优特云

智能家居

领域语言编译器使用手册

文件审批：

	部门/职位	签名	日期
编制			
编制			
会签			
会签			
审核			
批准			

文档修改记录：

版本	修改内容概要	修改人	批准人	生效日期
V0.1	添加整体目录	李宇哲		
V0.3	增加系统文件的产品属性定义描述 补充接口 SetInputTXTpath	李宇哲		
V0.4	接口更新	李宇哲		
V0.5	1、增加 gettokenList 接口 2、适配多个 ulc 的编译，增加 config 配置选项 3、编译接口输出修改	李宇哲		2021/12/06
V0.6	1、 移除 SetConfig 接口 2、 移除 SHCompileCloud 接口 3、 调整《家居领域语言的输入文件格式》中的系统数据，详情请参考该文档	王春华		2021/12/09
V0.7	1 、增加 C#的批注	李宇哲		2021/12/13

V0.8	1、增加接口 SHCommonCompile 2、增加接口 getKeywordsCommon	李宇哲		2021/12/14
V0.9	1、 children 字段更新变成 Children 2、 CshGetTokenList 接口的 dataJson 字段 Type 改成 DataType	李宇哲		2021/12/16
V0.10	1、增加表达式类型的描述:Expression	李宇哲		2021/12/17
V0.11	1、C 接口的 CshCompile 的返回数据详细描述和调整字段	李宇哲		2021/12/19
V0.12	1、 新增术语的拆分功能的接口字段信息 2、编译接口的入参文件 2 个改成 3 个	李宇哲		2021/12/28
V0.13	1、 C++接口调整 2、 C++/C 的接口入参返参的数据格式更新 3、 C++增加数据存储对象类 SHFileInfo,满足返回信息数据的完整	李宇哲		2021/1/6
V0.14	1、 CshCompile 的返回值的字段 U_FILE_TYPE_ULC 改成 SH_FILE_TYPE_ULC 2、 CshSetLogout 的入参增加 userData，相应回调函数 CSHLogCallback 也有 userData	李宇哲		2022/04/26

V0.15	1、C 接口的 CshGetErr 返回字符串调整为 json 字符串	李宇哲		2022/04/29
V0.16	1、术语 tokentype 增加 DelayGlossaryID	李宇哲		2022/05/18
V0.17	1、家居的产品属性定义字段更新	李宇哲		2022/07/14
V0.18	SHCLI 更新描述	李宇哲		2022/07/15
V0.19	Shcli 示例更新	李宇哲		2022/07/15

目录

1 简介

1.1 智能家居领域语言概述

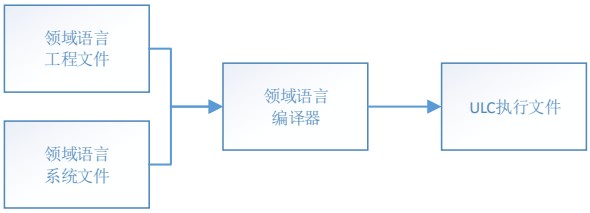
领域语言是在标准语言的基础上进行衍生扩展，形成的行业、领域里特有的方言（规则）。由用户语言基本语法或者家居规范定义的术语构成。它将用户语言的描述能力更加精准的用于家居领域，方便家居行业的使用。

智能家居领域语言，是优特云在智能家居业务领域，为优特物联公司提供的业务语言。用户可使用偏口语化的智能照明术语语言，再通过智能家居编译器编译成标准语言，标准语言再通过标准编译器编译形成虚拟机的可执行文件，最后在网关设备上执行，达到家居设备之间的本地化联动控制。

1.2 名词解释

名词	说明
标准语言	标准语言是用户语言标准语句的组成规则。该规则屏蔽了各行业、领域间的业务特性。详细内容见《用户语言 4.0 语言内核定义与设计_标准语言》。
领域语言	领域语言是在标准语言的基础上进行衍生扩展，形成的行业、领域里特有的方言（规则）。（如电子菜谱语言、智能照明语言等）
标准源程序	标准源程序是标准编译器输入数据的数据格式。详细内容见《1-用户语言 4.0 语言内核定义与设计_标准语言》。
标准编译器	标准编译器负责对标准源程序进行编译，结合标准源程序中的变量表、函数表等输入数据，将标准语言语句翻译为虚拟机字节码。
领域编译器	领域编译器负责对领域语言编写的文本进行翻译，输出标准源程序。

