FAW HS7

IVI System Design

PLC-API User Guide

Released on the: 2016.10.08

Version Number: 0.500

Neusoft Automotives ChangChun

**Version History**

| No. | Version | Section | Brief Description | Date | Rseponsible |
| --- | --- | --- | --- | --- | --- |
| 1 | 0.500 | All | Initial Create | 2016-10-08 | WangQi |
| 2 | 0.5001 | 3 | 1. 添加新的生命周期“OnPreHide” 2. 添加新的API（见3.8 3.9两节） | 2016-10-13 | ZhangShuang- |
| 3 | 0.5002 | 3 | 添加新的生命周期“OnShutdown” | 2016-10-31 | ZhangShuang |
| 4 | 0.5003 | 3 | 添加新的接口Power state change | 2016-11-22 | ZhangShuang |
| 5 | 0.5004 | 3 | 添加Intent机制相关说明 | 2016-11-25 | ZhangShuang |
| 6 | 0.5005 | 3 | 添加匿名AMPID的说明 | 2016-11-29 | ZhangShuang |
| 7 | 0.5006 | 3 | 添加HideMyself函数的说明 | 2016-12-12 | ZhangShuang |
| 8 | 0.5007 | 3.14 | 添加Quiet Intent的说明 | 2016-12-14 | ZhangShuang |
| 9 | 0.5008 | 3.14 | Quiet Intent现在支持后台启动App了。所以修改了关于不支持后台启动App的说明。 | 2017-3-9 | ZhangShuang |
| 10 | 0.5009 | 3.6-3.9 | 新增了返回所有电源状态的接口。其它3个电源状态相关接口废除。 | 2017-05-15 | ZhangShuang |
| 11 | 0.5010 | 3.6 | 更新电源状态回调参数列表 | 2017-05-24 | ZhangShuang |
| 12 | 0.5011 | 3.18 | 新增API | 2017-06-05 | ZhangShuang |
| 13 | 0.5012 | 3.7 | 恢复被删除的3.7API。因为它被错误的删除了。 | 2017-08-17 | ZhangShuang |
| 14 | 0.5013 | 3.5 | 增加了对各个生命周期通知的注解 | 2017-08-29 | ZhangShuang |
| 15 | 0.5014 | 3.14 | 增加了设置Under层surface对齐方式的API | 2017-09-04 | ZhangShuang |
| 16 | 0.5015 | 3.21 | 增加让当前App退出的接口 | 2018-01-08 | ZhangShuang |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Content

[1 概述 4](#_Toc503186600)

[2 相关文件 4](#_Toc503186601)

[3 接口说明 4](#_Toc503186602)

[3.1 PLC-API使用约束 4](#_Toc503186603)

[3.2 获取surfaceID接口 4](#_Toc503186604)

[3.3 获取AMPID接口 4](#_Toc503186605)

[3.4 AppPLC初始化 5](#_Toc503186606)

[3.5 PLC生命周期 5](#_Toc503186607)

[3.6 设置监听获取最新电源状态的回调函数的接口 6](#_Toc503186608)

[3.7 设置监听电源状态变化消息的回调函数的接口 6](#_Toc503186609)

[3.8 设置监听Shutdown消息的回调函数的接口 7](#_Toc503186610)

[3.9 ~~设置监听PowerStateChange消息的回调函数的接口~~ 7](#_Toc503186611)

[3.10 ~~设置监听ACCStateChange消息的回调函数的接口~~ 7](#_Toc503186612)

[3.11 ~~获取视频播放surfaceID接口~~ 8](#_Toc503186613)

[3.12 ~~获取视频播放背景surfaceID接口~~ 8](#_Toc503186614)

[3.13 获取under层surfaceID接口 8](#_Toc503186615)

[3.14 获取背景层surfaceID接口 8](#_Toc503186616)

[3.15 设置APP的Under层surface的对齐方式 8](#_Toc503186617)

[3.16 启动Home应用接口 9](#_Toc503186618)

[3.17 Intent接口 9](#_Toc503186619)

[3.18 匿名AMPID 13](#_Toc503186620)

[3.19 自我隐藏接口 13](#_Toc503186621)

[3.20 自我显示接口 13](#_Toc503186622)

[3.21 自我终结接口 14](#_Toc503186623)

[3.22 App激活态请求接口 14](#_Toc503186624)

[4 Refrences 14](#_Toc503186625)

