el valor integer en el objeto JSON de tipo number. Devuelve 0 en caso correcto y -1 en caso de error.

- json_int_t json_integer_value(const json_t *integer)
- Devuelve el valor comprendido en el objeto JSON de tipo número.

Jansson: Real

- json_t *json_real(double value)

  Devuelve un JSON de tipo real o NULL en caso de error.

- int json_real_set(const json_t *real, double value)

  Establece el valor de tipo real “value”, en el objeto JSON de tipo real. Devuelve 0 en caso de éxito y -1 en caso de error.

Jansson: Real

- double json_real_value(const json_t *real)

  Devuelve el valor real asociado dentro del objeto JSON de tipo real

Jansson: Array

Un objeto de tipo JSON de tipo array es una colección ordenada de objetos JSON de tipo valor.

- json_t *json_array(void)

  Devuelve un nuevo objeto JSON de tipo array. Devuelve nulo en caso de error. Al inicializarlo el array está vacío.

Jansson: Array

- size_t json_array_size(const json_t *array)

  Devuelve el número de elementos del array o 0 si el array es nulo.

- json_t *json_array_get(const json_t *array, size_t index)

  Devuelve el elemento comprendido en la posición “index” del objeto JSON de tipo array. En caso de error, devuelve NULL.

Jansson: Array

- int json_array_set(json_t *array, size_t index, json_t *value)

  Reemplaza el elemento comprendido en la posición index por el objeto JSON de tipo valor “value”. Devuelve 0 en caso de éxito y -1 en caso de error.

Jansson: Array

- int json_array_append(json_t *array, json_t *value)

  Añade el elemento al final del array. Devuelve 0 en caso de éxito, -1 en caso de error.

- int json_array_insert(json_t *array, size_t index, json_t *value)

  Inserta el valor en la posición index del array, moviendo todos los valores una posición a la derecha. Devuelve 0 en caso de éxito y -1 en caso de error.

Jansson: Array

- int json_array_remove(json_t *array, size_t index)

  Borra el elemento comprendido en la posición index del array. Devuelve 0 en caso de éxito, -1 en caso de error.

Jansson: Array

- json_array_foreach(array, index, value)

  Itera por todos los elementos comprendidos en el array. Devolviendo cada elemento y la posición de él mismo.

Jansson: Array

```
/* array is a JSON array */
size_t index;
json_t *value;

json_array_foreach(array, index, value) {
    /* block of code that uses index and value */
}
```

Jansson: Objeto

- Un objeto JSON de tipo objeto es un diccionario de pares clave-valor, donde la clave es una cadena y el valor un objeto JSON de tipo valor.

- json_t *json_object(void)

  Devuelve un objeto JSON de tipo objeto y NULL en caso de error.

Jansson: Objeto

- size_t json_object_size(const json_t *object)

  Devuelve el número de elementos comprendidos en el objeto.

- json_t *json_object_get(const json_t *object, const char *key)

  Devuelve el valor correspondiente a la clave del objeto. Devuelve nulo en caso de error o si no encuentra la clave.

Jansson: Objeto

- int json_object_set(json_t *object, const char *key, json_t *value)

  Establece el par clave-valor en el objeto. Si ya existe un par con dicho clave-valor, se reemplaza con el nuevo valor. Devuelve 0 en caso de éxito, -1 en caso de error.

Jansson: Objeto

- int json_object_del(json_t *object, const char *key)

  Borra el par clave-valor del objeto si existe. Devuelve 0 en caso de éxito, y -1 en caso de error o si no encuentra la clave dentro del objeto.

Jansson: Objeto

- json_object_foreach(object, key, value)

  Itera por todas los pares clave-valor del objeto. Devolviendo cada clave y su valor relacionado.

Jansson: Objeto

```
/* obj is a JSON object */
const char *key;
json_t *value;

json_object_foreach(obj, key, value) {
    /* block of code that uses key and value */
}
```

Jansson: Objeto

- void *json_object_iter(json_t *object)

  Devuelve un iterador que puede ser usado para iterar por todas los pares clave-valor del objeto. Devuelve NULL si el objeto esta vacío.

Jansson: Objeto

- const char *json_object_iter_key(void *iter)

  Devuelve la clave asociado al par clave-valor actual del iterador.

- json_t *json_object_iter_value(void *iter)

  Devuelve el valor asociado al par clave-valor actual del iterador

Jansson: Objeto

- void *json_object_iter_next(json_t *object, void *iter)

  Devuelve un puntero al iterador el cuál a cambiado el par clave-valor actual por el siguiente. Devuelve NULL en el caso que no haya mas elementos.