1.3 家居领域语言编译器

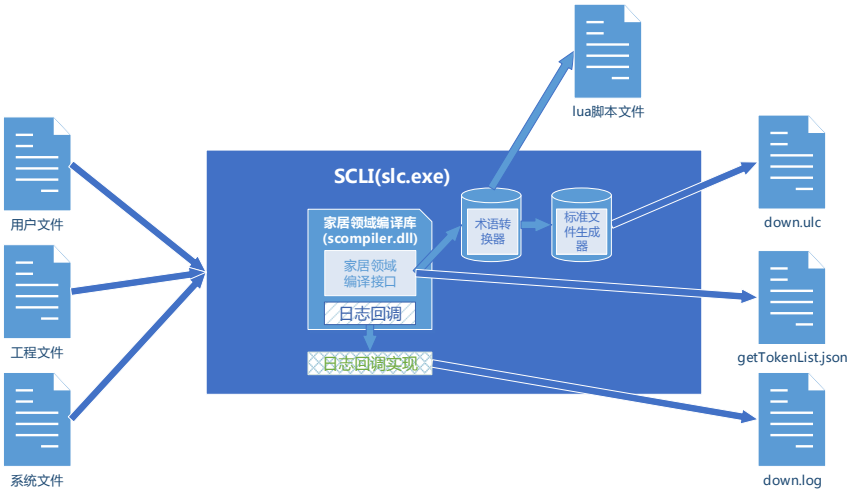


家居领域语言编译器SDK包含一个命令行工具shCLI及二次开发使用的标准编译库，如下所示：

1.3.1使用对象

主要面向领域语言编译器的开发者使用。

1.3.2主要流程



shCLI 命令行工具"shCLI.exe"接受家居领域语言源程序文件，用户文件 userData.txt，工程文件 project.json 和系统文件 system.json。在将该文件读取到字符串 txt 和 json 数据后，shCLI 将其作为输入参数，调用编译库提供的编译接口，得到输出数据 ULC 字节码。CLI 再将 ULC 字节码保存到文件中，即获得 shdown.ulc 文件。

另外，shCLI 命令行工具实现的日志回调实现会将编译过程中产生的编译日志写入 shdown.log 文件中。

2 领域语言编译库

家居领域语言编译器 SDK 是编译器的一个软件开发套件（Software Development Kit），使用者可基于 SDK 提供的编译库进行软件二次开发。目前只提供了 C++SDK，需使用 C++作为开发语言。

2.1 Hello world!

以下我们通过 1 个简单的实例来展示 SDK 领域语言编译库的使用。
下述文件格式的介绍，详见《家居领域语言的输入文件格式》
首先保存下述工程文件，将文本存储到工程文件 project.txt 中。

```
<<变量表>>
唯一标识          |变量名称          |设备名称      |产品名称          |产品内属性名称
0                 |厨房吊灯      |餐厅开关      |三开智能控制面板       |左继电器
varKey2               |厨房柜灯      |餐厅开关      |三开智能控制面板       |中继电器

<<语句表>>
条件                                                | 结果
餐厅开关.上左键按下                                  |厨房吊灯切换
a == 1                                              | b=2!
```

再保存下述数据，按 json 格式存储到系统文件 system.json 中。

1、家居术语数据定义

```
" matchGlossay": [
    {
                "level": 10,
                "category": 1,
                "type": "RelayGlossary",
                "name": "不低于",
                "logicRegularMatch": "(.+)不低于(.+)",
                "logicConversionRules":
"\"%s\",coverOperator(sz[1],sz[2],\">=\")",
                "tokenRegularMatch": "(.+)(不低于)(.+)",
                "tokenConversionRules":
"\"%s\",getTokenList(sz[1],\"Variable\",sz[2],\"RelayGlossary\",sz[3],\"Identif
er\")"
    },
]
```