# 概述

本文档主要描述PLC-API。PLC主要负责应用程序的生命周期，所有应用程序必须使用此接口。

推荐UI（Kanzi）程序使用此API。

# 相关文件

头文件：/usr/include/appSDK/AppPLC.h

库文件：/usr/lib/libappSDK.so

# 接口说明

## PLC-API使用约束

约束条件：

* 系统中AppManager进程正常运行；
* 此应用程序通过AppManager启动；
* 应用程序使用传递的surfaceID**成功创建surface**；
* 使用PLC-API前需要首先初始化AppSDK；
* 当前版本支持[匿名AMPID](#_匿名AMPID)，详情参见相关章节；

## 获取surfaceID接口

|  |  |  |
| --- | --- | --- |
| *declare* | unsigned int GetSurfaceID(void) | |
| *description* | 获取surfaceID | |
| *parameter* | type void | null |
| *return* | Surface ID的值 | |

UI（Kanzi）代码需要使用此surfaceID创建surface。

通常，这个surface ID又常常被称为“UI层surface ID”或“HMI层surface ID”。它们都是指由这个函数获得的surface ID。

之所以这样称呼是因为一个APP除了拥有UI（HMI）层之外，还可以拥有Under层（又称“视频层”，因为video应用是最早使用这个层的APP，它用这个层来播放视频）和背景层两个surface。

## 获取AMPID接口

AMPID是AppManager Process ID的简称，由AppManager定义的应用进程唯一标识，AppManager用此ID来识别和管理应用程序的生命周期。

|  |  |  |
| --- | --- | --- |
| *declare* | unsigned int GetAMPID(void) | |
| *description* | 获取AMPID | |
| *parameter* | type void | null |
| *return* | AMPID的值 | |

UI（Kanzi）代码需要使用此AMPID初始化AppPLC。

## AppPLC初始化

此方法为AppPLC基类的方法。

|  |  |  |
| --- | --- | --- |
| *declare* | bool Initialize(unsigned int AMPID) | |
| *description* | AppPLC初始化 | |
| *parameter* | type unsigned int | AMPID的值 |
| *return* | true：初始化成功 false：初始化失败 | |

## PLC生命周期

应用程序需要创建一个应用类继承AppPLC基类。此应用类调用初始化接口后，用于接收APP生命周期相关的通知。应用类需要实现以下虚函数：

void OnCreate(void);

void OnPreShow(long nArgs);//nArgs是历史遗留的产物，可以忽略

void OnShow(void);

void OnPreHide(void);

void OnHide(void);

void OnStop(void);

以上函数中，OnCreate在HMI层surface可用后被调用一次，通知APP，surface已经创建完毕，可以使用了。

OnPreShow在APP即将被显示时调用一次，通知APP，做好显示前的准备工作。

OnShow在APP可见后被调用一次，通知APP，当前APP已经可见了。

OnPreHide在APP即将被隐藏时调用一次，通知APP，做好被隐藏前的准备工作。

OnHide在APP刚刚被隐藏后调用一次，通知APP，隐藏操作已经完成。

OnStop在APP即将被关闭时调用一次，通知APP，当前APP即将被关闭。

特别地，对于OnPreShow和OnPreHide两个函数，它的执行时间最长不应该超过300ms。否则app-manager将不再等待APP进程，将APP隐藏起来或者显示出来。

OnPreShow和OnPreHide往往负责执行APP在前后台转换过程中需要执行的操作。对于复杂的APP或代码逻辑混乱的APP，这些操作可能是很沉重的。这时就要注意这两个函数的执行时间。

将这些任务放到其它线程执行并不能解决问题。因为错过了这两个时间段，APP就无法在用户不可见的情况下向UI输出内容了。**这个问题十分值得注意**：你可能会觉得这不是什么问题，但是对于wayland协议而言，你完全不能在一个surface不可见的情况下更新这个surface的任何内容。或许你认为你可以只向buffer输出而不向真正的surface输出，但是这完全取决于你使用的UI框架的实现。也就是说很可能离开了这个时间段，你调用的每一个会更新UI的函数都将阻塞或生成很多被积攒起来的帧。然后这些操作/帧会在APP可见时再一股脑释放出来，就像一系列快进一样，我们将这个现象称之为**“画面积攒”**。这个问题在KANZI引擎上很突出。你必须想办法在我们提供的时间段内完成界面的后台更新，否则就要接受你的APP出现“画面积攒”。

## 设置监听获取最新电源状态的回调函数的接口

调用此接口可以注册一个监听电源状态变化消息的函数。

|  |  |  |
| --- | --- | --- |
| *declare* | void ListenToPowerStateChangeExNotify(PowerStateChangeExNotifyCB f); | |
| *description* | 注册一个监听电源状态变化消息的函数 | |
| *parameter* | PowerStateChangeExNotifyCB f | typedef void (\*PowerStateChangeExNotifyCB)(ama\_PowerState\_e, unsigned int); |
| *return* | Null | |

