JSON Parser

JSON Parser Version 0.0.2
Mark Wharton, 28th October 2010

1. Introduction

Features

Parse JSON and optionally build the parse tree with callbacks:

Dependencies

- ragel for building jsonparser library
- tokyocabinet for json test application

Installation

Run the configuration script.

```
./configure
```

Build the library and programs.

```
make
```

Install the library and programs.

```
sudo make install
```

Using the JSON Library

```
gcc -I/usr/local/include jsonapp.c -o jsonapp -L/usr/local/lib -ljsonparser
```

Sample JSON Application

```
#include <jsonparser.h>

char *jsonTypes[] = { NULL, "string", "number", "{}", "[]", "true", "false", "null" };

bool jsonAddElement(void *userData, void *item, JSONParserValue *value)
{
  char *elementValue = value->string; /* string and number */
  elementValue = elementValue ? elementValue : jsonTypes[value->type];
  return fprintf(stdout, "element: %s\n", elementValue) > 0;
```

```
}

bool jsonSetMember(void *userData, void *item, char *name, JSONParserValue *value)
{
  char *memberValue = value->string; /* string and number */
  memberValue = memberValue ? memberValue : jsonTypes[value->type];
  return fprintf(stdout, "member: %s = %s\n", name, memberValue) > 0;
}

int main(int argc, char **argv)
{
  JSONParser parser = createJSONParser(NULL);
  if (parser) {
    jsonParserConfigureBuilders(parser, jsonAddElement, NULL, jsonSetMember);
    JSONParserBuffer *buffer = createJSONParserBuffer(JSON_PARSER_BUFFER_SIZE);
    if (buffer) {
      if (!jsonParserParseStream(parser, buffer, NULL, NULL, NULL)) {
        fprintf(stderr, "Error: parser error: %d %s (line %d)\n",
          jsonParserGetErrorCode(parser), jsonParserGetErrorString(parser),
          jsonParserGetCurrentLine(parser));
        return 1;
      }
      jsonParserBufferFree(buffer);
      buffer = NULL;
    } else {
      fprintf(stderr, "Error: could not allocate buffer: %lld\n", (long long)buffer->size);
      return 1;
    }
    jsonParserFree(parser);
    parser = NULL;
  } else {
    fprintf(stderr, "Error: could not create parser\n");
    return 1;
  }
  return 0;
}
```

Values

The following value types are defined:

Value Types	Field
JSON_PARSER_VALUE_TYPE_NONE	
JSON_PARSER_VALUE_TYPE_STRING [1]	string
JSON_PARSER_VALUE_TYPE_NUMBER [1,2]	number | string
JSON_PARSER_VALUE_TYPE_OBJECT	item
JSON_PARSER_VALUE_TYPE_ARRAY	item
JSON_PARSER_VALUE_TYPE_TRUE	
JSON_PARSER_VALUE_TYPE_FALSE	
JSON_PARSER_VALUE_TYPE_NULL	

Special Notes:

1. The string must be copied, value string destroyed on callback return.
2. The number type provides the value as `double` and `string` for convenience.

2. JSON Parser API

The **JSON Parser API** library provides the following functions:

- `createJSONParser`
- `createJSONParserBuffer`
- `createJSONParserString`
- `jsonParserBufferFree`
- `jsonParserConfigureBuilders`
- `jsonParserConfigureWriters`
- `jsonParserFree`
- `jsonParserGetCurrentLine`

- jsonParserGetErrorCode
- jsonParserGetErrorString
- jsonParserGetUserData
- jsonParserParseStream
- jsonParserParseString
- jsonParserSetUserData
- jsonParserStandardInputReader
- jsonParserStringAppend
- jsonParserStringFree

---

createJSONParser

```
JSONParser JSONPARSERAPI createJSONParser(JSONParserConfig *config)
```

Description

Create a new `parser` object. The parser must be freed with `jsonParserFree` when no longer needed.

Parameters

- **`config :`** `JSONParserConfig *`
  (optional) The config object. If the value is NULL the standard `emptyJSONParserConfig` config is used. With `emptyJSONParserConfig` all options are set to `false` and the callbacks set to `NULL`.

Return Value

- **`parser :`** `JSONParser`
  The new parser object. Note: The value will be `NULL` if memory could not be allocated.

---

createJSONParserBuffer

```
JSONParserBuffer JSONPARSERAPI *createJSONParserBuffer(size_t size)
```

Description

Create a new `buffer` object. The buffer must be freed with `jsonParserBufferFree` when no longer needed.