2、家居的产品属性定义

```
"ProductAttributeDefine": [
        {
            "productName": "三开智能控制面板",
            "productAttributeInfo": [
                {
                     "attributeName": "左继电器",
                     "attributeType": 2,
                     "dataType": "int8",
                     "valueDescription": "{断开=0,导通=1}"
                }
            ]
        }
]
```

3、家居领域函数数据定义

```
" FunctionDefine": [
{
        "funKey": "5",
         "funName": "AdaptSceneTable",
         "returnDataType": "bool",
          "parasList": [
              {
                "dataType": "int8",
                "parasDescription": ""
              },
              {
                "dataType": "int8",
                "parasDescription": ""
              }
            ],
           "description": "自适应场景表函数"
}
```

使用 SDK 编写以下程序：

```
...
  //调用标准编译接口
  const char* outData = CshCompile(intputJsonGBK.c_str(), &ret, &outSize);

  // 编译成功，将 outData 中的数据保存,长度 outSize
  if (ret)
  {
  }
  ...
```

```
  //释放内存
  FreeCshData（outData）;
  //释放编译器对象
  FreeCshSHCompile（）
…
```

2.1.1 注意事项

需要参照《用户语言 4.0-家居公共函数与公共变量》，将时间变量和延时变量加到家居的变量表中,将公共函数添加到函数表中并实现。

2.2 函数定义

2.2.1 C++接口

2.2.1.1 家居智能照明编译器提供的接口

2.2.1.1.1 编译接口类 SHCompiler

- 函数

名称	描述
SetLogout	日志回调的设置接口。
SHCompile	家居本地编译接口。
GetErr	获取错误信息
GetTokenList	获取语句的拆分信息

如需修改日志回调,在调用 Compile 前应先调用 SetLogout 接口,使之生效。

2.2.1.1.1.1 SetLogout

- 完成功能

  设置日志回调。

- 函数定义

void SetLogout(const string& logCtx, ULogCallback logCB,void* businessData)

- 输入参数说明

参数名称	参数说明	备注
logCtx	日志上下文。标准编译器调用日志回调时会传出 logCtx，日志回调实现逻辑中可配合 logCtx 决定输出行为。	
Logout	日志回调实现 UlogCallback 的函数句柄。	定义请查阅编译日志接口
businessData	用户业务数据	

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  无

2.2.1.1.1.2 SHCompile

- 完成功能

  完成家居本地编译功能。

- 函数定义

bool SHCompile(const std::map<SH_INTFILE_TYPE, std::string>& inputMap, std::map<SH_OUTFILE_TYPE, std::vector< SHFileInfo*>>& output);

- 输入参数说明

参数名称	参数说明	备注
inputMap	一个<key,value>结构。它保存了家居工程文本和家居系统文件。	见附录《家居领域语言的输入文件格式.docx》

- 输出参数说明

参数名称	参数说明	备注
output	一个<key,value>结构	Value 可以是多个输出文件

- 返回值

成功返回 true，否则返回 false。

SH_INTFILE_TYPE 是 SDK 提供的上述输入参数的 key 的枚举值：

```
enum SH_INTFILE_TYPE {
  SH_FILE_TYPE_USER = 0    //输入用户数据
  SH_FILE_TYPE_PROJECT = 1 //输入工程数据
  SH_FILE_TYPE_SYSTEM = 2  //输入系统数据

};
```

SHFileInfo 是 SDK 提供的一个数据存储对象：参见 2.2.1.1.2

SH_OUTFILE_TYPE 是 SDK 提供的上述输出参数的 key 的枚举值：

```
enum SH_OUTFILE_TYPE{
      SH_FILE_TYPE_ULC = 0, //领域语言编译器输出的 ULC 数据文件
      SH_FILE_TYPE_TXT = 1, //领域语言编译器输出的 txt 数据文件
      SH_FILE_TYPE_MONI = 2,//领域语言编译器的模拟数据，虚拟机模拟器要用
};
```

2.2.1.1.1.3 GetErr

- 完成功能

  获取编译错误信息

- 函数定义

  SHCompileLogInfo* GetErr();