此接口是获取最近的电源状态

enum ama\_PowerState\_e{

E\_PS\_ACCOFF = 0x00000000,

E\_PS\_ACCON\_ABNORMAL\_SOCVOLTAGE = 0x10000000 ,

E\_PS\_ACCON\_ABNORMAL\_SOCTEMP = 0x10000001,

E\_PS\_ACCON\_ABNORMAL\_SCRTEMP\_HIGH = 0x10000002,

E\_PS\_ACCON\_ABNORMAL\_SCRTEMP\_HIGHER = 0x10000003,

E\_PS\_ACCON\_NORMAL\_POWERON\_SCRON = 0x11110000,

E\_PS\_ACCON\_NORMAL\_POWERON\_SCROFF = 0x11100000,

E\_PS\_ACCON\_NORMAL\_POWERON\_SCROFF\_AWAKE = 0x11101000,

E\_PS\_ACCON\_NORMAL\_POWEROFF = 0x11000000,

E\_PS\_ACCON\_NORMAL\_POWEROFF\_AWAKE = 0x11001000,

E\_PS\_POWER\_INIT = 0x0000ffff,

E\_PSTA\_DEF\_MAX

};

## 设置监听电源状态变化消息的回调函数的接口

调用此接口可以注册一个监听电源状态变化消息的函数。

|  |  |  |
| --- | --- | --- |
| *declare* | void ListenToChangeOfPowerStateNotify(changeOfPowerStateCB f) | |
| *description* | 注册一个监听电源状态变化消息的函数 | |
| *parameter* | changeOfPowerStateCB f | typedef void (\*changeOfPowerStateCB)(ama\_changeOfPowerSta\_e, unsigned int); |
| *return* | Null | |

此接口是获取电源状态的变化

enum ama\_changeOfPowerSta\_e

{

E\_PS\_NO\_CHANGE,

E\_ACC\_CHANGE\_TO\_ON,

E\_ACC\_CHANGE\_TO\_OFF,

E\_STATE\_CHANGE\_TO\_NORMAL,

E\_STATE\_CHANGE\_TO\_ABNORMAL,

E\_POWER\_CHANGE\_TO\_ON,

E\_POWER\_CHANGE\_TO\_OFF,

E\_SCR\_CHANGE\_TO\_ON,

E\_SCR\_CHANGE\_TO\_OFF,

};

## 设置监听Shutdown消息的回调函数的接口

IG硬线从高电位变为低电位时，APPManager会给所有接受APPManager管理的进程发送Shutdown消息。

调用此接口可以注册一个监听Shutdown消息的函数。

|  |  |  |
| --- | --- | --- |
| *declare* | void ListenToShutdownNotify(ShutdownNotifyCB f) | |
| *description* | 注册一个监听Shutdown消息的函数 | |
| *parameter* | ShutdownNotifyCB f | typedef void (\*ShutdownNotifyCB)(unsigned int) |
| *return* | Null | |

## ~~设置监听PowerStateChange消息的回调函数的接口~~

该接口已过时，请勿使用。相关消息请从[3.6 设置监听电源状态变化消息的回调函数的接口](#_设置监听电源状态变化消息的回调函数的接口)获取。

在系统power on或power off时，注册的回调函数会被调用。

|  |  |  |
| --- | --- | --- |
| *declare* | void ListenToPowerStateChangeNotify(PowerStateChangeNotifyCB f) | |
| *description* | 注册一个监听PowerStateChange消息的函数 | |
| *parameter* | PowerStateChangeNotifyCB f | typedef void (\*PowerStateChangeNotifyCB)(bool) |
| *return* | Null | |

## ~~设置监听ACCStateChange消息的回调函数的接口~~

该接口已过时，请勿使用。相关消息请从[3.6 设置监听电源状态变化消息的回调函数的接口](#_设置监听电源状态变化消息的回调函数的接口)获取。

在系统ACC on或ACC off时，注册的回调函数会被调用。

|  |  |  |
| --- | --- | --- |
| *declare* | void ListenToACCStateChangeNotify(ACCStateChangeNotifyCB f) | |
| *description* | 注册一个监听ACCStateChange消息的函数 | |
| *parameter* | ACCStateChangeNotifyCB f | typedef void (\*ACCStateChangeNotifyCB)(bool) |
| *return* | Null | |

