



Hi3861V100 / Hi3861LV100 cJson

开发指南

文档版本 01

发布日期 2020-04-30

版权所有 © 上海海思技术有限公司2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://www.hisilicon.com/cn/>

客户服务邮箱： support@hisilicon.com



前言

概述

本文档主要介绍基于cJson 1.7.12的Json报文解析功能开发实现示例，以及接口说明。

cJson 1.7.12的详细说明请参见官方说明文档：<https://github.com/DaveGamble/cJSON/blob/master/README.md>

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3861	V100
Hi3861L	V100



读者对象

本文档主要适用于以下工程师：



- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。



符号	说明
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2020-04-30	第一次正式版本发布。
00B01	2020-01-15	第一次临时版本发布。



目录

前言.....	i
1 API 接口说明.....	1
1.1 结构体说明.....	1
1.1.1 struct cJSON.....	1
1.1.2 struct cJSON_Hooks.....	1
1.2 API 列表.....	2
2 开发指南.....	7
2.1 创建 Json 字符串示例.....	7
2.2 解析 Json 字符串示例.....	8



1 API 接口说明

说明

cJSON 1.7.12为开源软件，以下为部分结构体和API描述，其余部分请参考官方说明。

[1.1 结构体说明](#)

[1.2 API列表](#)

1.1 结构体说明

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 API 列表

cJSON API	说明
CJSON_PUBLIC(const char*) cJSON_Version(void);	获得cJSON的版本。
CJSON_PUBLIC(void) cJSON_InitHooks(cJSON_Hooks* hooks);	初始化cJSON_Hooks。
CJSON_PUBLIC(cJSON *) cJSON_Parse(const char *value);	将字符串解析成cJSON结构体。
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);	将cJSON结构体转换成格式化的字符串。
CJSON_PUBLIC(char *) cJSON_PrintUnformatted(const cJSON *item);	将cJSON结构体转换成未格式化的字符串。
CJSON_PUBLIC(char *) cJSON_PrintBuffered(const cJSON *item, int prebuffer, cJSON_bool fmt);	将cJSON结构体转换为字符串，通过入参传入预估结果尺寸和是否格式化。
CJSON_PUBLIC(cJSON_bool) cJSON_PrintPreallocated(cJSON *item, char *buffer, const int length, const cJSON_bool format);	将cJSON结构体转换为字符串，并存入预先申请的长度确定的缓冲空间，通过传入参数决定是否格式化。
CJSON_PUBLIC(void) cJSON_Delete(cJSON *c);	删除cJSON结构体。
CJSON_PUBLIC(int) cJSON_GetArraySize(const cJSON *array);	返回Array类型的大小，对Object类型也有效。
CJSON_PUBLIC(cJSON *) cJSON_GetArrayItem(const cJSON *array, int index);	返回Array类型的index的值，对Object类型也有效。
CJSON_PUBLIC(cJSON *) cJSON_GetObjectItem(const cJSON * const object, const char * const string);	使用string获得对应的value。
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);	判断string在Object中是否存在。



cJSON API	说明
CJSON_PUBLIC(const char *) cJSON_GetErrorPtr(void);	获得错误信息。
CJSON_PUBLIC(char *) cJSON_GetStringValue(cJSON *item);	判断Item是否为字符串，返回键值字符串或NULL。
CJSON_PUBLIC(cJSON_bool) cJSON_IsInvalid(const cJSON * const item);	Item类型判断。
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);	创造对应类型的Item。
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_PUBLIC(cJSON *) cJSON_CreateRaw(const char *raw);	



cJSON API	说明
CJSON_PUBLIC(cJSON *) cJSON_CreateArray(void);	批量创造包含对应类型和数量Items的Array。
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);	
CJSON_PUBLIC(cJSON *) cJSON_CreateDoubleArray(const double *numbers, int count);	
CJSON_PUBLIC(cJSON *) cJSON_CreateStringArray(const char **strings, int count);	
CJSON_PUBLIC(void) cJSON_AddItemToArray(cJSON *array, cJSON *item);	向Array或Object增加Item。
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);	从Arrays或Objects中删除Items。
CJSON_PUBLIC(cJSON *) cJSON_DetachItemFromArray(cJSON *array, int which);	
CJSON_PUBLIC(void) cJSON_DeleteItemFromArray(cJSON *array, int which);	



cJSON API	说明
cJSON_PUBLIC(cJSON *) cJSON_DetachItemFromObject(cJSON *object, const char *string);	
cJSON_PUBLIC(cJSON *) cJSON_DetachItemFromObjectCaseSensitive(cJSON *object, const char *string);	
cJSON_PUBLIC(void) cJSON_DeleteItemFromObject(cJSON *object, const char *string);	
cJSON_PUBLIC(void) cJSON_DeleteItemFromObjectCaseSensitive(cJSON *object, const char *string);	
cJSON_PUBLIC(void) cJSON_InsertItemInArray(cJSON *array, int which, cJSON *newitem);	更新Items。
cJSON_PUBLIC(cJSON_bool) cJSON_ReplaceItemViaPointer(cJSON * const parent, cJSON * const item, cJSON * replacement);	
cJSON_PUBLIC(void) cJSON_ReplaceItemInArray(cJSON *array, int which, cJSON *newitem);	
cJSON_PUBLIC(void) cJSON_ReplaceItemInObject(cJSON *object, const char *string, cJSON *newitem);	
cJSON_PUBLIC(void) cJSON_ReplaceItemInObjectCaseSensitive(cJSON *object, const char *string, cJSON *newitem);	
cJSON_PUBLIC(cJSON *) cJSON_Duplicate(const cJSON *item, cJSON_bool recurse);	复制cJSON结构体。
cJSON_PUBLIC(cJSON_bool) cJSON_Compare(const cJSON * const a, const cJSON * const b, const cJSON_bool case_sensitive);	比较两个cJSON结构体。
cJSON_PUBLIC(void) cJSON_Minify(char *json);	将格式化的字符串压缩。
cJSON_PUBLIC(cJSON*) cJSON_AddNullToObject(cJSON * const object, const char * const name);	用于同时创建对象和将Items 添加到Object。返回添加的 Items或返回NULL表示失败。
cJSON_PUBLIC(cJSON*) cJSON_AddTrueToObject(cJSON * const object, const char * const name);	



cJSON API	说明
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)	设置整形键值，同时设置双精度浮点型键值。
cJSON_SetNumberValue(object, number)	设置双精度浮点型键值，同时设置整形键值。
cJSON_ArrayForEach(element, array)	遍历数组。
CJSON_PUBLIC(double) cJSON_SetNumberHelper(cJSON *object, double number);	设置整形和双精度浮点型键值，超过极限值的输入将按照极限值设置。
CJSON_PUBLIC(void *) cJSON_malloc(size_t size);	使用cJSON_InitHooks中注册的函数实现malloc/free操作。
CJSON_PUBLIC(void) cJSON_free(void *object);	



2 开发指南

以下为组织Json字符串和解析Json字符串的官方代码样例。

2.1 创建Json字符串示例

2.2 解析Json字符串示例

2.1 创建 Json 字符串示例

```
//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 解析 Json 字符串示例

```
/* 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->valuelstring != NULL))
    {
        printf("Checking monitor \"%s\"\n", name->valuelstring);
    }

    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;  
}
```