- 输入参数说明

  无

- 输出参数说明

  无

- 返回值

  错误类信息

2.2.1.1.1.4 GetTokenList

- 完成功能

  语句的拆分。

- 函数定义

  String getTokenList(const std::map<SH_INTFILE_TYPE, std::string>& inputMap, string eventConditon,string partConditon, string trueResult, string falseResult);

- 输入参数说明

参数名称	参数说明	备注
inputMap	Map 数据，输入 3 个文件字符串，编码格式必须 UTF8	包含用户文件,工程文件，系统文件
eventConditon	触发条件字符串	没有填“”
partConditon	参与条件字符串	没有填“”
trueResult	真结果	没有填“”
falseResult	假结果	没有填“”,术语语句该字段必须为空

- 返回值

  返回中序遍历的树形结构，以 json 字符串格式 token 表示。

```
{
    "eventCondition":
    {
            "SrcValue": "1",  //token 名称
            "Pos": 1,        //token 位置
            "TokenType": 1,  //token 类型
            "Children": [       //子节点
                {
                    "SrcValue": "2",
                    "Pos": 1,
                    "TokenType": 1,
                    " Children ": []
                }
            ]
    },
    "partConditon": {
    },
    "trueResult": {
    },
    "falseResult": {
```

```
        }
    }
```

2.2.1.1.1.4.1 返回标准语句的拆分字段信息

标准语句的字段信息

字段	含义	
SrcValue	所有子节点的 srcvalue 组合	
Pos	所有子节点中取最小 pos,没有子节点则取当前节点 pos	
TokenType	“Variable”,”Const”,”Operator”,”Function”,”Identifer”,”Expression”	
	变量类型	“Variable”
	常量类型	”Const”
	操作符类型	”Operator”
	函数类型	”Function”
	未知类型	”Identifer”
	表达式类型	“Expression”

举例如下：

newTokenlist.json

2.2.1.1.1.4.2 返回术语语句的拆分字段信息

字段	含义
SrcValue	所有子节点的 srcvalue 组合
Pos	所有子节点中取最小 pos,没有子节点则取当前节点 pos
TokenType	术语类型

类型	类型（字符串）
未知类型	Identifer
变量	Variable
常量	Const
延时术语	DelayGlossary
延时术语的 ID	DelayGlossaryID
取消延时术语	CancelDelayGlossary
时间点术语	TimeDotGlossary
时间段术语	TimeRangeGlossary
星期术语	WeekGlossary
节假日术语	HolidayGlossary

	灯亮术语	LightOnGlossary
	灯灭术语	LightOffGlossary
	常量值术语	ConstGlossary
	逻辑运算符术语	LogicGlossary
	关系运算符术语	RelayGlossary
	动作分隔符术语	ActionGlossary
	场景术语	ScenceGlossary
	置反术语	ReverseGlossary
	表达式	Expression
children	子节点	

举例如下：

2.2.1.1.2 数据存储对象接口类 SHFileInfo

2.2.1.1.2.1 getName

- 完成功能

  获取输出数据的编号名称。

- 函数定义

  ```
  const std::string& getName();
  ```

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  返回输出数据的编号名称

2.2.1.1.2.2 getNoVec

- 完成功能

  获取输出数据的语句编号列表，由哪些语句组成了输出数据

- 函数定义

```
std::vector<int> getNoVec();
```

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  返回输出数据的语句编号列表

2.2.1.1.2.3 getData

- 完成功能

  获取输出数据的首地址

- 函数定义

```
const char* getData();
```

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  返回输出数据的首地址

2.2.1.1.2.4 getDataSize

- 完成功能

  获取输出数据的字节长度

- 函数定义

  ```
  int getDataSize();
  ```

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  返回输出数据的字节长度

2.2.1.1.3 编译日志接口类 SHCompileLogInfo

2.2.1.1.3.1 GetLevel

- 完成功能

  获取日志级别。

- 函数定义

  ```
  SHCompileLogLevel GetLevel() const;
  ```

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  日志级别

注：

```
enum SHCompileLogLevel
{
    Debug = 0,//无关紧要的信息，例如一些调试信息，请使用此值。
    Info,     //比较重要的信息，例如“编译结束！”等。
    Warning,  //警告信息。
    Error,    //错误信息。
};
```