## ~~获取视频播放surfaceID接口~~

该接口已经过时，会在不可预见的时候被删除。推荐使用[GetUnderLayerSurfaceID](#_获取under层surfaceID接口)。

此接口专门为Video应用设计，用于获取视频播放surfaceID。

|  |  |  |
| --- | --- | --- |
| *declare* | unsigned int GetVideoPlaySurfaceID(void) | |
| *description* | 获取视频播放surfaceID | |
| *parameter* | type void | null |
| *return* | surfaceID的值 | |

## ~~获取视频播放背景surfaceID接口~~

该接口已经过时，会在不可预见的时候被删除。推荐使用[GetBKGLayerSurfaceID](#_获取背景层surfaceID接口)。

此接口专门为Video应用设计，用于获取视频播放背景surfaceID。

|  |  |  |
| --- | --- | --- |
| *declare* | unsigned int GetVideoBKSurfaceID(void) | |
| *description* | 获取视频播放背景surfaceID | |
| *parameter* | type void | null |
| *return* | 背景surfaceID的值 | |

## 获取under层surfaceID接口

此接口为3层应用设计，用于获取HMI层的下一层。HMI层具有透明背景时，该层可见。

|  |  |  |
| --- | --- | --- |
| *declare* | unsigned int GetUnderLayerSurfaceID (void) | |
| *description* | 获取under层surfaceID | |
| *parameter* | type void | null |
| *return* | surfaceID的值 | |

## 获取背景层surfaceID接口

此接口为3层应用设计，用于获取under层的下一层。~~由于当前的设计中还没有所有APP共享的背景图，HMI层和under层不更新的区域会留下以前绘制的图案的残影。如果需要去除残影，可以在背景层上画上背景。~~该层可以用于描画APP的背景，也可以完全不使用。不使用的情况下APP拥有一个黑色的背景。

|  |  |  |
| --- | --- | --- |
| *declare* | unsigned int GetBKGLayerSurfaceID (void) | |
| *description* | 获取背景层surfaceID | |
| *parameter* | type void | null |
| *return* | 背景surfaceID的值 | |

## 设置APP的Under层surface的对齐方式

HS7系统建议APP总是创建分辨率跟屏幕分辨率相同的surface。但是有些时候受工具限制，surface的分辨率可能会小于屏幕的分辨率（大于屏幕的话，目前没有办法处理）。

所以HS7系统提供了以下函数用于设置Under层surface的对齐方式。**目前只支持居中和靠左上角两种对齐方式。**不设置的情况下，默认会将Under层surface居中显示。

|  |  |  |
| --- | --- | --- |
| *declare* | void SetUnderLayerCenter(bool isCenter = true) | |
| *description* | 设置Under层surface的对齐方式 | |
| *parameter* | bool | 是否居中显示Under层surface |
| *return* | null | |

获取APP的Under层surface的对齐方式：

|  |  |  |
| --- | --- | --- |
| *declare* | bool IsUnderLayerCenter() | |
| *description* | 获取Under层surface的对齐方式 | |
| *parameter* | type void | null |
| *return* | 是否居中显示Under层surface | |

## 启动Home应用接口

|  |  |  |
| --- | --- | --- |
| *declare* | void StartHome(void) | |
| *description* | 启动Home应用 | |
| *parameter* | type void | null |
| *return* | null | |

## Intent接口

Intent机制用于帮助当前APP控制其它已经安装的APP完成一些功能。比如语音助手可以根据用户的输入打开电话簿并添加一个联系人，并让电话簿留在编辑界面等待用户的确认或进一步编辑。

Intent命令分3个部分：Item ID、action、data。

Item ID是被调用的APP的Item ID；

action是一个字符串，代表被调用APP提供的某个操作接口；

data也是一个字符串，代表传递给操作接口的数据或者数据的地址。

负责执行Intent命令的APP会被启动并置顶。

下面介绍Intent接口的具体使用方法。

|  |  |  |
| --- | --- | --- |
| *declare* | void Intent(const std::string &itemID, const std::string &action, const std::string &data) | |
| *description* | 发送Intent命令 | |
| *parameter* | const std::string & | 被调用的APP的Item ID |
| *parameter* | const std::string & | 被调用APP提供的操作接口的名字 |
| *parameter* | const std::string & | 传递给操作接口的数据或者数据的地址 |
| *return* | null | |

action为空时，Intent命令与StartApp函数功能相同。

|  |  |  |
| --- | --- | --- |
| *declare* | std::string GetItemID() | |
| *description* | 获取当前应用的Item ID | |
| *parameter* | type void | null |
| *return* | std::string | |

|  |  |  |
| --- | --- | --- |
| *declare* | void ListenToIntentNotify(IntentNotifyCB f) | |
| *description* | 设置监听发给当前应用的Intent命令的回调函数 | |
| *parameter* | IntentNotifyCB | 用于监听Intent命令的回调函数。 |
| *return* | null | |

|  |  |  |
| --- | --- | --- |
| *declare* | typedef void (\*IntentNotifyCB)(const std::string&, const std::string&) | |
| *description* | 监听Intent命令的回调函数的原型。 | |
| *parameter* | const std::string& | Intent命令的action |
| *parameter* | const std::string& | Intent命令的data |
| *return* | null | |