Parameters

- **`size :`** `size_t`
  The buffer size.

Return Value

- **`buffer :`** `JSONParserBuffer *`
  The new buffer object. Note: The value will be `NULL` if memory could not be allocated.

---

createJSONParserString

```
char JSONPARSERAPI *createJSONParserString(const char *data, size_t size)
```

Description

Create a new `string` object with text. The string must be freed with `jsonParserStringFree` when no longer needed.

Parameters

- **`data :`** `const char *`
  The text buffer data.
- **`size :`** `size_t`
  The text buffer size.

Return Value

- **`string :`** `char *`
  The new string object. Note: The value will be `NULL` if memory could not be allocated.

---

jsonParserBufferFree

```
void JSONPARSERAPI jsonParserBufferFree(JSONParserBuffer *buffer)
```

Description

Free the memory allocated to `buffer`. The buffer must be a valid buffer created with an earlier call to createJSONParserBuffer. The buffer value must not be used after it is freed. It is good practice to set the buffer value to `NULL` after calling this function.

Parameters

- **buffer :** `JSONParserBuffer *`
  The buffer to be freed.

Return Value

- **none :** `void`

---

jsonParserConfigureBuilders

```
void JSONPARSERAPI jsonParserConfigureBuilders(JSONParser parser, BuildAddElementFunc
buildAddElement, BuildNewItemFunc buildNewItem, BuildSetMemberFunc buildSetMember)
```

Description

Configure the builder callbacks for the parser. These callbacks are applied to a copy of the original config object (if any) which was passed in the call to createJSONParser. The original config object remains unchanged.

The builder callback functions are declared as follows:

```
typedef bool (*BuildAddElementFunc)(void *userData, void *item, JSONParserValue *value);
typedef bool (*BuildNewItemFunc)(void *userData, JSONParserValue *value/*, int depth*/);
typedef bool (*BuildSetMemberFunc)(void *userData, void *item, char *name, JSONParserValue *value);
```

Return `true` on success and `false` on failure. Record any specific error details in `userData` if they are needed.

Note: Parser stack overflow (`JSON_PARSER_ERROR_PSTACK`) is a product of the builder functions. If support for builder functions is removed then the stack issues will also go away.

Parameters

- **parser :** `JSONParser`
  The parser object.
- **buildAddElement :** `BuildAddElementFunc`
  Builder callback to add an element to the array item.
- **buildNewItem :** `BuildNewItemFunc`
  Builder callback to make a new array or object item.
- **buildSetMember :** `BuildSetMemberFunc`
  Builder callback to set a member of the object item.

Return Value

- **none :** `void`

---

jsonParserConfigureWriters

```
void JSONPARSERAPI jsonParserConfigureWriters(JSONParser parser, WriteArrayElementFunc
writeArrayElement, WriteObjectMemberFunc writeObjectMember, WriteStartFunc writeStart,
WriteStartArrayFunc writeStartArray, WriteStartObjectFunc writeStartObject, WriteStopFunc
writeStop, WriteStopArrayFunc writeStopArray, WriteStopObjectFunc writeStopObject)
```

Description

Configure the writer callbacks for the parser. These callbacks are applied to a copy of the original config object (if any) which was passed in the call to createJSONParser. The original config object remains unchanged.

The writer callback functions are declared as follows:

```
typedef bool (*WriteArrayElementFunc)(void *userData, JSONParserValue *value);
typedef bool (*WriteObjectMemberFunc)(void *userData, char *name, JSONParserValue *value);
```

```
typedef bool (*WriteStartFunc)(void *userData);
typedef bool (*WriteStartArrayFunc)(void *userData, char *name);
typedef bool (*WriteStartObjectFunc)(void *userData, char *name);
typedef bool (*WriteStopFunc)(void *userData);
typedef bool (*WriteStopArrayFunc)(void *userData);
typedef bool (*WriteStopObjectFunc)(void *userData);
```

Return `true` on success and `false` on failure. Record any specific error details in `userData` if they are needed.