2.2.1.1.3.2 GetCode

- 完成功能

  获取错误码。

- 函数定义

  int GetCode() const

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  错误码

2.2.1.1.3.3 GetRowNo

- 完成功能

  获取语句编号。

- 函数定义

int GetRowNo() const

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  语句编号

2.2.1.1.3.4 GetPos

- 完成功能

  获取字符位置。

- 函数定义

  int GetPos() const

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  字符位置

2.2.1.1.3.5 GetModule

- 完成功能

  获取模块类型。

- 函数定义

SHCompileModule GetModule() const

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  模块类型

2.2.1.1.3.6 GetMessage

- 完成功能

  获取日志概要。

- 函数定义

  const std::string& GetMessage() const

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  日志概要

2.2.1.1.3.7 ToString

- 完成功能

  格式化输出日志信息

注：默认日志格式

```
${moduleName}:${rowNo}:${pos}: ${level}(${code}): $detail
数据字典初始化:2:20: error(3001): 变量名 var1 不符合命名规则
```

- 函数定义

  std::string ToString() const

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  格式化的日志信息

2.2.1.1.3.8 GetKeywords

- 完成功能

  关键字。

- 函数定义

  const std::map<std::string, std::string>& GetKeywords() const

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  关键字

2.2.1.1.3.9 GetLogicNo

- 完成功能

  获取语句编号

- 函数定义

  int GetLogicNo() const

- 输入参数说明

参数名称	参数说明	备注
无		

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  语句在语句表中的编号，从 1 开始计算

2.2.1.1.4 获取 SDK 版本号接口

2.2.1.1.4.1 GetSdkVersion()

- 完成功能

  获取 SDK 版本号。

- 函数定义

  std::string GetSdkVersion ();

- 返回值

  返回编译器 SDK 版本号。

```
{
    “shCompilerVersion”: 4.0.0.1
    “glossaryEngineVersion”: 4.0.0.1
    “compilerVersion”:4.0.0.1
```

}

2.2.1.2 接入方需要实现的接口

2.2.1.2.1 日志回调接口 ULogCallback

- 完成功能

  对编译器的日志进行处理。

- 函数定义

```
typedef void(*ULogCallback)(const std::string& logCtx,
                            SHCompileLogInfo* msg,void*
businessData)
```

- 输入参数说明

参数名称	参数说明	备注
logCtx	日志上下文。标准编译器调用日志回调时会传出 logCtx，日志回调实现逻辑中可配合 logCtx 决定输出行为。	
msg	保存了单条编译日志的容器，具体日志容器类说明。SDK 的每条编译日志都单独调用日志回调对外输出。	
businessData	用户的自定义数据	没有则填 NULL

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  无

2.2.2 C 接口

方便 C#调用

2.2.2.1 家居智能照明编译器提供的接口

2.2.2.1.1 编译接口类 SHCompiler

- 函数

名称	描述
CshSetLogout	日志回调的设置接口。
CshCompile	家居本地编译接口。
CshGetErr	获取错误信息
CshGetTokenList	获取语句的拆分信息

如需修改日志回调,在调用 Compile 前应先调用 SetLogout 接口,使之生效。

2.2.2.1.1.1 CshSetLogout

- 完成功能

  设置日志回调。

- 函数定义

  void CshSetLogout (const char* logCtx, CSHLogCallback logCB, void* businessData)

批注 [李宇哲1]: C 的 Char* ->C#的 string

批注 [李宇哲2]: C 的 Void* ->C#的 IntPtr

- 输入参数说明

参数名称	参数说明	备注
logCtx	日志上下文。编译器调用日志回调时会传出 logCtx，日志回调实现逻辑中可配合 logCtx 决定输出行为。	
logout	日志回调实现 ULogCallback 的函数句柄。	定义请查阅编译日志接口
businessData	用户自定义数据	没有填 NULL

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  无

2.2.2.1.1.2 CshCompile

- 完成功能

  完成家居本地编译功能。

- 函数定义

```
const char* CshCompile(const char* intPutJson, bool* ret, int* outSize);
```

