



Hi3861 V100 / Hi3861L V100 cJSON

Development Guide

Issue	01
Date	2020-04-30

Copyright © HiSilicon (Shanghai) Technologies Co., Ltd. 2020. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon (Shanghai) Technologies Co., Ltd.

Trademarks and Permissions



HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided “AS IS” without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road,
Bantian, Longgang District,
Shenzhen 518129 P. R. China

Website: <https://www.hisilicon.com/en/>

Email: support@hisilicon.com



About This Document

Purpose

This document describes the code samples and API description of JSON parsing development based on cJSON 1.7.12.

For details about cJSON 1.7.12, see <https://github.com/DaveGamble/cJSON/blob/master/README.md>.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3861	V100
Hi3861L	V100


Intended Audience

The document is intended for:




- Technical support engineers
- Software development engineers

Symbol Conventions

The following table describes the symbols that may be found in this document.

Symbol	Description
	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.



Symbol	Description
 WARNING	Indicates a hazard with a medium level of risk which, if not avoided, could result in death or serious injury.
 CAUTION	Indicates a hazard with a low level of risk which, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.
 NOTE	Supplements the important information in the main text. NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration.

Change History

Issue	Date	Change Description
01	2020-04-30	This issue is the first official release.
00B01	2020-01-15	This issue is the first draft release.



Contents

About This Document.....	i
1 API Description.....	1
1.1 Data Structures.....	1
1.1.1 Struct cJSON.....	1
1.1.2 Struct cJSON_Hooks.....	1
1.2 APIs.....	2
2 Development Guidance.....	7
2.1 Printing a JSON String.....	7
2.2 Parsing a JSON String.....	8



1 API Description

NOTE

cJSON 1.7.12 is an open-source JSON parser. Only some of the data structures and APIs are included in this document. For details about the excluded, see the cJSON website.

[1.1 Data Structures](#)

[1.2 APIs](#)

1.1 Data Structures

1.1.1 Struct cJSON

```
/* The cJSON structure: */
typedef struct cJSON
{
    struct cJSON *next;
    struct cJSON *prev;
    struct cJSON *child;
    int type;
    char *valuestring;
    /* writing to valueint is DEPRECATED, use cJSON_SetNumberValue instead */
    int valueint;
    double valuedouble;
    char *string;
} cJSON;
```

1.1.2 Struct cJSON_Hooks

```
typedef struct cJSON_Hooks
{
    /* malloc/free are CDECL on Windows regardless of the default calling convention of the
    compiler, so ensure the hooks allow passing those functions directly. */
    void *(CJSON_CDECL *malloc_fn)(size_t sz);
    void (CJSON_CDECL *free_fn)(void *ptr);
} cJSON_Hooks;
```



1.2 APIs

cJSON API	Description
CJSON_PUBLIC(const char*) cJSON_Version(void);	Obtains the cJSON version.
CJSON_PUBLIC(void) cJSON_InitHooks(cJSON_Hooks* hooks);	Resets cJSON hooks.
CJSON_PUBLIC(cJSON *) cJSON_Parse(const char *value);	Parses a string into a cJSON structure.
CJSON_PUBLIC(cJSON *) cJSON_ParseWithOpts(const char *value, const char **return_parse_end, cJSON_bool require_null_terminated);	
CJSON_PUBLIC(char *) cJSON_Print(const cJSON *item);	Prints a cJSON structure to a string.
CJSON_PUBLIC(char *) cJSON_PrintUnformatted(const cJSON *item);	Prints a cJSON structure to a string without any formatting.
CJSON_PUBLIC(char *) cJSON_PrintBuffered(const cJSON *item, int prebuffer, cJSON_bool fmt);	Prints a cJSON structure to a string using a buffered strategy.
CJSON_PUBLIC(cJSON_bool) cJSON_PrintPreallocated(cJSON *item, char *buffer, const int length, const cJSON_bool format);	Prints a cJSON structure to a string using a buffer already allocated in memory with a given length.
CJSON_PUBLIC(void) cJSON_Delete(cJSON *c);	Deletes a cJSON structure.
CJSON_PUBLIC(int) cJSON_GetArraySize(const cJSON *array);	Returns the number of items in an array or object.
CJSON_PUBLIC(cJSON *) cJSON_GetArrayItem(const cJSON *array, int index);	Returns item number index from an array or object.
CJSON_PUBLIC(cJSON *) cJSON_GetObjectItem(const cJSON * const object, const char * const string);	Gets item string from an object.
CJSON_PUBLIC(cJSON *) cJSON_GetObjectItemCaseSensitive(const cJSON * const object, const char * const string);	
CJSON_PUBLIC(cJSON_bool) cJSON_HasObjectItem(const cJSON *object, const char *string);	Checks if item string exists in an object.