Parameters

- **`parser`** : `JSONParser`
  The parser object.
- **`writeArrayElement`** : `WriteArrayElementFunc`
  Writer callback to add an element to the array context. Primitive values only, array and object values are handled separately.
- **`writeObjectMember`** : `WriteObjectMemberFunc`
  Writer callback to set a member of the object context. Primitive values only, array and object values are handled separately.
- **`writeStart`** : `WriteStartFunc`
  Writer callback to initialize the document context.
- **`writeStartArray`** : `WriteStartArrayFunc`
  Writer callback to push a new array context. If the `name` parameter is `NULL` the parent context is an array, otherwise it is an object.
- **`writeStartObject`** : `WriteStartObjectFunc`
  Writer callback to push a new object context. If the `name` parameter is `NULL` the parent context is an array, otherwise it is an object.
- **`writeStop`** : `WriteStopFunc`
  Writer callback to terminate the document context.
- **`writeStopArray`** : `WriteStopArrayFunc`
  Writer callback to pop the array and restore the parent context.
- **`writeStopObject`** : `WriteStopObjectFunc`
  Writer callback to pop the object and restore the parent context.

Return Value

- **`none`** : `void`

---

jsonParserFree

```
void JSONPARSERAPI jsonParserFree(JSONParser parser)
```

Description

Free the memory allocated to `parser`. The parser must be a valid parser created with an earlier call to createJSONParser. The parser value must not be used after it is freed. It is good practice to set the parser value to `NULL` after calling this function.

Parameters

- **`parser`** : `JSONParser`
  The parser to be freed.

Return Value

- **`none`** : `void`

---

jsonParserGetCurrentLine

```
int JSONPARSERAPI jsonParserGetCurrentLine(JSONParser parser)
```

Description

Return the current line of the parser. Line numbers start from `1` and continue to the end of the document. Before the parser has started the value will be `0`, on completion the value will be the total number of lines in the document. When an error occurs the value will be the line the error occurred on.

Parameters

- **`parser`** : `JSONParser`

The parser object.

Return Value

- **value :** `int`
  The current line.

---

jsonParserGetErrorCode

```
enum JSON_PARSER_ERROR JSONPARSERAPI jsonParserGetErrorCode(JSONParser parser)
```

Description

Return the error code of the parser. The error code is set when an error occurs during parsing. Errors which occur during reading or writing are caught in the parser and returned as `JSON_PARSER_ERROR_READER` and `JSON_PARSER_ERROR_WRITER` respectively. Readers and writers should record any specific error details in `userData` if they are needed. The `JSON_PARSER_ERROR_PARSER` error indicates a syntax error, `JSON_PARSER_ERROR_MEMORY` indicates an out of memory error, `JSON_PARSER_ERROR_BUFFER` indicates a token in the stream was too big for the user supplied buffer.

Errors	Description
`JSON_PARSER_ERROR_UNKNOWN`	An unknown error occurred.
`JSON_PARSER_ERROR_NONE`	No error.
`JSON_PARSER_ERROR_BUFFER`	Token too big for buffer.
`JSON_PARSER_ERROR_MEMORY`	An out of memory error occurred.
`JSON_PARSER_ERROR_PARSER`	Input could not be parsed.
`JSON_PARSER_ERROR_PSTACK`	Parser stack overflow.
`JSON_PARSER_ERROR_READER`	An error occurred with the reader.
`JSON_PARSER_ERROR_WRITER`	An error occurred with the writer.

The `JSON_PARSER_ERROR_NONE` error is equal to `0`, `JSON_PARSER_ERROR_UNKNOWN` is less than `0`, all other errors are greater than `0`.

Parameters

- **parser :** `JSONParser`
  The parser object.

Return Value

- **value :** `enum JSON_PARSER_ERROR`
  The error code.

---

jsonParserGetErrorString

```
const char JSONPARSERAPI *jsonParserGetErrorString(JSONParser parser)
```

Description

Return the error string associated with the error code of the parser. See `jsonParserGetErrorCode` for details.

Parameters

- **parser :** `JSONParser`
  The parser object.

Return Value

- **value :** `const char *`
  The error string.

---

jsonParserGetUserData

```
void JSONPARSERAPI *jsonParserGetUserData(JSONParser parser)
```

Description

Return user data associated with the parser in an earlier call to jsonParserSetUserData. The `void *userData` should be recast to use. Callback functions are provided the same `void *userData`. Macros can make using the user data easier, for example:

```
#define userDataRowCount (((JSON2HTMLUserDataPtr)userData)->rowCount)
```

Parameters

- **parser :** JSONParser
  The parser object.

Return Value

- **value :** void *
  The user data.

---

jsonParserParseStream

```
bool JSONPARSERAPI jsonParserParseStream(JSONParser parser, JSONParserBuffer *buffer, ReaderFunc reader, void *item, char *name)
```

Description

Parse the reader input. Callbacks are used to build the JSON array/elements and object/members parse tree.