批注 [李宇哲3]: C 的 Char* ->C#的 IntPtr

批注 [李宇哲4]: C 的 Char* ->C#的 string

- 输入参数说明

参数名称	参数说明	备注

intPutJson	Json 字符串	intPutJson 的输入数据格式如下
		{
		"SH_FILE_TYPE_USER":"userdata.txt 用户文件的字符串内容",
		"SH_FILE_TYPE_PROJECT":"project.json 工程文件的 json 内容",
		"SH_FILE_TYPE_SYSTEM":"system.json 系统文件的 json 内容",
		}

- 输出参数说明

参数名称	参数说明	备注
ret	编译正确或者失败	
outSize	返参的字符串长度	

- 返回值

  成功返回输出字符串，否则返回空。

返回值的输出数据格式如下：

字段名称	参数说明	备注
SH_FILE_TYPE_ULC	领域语言编译器输出的 ULC 数据文件,是虚拟机的执行文件	Base64 解码内容是二进制数据流
SH_FILE_TYPE_TXT	返领域语言编译器输出的 txt	Base64 解码内容是字符

	数据文件，是标准编译器的输入文件	串
SH_FILE_TYPE_MONI	领域语言编译器输出的模拟数据，是虚拟机模拟器测试用的数据	Base64 解码内容是字符串

子字段名称	参数说明	备注
ulcId	唯一编号，每一个字段的 id 都是一一对应的。	
logicNoList	语句编号的数组	表示哪些语句生成了 ULC 文件
content	Base64 的编码内容	

举例 家居编译的输出数据格式.json

2.2.2.1.1.3 CshGetErr

- 完成功能

  获取编译错误信息

- 函数定义

  ```
  const char* CshGetErr(int* outSize);
  ```

  批注 [李宇哲5]: C 的 Char* ->C#的 IntPtr

- 输入参数说明

  无

- 输出参数说明

参数名称	参数说明	备注
outSize	返参的字符串长度	

- 返回值

  错误类信息的 json 字符串

```
{
  "Level":0,
  "Code":1,
  "RowNo":1,
  "Pos":1,
```

```
        "Module":"SHEGlobal",
        "Message":"附近出现语法错误",
        "KeyWords":[
            {
                "key":"变量名",
                "Value":"value1"
            }
        ]

}
```

字段	字段类型	含义
Level	数值	日志级别 0 Debug 1 Info 2 Warning 3 Error
Code	数值	错误码
RowNo	数值	逻辑语句的行号
Pos	数值	错误字符的位置
Module	字符串	字符内容 / 含义 SHEGlobal 全局 SHELanguageData 数据字典 SHEPreCompile 预编译 SHELexer 词法分析 SHEParser 语法分析 SHESemanticAnalyzer 语义分析 SHETransferTxt 标准文件生成 SHECompiler 标准编译 SHEDefault 未知模块
Message	字符串	错误码含义或日志信息
KeyWords	数组	错误具体描述

KeyWords 字段		
字段	字段类型	含义
Key	字符串	
Value	字符串	

2.2.2.1.1.4 CshGetTokenList

- 完成功能

  语句的拆分。

- 函数定义

```
const char* CshGetTokenList(const char* intPutJson, const char* eventConditon,const char* partCondition, const char* trueResult, const char* falseResult, int* outSize);
```

批注 [李宇哲6]: C 的 Char* ->C#的 IntPtr

- 输入参数说明

参数名称	参数说明	备注

intPutJson	Json 字符串	intPutJson 的输入数据格式如下
		{
		"SH_FILE_TYPE_USER":"userdata.txt 用户文件的字符串内容",
		"SH_FILE_TYPE_PROJECT":"project.json 工程文件的 json 内容",
		"SH_FILE_TYPE_SYSTEM":"system.json 系统文件的 json 内容",
		}

eventConditon	触发条件字符串	没有填“”
partConditon	参与条件字符串	没有填“”
trueResult	真结果	没有填“”
falseResult	假结果	没有填“”，术语语句该字段必须为空