cJSON API	Description
CJSON_PUBLIC(const char *) cJSON_GetErrorPtr(void);	Returns a pointer to the parse error.
CJSON_PUBLIC(char *) cJSON_GetStringValue(cJSON *item);	Checks if the item is a string and return its valuestring (or NULL).
CJSON_PUBLIC(cJSON_bool) cJSON_IsInvalid(const cJSON * const item);	Checks the type of an item.
CJSON_PUBLIC(cJSON_bool) cJSON_IsFalse(const cJSON * const item);	
CJSON_PUBLIC(cJSON_bool) cJSON_IsTrue(const cJSON * const item);	
CJSON_PUBLIC(cJSON_bool) cJSON_IsBool(const cJSON * const item);	
CJSON_PUBLIC(cJSON_bool) cJSON_IsNull(const cJSON * const item);	
CJSON_PUBLIC(cJSON_bool) cJSON_IsNumber(const cJSON * const item);	
CJSON_PUBLIC(cJSON_bool) cJSON_IsString(const cJSON * const item);	
CJSON_PUBLIC(cJSON_bool) cJSON_IsArray(const cJSON * const item);	
CJSON_PUBLIC(cJSON_bool) cJSON_IsObject(const cJSON * const item);	
CJSON_PUBLIC(cJSON_bool) cJSON_IsRaw(const cJSON * const item);	
CJSON_PUBLIC(cJSON *) cJSON_CreateNull(void);	Creates a cJSON item of the appropriate type.
CJSON_PUBLIC(cJSON *) cJSON_CreateTrue(void);	
CJSON_PUBLIC(cJSON *) cJSON_CreateFalse(void);	
CJSON_PUBLIC(cJSON *) cJSON_CreateBool(cJSON_bool boolean);	
CJSON_PUBLIC(cJSON *) cJSON_CreateNumber(double num);	
CJSON_PUBLIC(cJSON *) cJSON_CreateString(const char *string);	



cJSON API	Description
CJSON_PUBLIC(cJSON *) cJSON_CreateRaw(const char *raw);	
CJSON_PUBLIC(cJSON *) cJSON_CreateArray(void);	
CJSON_PUBLIC(cJSON *) cJSON_CreateObject(void);	
CJSON_PUBLIC(cJSON *) cJSON_CreateIntArray(const int *numbers, int count);	
CJSON_PUBLIC(cJSON *) cJSON_CreateFloatArray(const float *numbers, int count);	Creates an array of count items.
CJSON_PUBLIC(cJSON *) cJSON_CreateDoubleArray(const double *numbers, int count);	
CJSON_PUBLIC(cJSON *) cJSON_CreateStringArray(const char **strings, int count);	
CJSON_PUBLIC(cJSON *) cJSON_CreateStringArray(const char **strings, int count);	
CJSON_PUBLIC(void) cJSON_AddItemToArray(cJSON *array, cJSON *item);	Appends an item to the specified array or object.
CJSON_PUBLIC(void) cJSON_AddItemToObject(cJSON *object, const char *string, cJSON *item);	
CJSON_PUBLIC(void) cJSON_AddItemToObjectCS(cJSON *object, const char *string, cJSON *item);	
CJSON_PUBLIC(void) cJSON_AddItemReferenceToArray(cJSON *array, cJSON *item);	
CJSON_PUBLIC(void) cJSON_AddItemReferenceToObject(cJSON *object, const char *string, cJSON *item);	
CJSON_PUBLIC(cJSON *) cJSON_DetachItemViaPointer(cJSON *parent, cJSON * const item);	Removes items from arrays or objects.
CJSON_PUBLIC(cJSON *) cJSON_DetachItemFromArray(cJSON *array, int which);	
CJSON_PUBLIC(void) cJSON_DeleteItemFromArray(cJSON *array, int which);	