The `ReaderFunc` reader callback is declared as follows:

```
typedef bool (*ReaderFunc)(void *userData, JSONParserBuffer *buffer, size_t *size);

bool jsonParserStandardInputReader(void *userData, JSONParserBuffer *buffer, size_t *size) {
    *size = fread((char *)buffer->data, 1, buffer->size, stdin);
    return ferror(stdin) == 0;
}
```

Return `true` on success and `false` on failure. Record any specific error details in `userData` if they are needed.

Parameters

- **parser :** JSONParser
  The parser object.
- **buffer :** JSONParserBufferPtr
  The buffer for the parser. Buffer management functions are performed automatically.
- **reader :** ReaderFunc
  (optional) The parser calls this functions whenever it needs to fill the buffer. If the value is NULL the default jsonParserStandardInputReader callback function is used.
- **item :** void *
  The root object for the parse value (i.e. the top most result of the parse).
- **name :** char *
  (optional) The member name in the root object. If the value is NULL the default name "`root`" is used.

Return Value

- **success :** bool
  Returns `true` on success and `false` on failure. Call the jsonParserGetErrorCode and jsonParserGetErrorString functions for information about the failure. Possible errors include: `JSON_PARSER_ERROR_BUFFER`, `JSON_PARSER_ERROR_MEMORY`, `JSON_PARSER_ERROR_PARSER`, `JSON_PARSER_ERROR_PSTACK`, and `JSON_PARSER_ERROR_READER`.

---

jsonParserParseString

```
bool JSONPARSERAPI jsonParserParseString(JSONParser parser, char *string, void *item, char *name)
```

Description

Parse the string. Callbacks are used to build the JSON array/elements and object/members parse tree.

Parameters

- **parser :** JSONParser

The parser object.
- **string :** `char *`
  The string to parse.
- **item :** `void *`
  The root object for the parse value (i.e. the top most result of the parse).
- **name :** `char *`
  (optional) The member name in the root object. If the value is NULL the default name "`root`" is used.

Return Value

- **success :** `bool`
  Returns `true` on success and `false` on failure. Call the `jsonParserGetErrorCode` and `jsonParserGetErrorString` functions for information about the failure. Possible errors include: `JSON_PARSER_ERROR_MEMORY`, `JSON_PARSER_ERROR_PARSER`, and `JSON_PARSER_ERROR_PSTACK`.

---

jsonParserSetUserData

```
void JSONPARSERAPI *jsonParserSetUserData(JSONParser parser, void *userData)
```

Description

Associate user data with the parser for callback functions. See also `jsonParserGetUserData`.

Parameters

- **parser :** `JSONParser`
  The parser object.
- **userData :** `void *`
  The user data.

Return Value

- **none :** `void`

---

jsonParserStandardInputReader

```
bool jsonParserStandardInputReader(void *userData, JSONParserBuffer *buffer, size_t *size)
```

Description

The default reader callback for filling the buffer with `stdin`. See also `jsonParserParseStream`.

Parameters

- **userData :** `void *`
  The user data associated with the parser.
- **buffer :** `JSONParserBuffer *`
  The buffer to read into.
- **size :** `size_t *`
  Pointer for returning the amount of bytes read into the buffer.

Return Value

- **success :** `bool`
  Returns `true` on success and `false` on failure. No specific error details are recorded for `JSON_PARSER_ERROR_READER`.

---

jsonParserStringAppend

```
char JSONPARSERAPI *jsonParserStringAppend(char *string, const char *data, size_t size)
```

Description

Append text to `string`. The string must be a valid string created with an earlier call to createJSONParserString. The string must be freed with `jsonParserStringFree` when no longer needed.

Parameters

- **string :** `char *`

(optional) The string object. If the value is NULL a new string will be created.
- **data :** `const char *`
  The text buffer data.
- **size :** `size_t`
  The text buffer size.

Return Value

- **string :** `char *`
  The (possibly) new string object. Note: The value will be `NULL` if memory could not be allocated.

---

jsonParserStringFree

```
void JSONPARSERAPI jsonParserStringFree(char *string)
```

Description

Free the memory allocated to `string`. The string must be a valid string created with an earlier call to createJSONParserString. The string value must not be used after it is freed. It is good practice to set the string value to `NULL` after calling this function.

Parameters

- **string :** `char *`
  The string to be freed.

Return Value

- **none :** `void`

---

3. Release Notes

28 October 2010 - 0.0.2

- Added support for builders and writers.

27 October 2010 - 0.0.1

- Initial release for discussion.