- 输出参数说明

参数名称	参数说明	备注
outSize	返参的字符串长度	

- 返回值

  返回中序遍历的树形结构，以 json 字符串格式 token 表示。

```
{
    "eventCondition":
    {
            "SrcValue": "1",  //token 名称
            "Pos": 1,         //token 位置
```

```
                "TokenType": 1,   //token 类型
                "Children": [       //子节点
                    {
                        "SrcValue": "2",
                        "Pos": 1,
                        "TokenType": 1,
                        " Children ": []
                    }
                ]
            },
            "partConditon": {
            },
            "trueResult": {
            },
            "falseResult": {
            }
        }
```

注：

1）标准语句返回值　参考　2.2.2.1.1.4.1

2）术语语句返回值　参考 2.2.2.1.1.4.2

2.2.2.1.1.4.1 返回标准语句的拆分字段信息

标准语句的字段信息

字段	含义
SrcValue	所有子节点的 srcvalue 组合
Pos	所有子节点中取最小 pos,没有子节点则取当前节点 pos
TokenType	“Variable”,”Const”,”Operator”,”Function”,”Identifer”,”Expression”
	变量类型 “Variable” 常量类型 ”Const” 操作符类型 ”Operator” 函数类型 ”Function” 未知类型 ”Identifer” 表达式类型 “Expression”

举例如下：

newTokenlist.json

2.2.2.1.1.4.2 返回术语语句的拆分字段信息

<table>
<tr><td>字段</td><td>含义</td></tr>
<tr><td>SrcValue</td><td>所有子节点的 srcvalue 组合</td></tr>
<tr><td>Pos</td><td>所有子节点中取最小 pos,没有子节点则取当前节点 pos</td></tr>
<tr><td>TokenType</td><td>术语类型
<table>
<tr><td>类型</td><td>类型（字符串）</td></tr>
<tr><td>未知类型</td><td>Identifer</td></tr>
<tr><td>变量</td><td>Variable</td></tr>
<tr><td>常量</td><td>Const</td></tr>
<tr><td>延时术语</td><td>DelayGlossary</td></tr>
<tr><td>延时术语的 ID</td><td>DelayGlossaryID</td></tr>
<tr><td>取消延时术语</td><td>CancelDelayGlossary</td></tr>
<tr><td>时间点术语</td><td>TimeDotGlossary</td></tr>
<tr><td>时间段术语</td><td>TimeRangeGlossary</td></tr>
<tr><td>星期术语</td><td>WeekGlossary</td></tr>
<tr><td>节假日术语</td><td>HolidayGlossary</td></tr>
<tr><td>灯亮术语</td><td>LightOnGlossary</td></tr>
<tr><td>灯灭术语</td><td>LightOffGlossary</td></tr>
<tr><td>常量值术语</td><td>ConstGlossary</td></tr>
<tr><td>逻辑运算符术语</td><td>LogicGlossary</td></tr>
<tr><td>关系运算符术语</td><td>RelayGlossary</td></tr>
<tr><td>动作分隔符术语</td><td>ActionGlossary</td></tr>
<tr><td>场景术语</td><td>ScenceGlossary</td></tr>
<tr><td>置反术语</td><td>ReverseGlossary</td></tr>
<tr><td>表达式</td><td>Expression</td></tr>
</table></td></tr>
<tr><td>children</td><td>子节点</td></tr>
</table>

getTokenList-Glossary.json

举例如下：

2.2.2.1.2 获取 SDK 版本号接口

2.2.2.1.3 CshGetVersion

- 完成功能

获取 SDK 版本号。

- 函数定义

```
const char* CshGetVersion(int* outSize);
```

批注 [李宇哲7]: C 的 Char* ->C#的 IntPtr

- 输出参数说明

参数名称	参数说明	备注
outSize	返参的字符串长度	

- 返回值

返回编译器 SDK 版本号。

```
{
    “shCompilerVersion”: 4.0.0.1
    “glossaryEngineVersion”: 4.0.0.1
    “compilerVersion”:4.0.0.1
}
```