cJSON API	Description
<code>cJSON_PUBLIC(cJSON *) cJSON_DetachItemFromObject(cJSON *object, const char *string);</code>	
<code>cJSON_PUBLIC(cJSON *) cJSON_DetachItemFromObjectCaseSensi- tive(cJSON *object, const char *string);</code>	
<code>cJSON_PUBLIC(void) cJSON_DeleteItemFromOb- ject(cJSON *object, const char *string);</code>	
<code>cJSON_PUBLIC(void) cJSON_DeleteItemFromOb- jectCaseSensitive(cJSON *object, const char *string);</code>	
<code>cJSON_PUBLIC(void) cJSON_InsertItemInArray(cJSON *array, int which, cJSON *newitem);</code>	Updates items.
<code>cJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemViaPointer(cJSON * const parent, cJSON * const item, cJSON * replacement);</code>	
<code>cJSON_PUBLIC(void) cJSON_ReplaceItemInAr- ray(cJSON *array, int which, cJSON *newitem);</code>	
<code>cJSON_PUBLIC(void) cJSON_ReplaceItemInOb- ject(cJSON *object, const char *string, cJSON *newitem);</code>	
<code>cJSON_PUBLIC(void) cJSON_ReplaceItemInOb- jectCaseSensitive(cJSON *object, const char *string, cJSON *newitem);</code>	
<code>cJSON_PUBLIC(cJSON *) cJSON_Duplicate(const cJSON *item, cJSON_bool recurse);</code>	Duplicates a cJSON structure.
<code>cJSON_PUBLIC(cJSON_bool) cJSON_Compare(const cJSON * const a, const cJSON * const b, const cJSON_bool case_sensitive);</code>	Compares two cJSON structures.
<code>cJSON_PUBLIC(void) cJSON_Minify(char *json);</code>	Minifies strings by removing blank characters.
<code>cJSON_PUBLIC(cJSON*) cJSON_AddNullToObject(cJSON * const object, const char * const name);</code>	Creates and adds items to an object. Returns the added item or NULL on failure.
<code>cJSON_PUBLIC(cJSON*) cJSON_AddTrueToObject(cJSON * const object, const char * const name);</code>	



cJSON API	Description
CJSON_PUBLIC(cJSON*) cJSON_AddFalseToObject(cJSON * const object, const char * const name);	
CJSON_PUBLIC(cJSON*) cJSON_AddBoolToObject(cJSON * const object, const char * const name, const cJSON_bool boolean);	
CJSON_PUBLIC(cJSON*) cJSON_AddNumberToObject(cJSON * const object, const char * const name, const double number);	
CJSON_PUBLIC(cJSON*) cJSON_AddStringToObject(cJSON * const object, const char * const name, const char * const string);	
CJSON_PUBLIC(cJSON*) cJSON_AddRawToObject(cJSON * const object, const char * const name, const char * const raw);	
CJSON_PUBLIC(cJSON*) cJSON_AddObjectToObject(cJSON * const object, const char * const name);	
CJSON_PUBLIC(cJSON*) cJSON_AddArrayToObject(cJSON * const object, const char * const name);	
cJSON_SetIntValue(object, number)	Sets an integer value. It is propagated to valuedouble at the same time.
cJSON_SetNumberValue(object, number)	Sets a valuedouble. It is propagated to an integer at the same time.
cJSON_ArrayForEach(element, array)	Iterates over an array or object
CJSON_PUBLIC(double) cJSON_SetNumberHelper(cJSON *object, double number);	Sets an integer and valuedouble. If the argument exceeds the limit, the maximum value is used.
CJSON_PUBLIC(void *) cJSON_malloc(size_t size);	Mallocs/Frees objects using the malloc/free functions that have been set with cJSON_InitHooks.
CJSON_PUBLIC(void) cJSON_free(void *object);	