Jansson: Objeto

```
/* obj is a JSON object */
const char *key;
json_t *value;

void *iter = json_object_iter(obj);
while(iter)
{
    key = json_object_iter_key(iter);
    value = json_object_iter_value(iter);
    /* use key and value ... */
    iter = json_object_iter_next(obj, iter);
}
```

Jansson: Error

- Jansson usa una estructura llamada json_error_t para almacenar útil para reportar errores al usuario.

  Dicha estructura contiene una serie de atributos como:

  - char text[]: El cuál contiene la descripción del error
  - char source[]: El origen del error

Jansson: Error

- Int line: El número de línea donde se ha producido el error
- Int column: El número de columna donde se ha producido el error
- size_t position: La posición en bytes donde comienza el error.

Jansson: Error

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

    json = json_load_file("/path/to/file.json", 0, &error);
    if(!json) {
        /* the error variable contains error information */
    }
    ...
}
```

Jansson: Importar

json_t *json_loadb(const char *buffer, size_t buflen, size_t flags, json_error_t *error)

- Crea el árbol a partir del buffer el cuál tiene la información en formato JSON
- Cuyo tamaño es buflen
- Devuelve el array o el objeto que contiene la información parseada. En caso de error, se almacenará la causa en la estructura error.

Jansson: Importar

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

- Crear el árbol a partir de un stream de entrada en formato JSON
- Devuelve el array o el objeto que contiene la información parseada. En caso de error, se almacenará la causa en la estructura error.

Jansson: Importar

- Las funciones anteriores pueden utilizar flags para configurar la creación de nuestros árboles, como por ejemplo:
  - JSON_REJECT_DUPLICATES: Emite un error si existen clave duplicadas en la entrada que vamos a importar
  - JSON_DECODE_ANY: Por defecto, nuestras funciones de importación esperan un objeto o un array. Con este flag activo, parseará todo elemento.

Jansson: Importar

- JSON_DECODE_INT_AS_REAL: Si este flag esta activo, Jansson importará todos los valores como tipos reales. En caso contrario serán importados como enteros

Jansson: Exportar

- char *json_dumps(const json_t *root, size_t flags)

  Devuelve la representación JSON del arbol que pasamos como “root”. La devolución debe ser liberada usando free.

Jansson: Exportar

- int json_dumpf(const json_t *root, FILE *output, size_t flags)

  Escribe la representación JSON del arbol “root” en un stream de salida.

Jansson: Exportar

- Las funciones anteriores pueden utilizar flags para configurar el formato de la salida, como por ejemplo:
  - JSON_INDENT(n): Sirve para exportar la salida añadiendo identaciones y facilitando así una visualización mas fácil para los usuarios.
  - JSON_COMPACT: Establece que no exista ningún espacio entre nuestros pares clave-valor ni entre la clave y el valor.

Jansson: Exportar

- JSON_SORT_KEYS: Ordena los pares claves-valor a partir de su clave.
- JSON_REAL_PRECISION(n): Representa los números reales con n números decimales
- JSON_ENSURE_ASCII: El árbol será exportado usando solo codificación de tipo ASCII

Ejemplos

Bibliografía

- API de Jansson (http://jansson.readthedocs.org/en/latest/apiref.html)
- Web de JSON (http://json.org/)
- Web/herramienta para validar salidas JSON (http://jsonlint.com/)

Libsqlite: Biblioteca para uso de bases de datos sqlite.

- Introducción.
- Ejemplo 1: Conexión con una base de datos y creación de una tabla.
- Ejemplo 2: Añadir, borrar y actualizar datos en una base datos.

Introducción

- SQLite es un sistema de gestión de bases de datos relacional
- Diseñado por D. Richard Hipp
- Programado en 17 de agosto de 2000
- Programado en lenguaje C

Software necesario

- sqlite3
- libsqlite3-*

Usar la biblioteca

- #include <sqlite3.h>

  Biblioteca que contiene todas las funciones para usar sqlite en C

- Gcc -o main main.c -lsqlite3

  Flag para activar el uso de sqlite3 en nuestro programa

Conectar BD

- sqlite3_open(const char *database_name, sqlite3 *db);

  Función para conectar la base de datos y crea un objeto sqlite3 con la conexión.

Ejecutar sentencias SQL

- int sqlite3_exec(
  sqlite3*, /* Base de datos abierta */
  const char *sql, /* Sentencia SQL */
  int (*callback)(void*,int,char**,char**), /* funcion Callback */
  void *, /* Argumento para callback */
  char **errmsg /* Mensaje de error */
  );

- Devuelve SQLITE_OK si la sentencia SQL ha sido bien ejecutada

Cerrar BD

- sqlite3_close(sqlite3 *db);

  Cierra la BD abierta

Ejemplos

Ejercicios