2.2.2.1.4 内存释放的接口

2.2.2.1.4.1 FreeCshData

- 完成功能

用于释放所有返回 const char*的数据内存。

- 函数定义

```
void FreeCshData(void* buffer);
```

批注 [李宇哲8]: C 的 void* ->C#的 IntPtr

- 输入参数说明

参数名称	参数说明	备注
buffer	需要释放内存的数据	

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

无

2.2.2.1.4.2 FreeCshSHCompile

- 完成功能

  编译退出时需要调用，释放家居编译器对象的内存

- 函数定义

  void FreeCshSHCompile（）；

- 返回值

  无

2.2.2.2 接入方需要实现的接口

2.2.2.2.1 日志回调接口 CSHLogCallback

- 完成功能

  对编译器的日志进行处理。

- 函数定义

  typedef void(*CSHLogCallBack)(const char* sourceId,const char* msg,void* businessData);

- 输入参数说明

参数名称	参数说明	备注
logCtx	日志上下文。编译器调用日志回调时会传出 logCtx，日志回调实现逻辑中可配合 logCtx 决定输出行为。	
msg	保存了单条编译日志的信息， SDK 的每条编译日志都单独调用日志回调对外输出。	
businessData	用户自定义数据	

- 输出参数说明

参数名称	参数说明	备注
无		

- 返回值

  无

3 命令行工具 shCLI

shCLI 是领域语言编译器提供的一个命令行工具（SmartHome Command Line Interface），是一个基于领域语言编译库开发的可执行程序。使用者可通过 shCLI 进行领域语言源程序的批量编译，或是利用 shCLI 生成的编译日志进行回归测试。

3.1 Hello world!

以下我们通过 1 个简单的实例在 linux 系统展示 shCLI 的使用。

将输入数据保存到文件 system.json 和 userdata.txt,project.json 中，使用 shCLI 编译上述文件，输出结果如下：

```
$ scli -i userdata.txt system.json project.json
开始编译
编译完成
编译总共用时:95ms
```

使用 linux 的 ls 指令可看到当前文件夹下新增的 3 个文件：outFile.json，shlog.txt outFileUlc.ulc：

```
$ ls
scli outFile.json shlog.txt outFileUlc.ulc …
```

使用 linux 的 cat 指令可查看 shlog.txt 的日志内容：

```
$ cat shlog.txt
语句编号:-1;模块名:SHEGlobal;行号:-1;位置:-1;日志等级:Info;错误码:0;开始标准编译!
语句编号:-1;模块名:SHEGlobal;行号:-1;位置:-1;日志等级:Info;错误码:0;标准输入文件 id2 标准编译!…
```

3.2 使用方式

家居领域编译器 shCLI 是一个命令行工具，可将软件位置注册到环境变量中，即可在任意路径下使用。命令格式如下：

```
$ shCLI [options] [file]
```

其中 options 是 shCLI 支持的运行选项，file 需填入标准源程序的文件名(路径)。

另外地，在与 shCLI 相同路径下还有一个编译配置文件 shlc.properties。使用者可通过修改配置值调整编译配置。编译配置详细内容见章节 2.2.1.2“SetConfig”。

3.2.1运行选项

分类	选项	缩写	选项说明
程序信息	--help	-h	显示工具的使用说明。
	--version	-v	显示工具的版本号，及所使用的标准编译器 SDK 的版本号。
输入	--input	-i	指定 3 个输入文件,用户文件/系统文件/工程文件
	--tokenlist	-t	指定 4 个输入文件,用户文件/系统文件/工程文件/语句拆分数据文件
输出	--output	-o	指定文件路径。默认当前路径下的 outFile.json/outFileUlc/outFileTxt/outFileMoni 等文件
	--log	-l	指定编译日志文件。默认当前路径下的 shlog.txt

3.2.2使用示例

```
$ shCLI -i userdata.txt system.json project.json
$ shCLI -t userdata.txt system.json project.json token.json
$ shCLI - v
SHCLI version:4.01.01.09
{"shCompilerVersion":"4.01.01.09","glossaryEngineVersion":"1.0.1.2","compilerVer
sion":"4.00.04.26"}
Copyright @ 2021 All Rights Reserved 广东优特云科技有限公司版权所有
```

4 运行环境

家居领域语言编译器 SDK 提供 linux、windows 多个平台的 SDK。

5 附录

家居领域语言的输入文件格式.docx