2 Development Guidance

The official code samples for printing and parsing a JSON string are provided.

[2.1 Printing a JSON String](#)

[2.2 Parsing a JSON String](#)

2.1 Printing a JSON String

```
//NOTE: Returns a heap allocated string, you are required to free it after use.
char *create_monitor_with_helpers(void)
{
    const unsigned int resolution_numbers[3][2] = {
        {1280, 720},
        {1920, 1080},
        {3840, 2160}
    };
    char *string = NULL;
    cJSON *resolutions = NULL;
    size_t index = 0;

    cJSON *monitor = cJSON_CreateObject();

    if (cJSON_AddStringToObject(monitor, "name", "Awesome 4K") == NULL)
    {
        goto end;
    }

    resolutions = cJSON_AddArrayToObject(monitor, "resolutions");
    if (resolutions == NULL)
    {
        goto end;
    }

    for (index = 0; index < (sizeof(resolution_numbers) / (2 * sizeof(int))); ++index)
    {
        cJSON *resolution = cJSON_CreateObject();

        if (cJSON_AddNumberToObject(resolution, "width", resolution_numbers[index][0]) ==
        NULL)
        {
            goto end;
        }
    }
}
```



```
        if(cJSON_AddNumberToObject(resolution, "height", resolution_numbers[index][1]) ==
        NULL)
        {
            goto end;
        }

        cJSON_AddItemToArray(resolutions, resolution);
    }

    string = cJSON_Print(monitor);
    if (string == NULL) {
        fprintf(stderr, "Failed to print monitor.\n");
    }

end:
    cJSON_Delete(monitor);
    return string;
}
```

2.2 Parsing a JSON String

```
/* return 1 if the monitor supports full hd, 0 otherwise */
int supports_full_hd(const char * const monitor)
{
    const cJSON *resolution = NULL;
    const cJSON *resolutions = NULL;
    const cJSON *name = NULL;
    int status = 0;
    cJSON *monitor_json = cJSON_Parse(monitor);
    if (monitor_json == NULL)
    {
        const char *error_ptr = cJSON_GetErrorPtr();
        if (error_ptr != NULL)
        {
            fprintf(stderr, "Error before: %s\n", error_ptr);
        }
        status = 0;
        goto end;
    }

    name = cJSON_GetObjectItemCaseSensitive(monitor_json, "name");
    if (cJSON_IsString(name) && (name->valstring != NULL))
    {
        printf("Checking monitor \"%s\"\n", name->valstring);
    }

    resolutions = cJSON_GetObjectItemCaseSensitive(monitor_json, "resolutions");
    cJSON_ArrayForEach(resolution, resolutions)
    {
        cJSON *width = cJSON_GetObjectItemCaseSensitive(resolution, "width");
        cJSON *height = cJSON_GetObjectItemCaseSensitive(resolution, "height");

        if (!cJSON_IsNumber(width) || !cJSON_IsNumber(height))
        {
            status = 0;
            goto end;
        }

        if ((width->valuedouble == 1920) && (height->valuedouble == 1080))
        {

```



```
        status = 1;  
        goto end;  
    }  
}  
  
end:  
    cJSON_Delete(monitor_json);  
    return status;  
}
```