以上就是Intent机制的基本组件。Intent函数用于发送Intent命令，接收方通过ListenToIntentNotify函数注册回调函数进而接收发来的Intent命令。命令的具体格式由收发双方共通协定。

下面是便于Intent函数使用而提供的工具：

|  |  |  |
| --- | --- | --- |
| *declare* | std::string MakeDelimitedString(const std::string delimiter, const std::string str1, ...) | |
| *description* | 将数个字符串以指定的分隔符连接成单个字符串。 | |
| *parameter* | const std::string delimiter | 分隔符 |
| *parameter* | const std::string str1, ... | 被连接的字符串 |
| *return* | null | |

它主要用于将多个参数按顺序组合成一个字符串，作为data传递给接收方。如果嫌名字长，还可以使用MKDS宏。

MakeDelimitedString函数还有char\*版本和vector版本。

|  |  |  |
| --- | --- | --- |
| *declare* | std::vector<std::string> Split(const std::string str, const std::string delimiter) | |
| *description* | 按照指定的分隔符将单个字符串分拆成多个字符串 | |
| *parameter* | const std::string str | 被拆分的字符串 |
| *parameter* | const std::string delimiter | 分隔符 |
| *return* | std::vector<std::string> 拆分好的字符串 | |

Split函数用于在获取data后，将字符串按顺序分解为多个参数。

**ItemID类**：

ItemID类能极大的方便Intent命令的使用者。

它使用起来大概像这样：

logInfo(BNSS, "Intent test 1 ");

ItemID("?.?.radio.?").Intent();

logInfo(BNSS, "Intent test 2 ");

ItemID itemID("?.?.radio.?");

itemID.ShiftAppID("media-image").ShiftScreenID("scr3").ShiftScreenID(3).Intent();

logInfo(BNSS, "Intent test 3 ");

ItemID("?.?.radio.?").Intent("open", "FM 90.8MHz")

它主要提供了模糊item ID和域切换功能。

**模糊item ID：**

一个常规的item ID是这样的：

com.hs7.radio.scr1

它包含4个域：域名称、机构名、app名、屏幕ID

初始化ItemID类时，如果对于与当前APP相同的域，可以直接用“?”表示。比如当com.hs7.home.scr1想要启动com.hs7.radio.scr1时，只要调用ItemID("?.?.radio.?").Intent()就可以了。因为两者的域名称、机构名称、屏幕ID都相同。

**域切换：**

ShiftXXXX系列函数用于切换XXXX所带表的域的内容，比如itemID.ShiftAppID("media-image")则将当前itemID的App名变成"media-image"。ShiftXXXX函数会返回一个全新的ItemID对象。所以切换后应该直接调用Intent函数发出Intent命令或者继续调用ShiftXXXX函数。

ItemID默认被初始化为“?.?.?.?”，即代表当前APP的ItemID。如果1号屏的radio应用想要命令3号屏的radio应用播放FM 90.8MHz，只需要这样写：ItemID().ShiftScreenID(3).Intent("open", "FM 90.8MHz")

下面是具体的函数声明：

|  |  |  |
| --- | --- | --- |
| *declare* | ItemID ShiftDomain(const std::string domain)const | |
| *description* | ItemID类的成员函数，用于切换域名称为指定的值 | |
| *parameter* | const std::string domain | 新的域名称 |
| *return* | ItemID | |

|  |  |  |
| --- | --- | --- |
| *declare* | ItemID ShiftOrganization(const std::string organization)const | |
| *description* | ItemID类的成员函数，用于切换组织名称为指定的值 | |
| *parameter* | const std::string organization | 新的组织名称 |
| *return* | ItemID | |

|  |  |  |
| --- | --- | --- |
| *declare* | ItemID ShiftAppID(const std::string appID)const | |
| *description* | ItemID类的成员函数，用于切换App名称为指定的值 | |
| *parameter* | const std::string appID | 新的App名称 |
| *return* | ItemID | |

|  |  |  |
| --- | --- | --- |
| *declare* | ItemID ShiftScreenID(const std::string screenID)const | |
| *description* | ItemID类的成员函数，用于切换屏幕ID为指定的值 | |
| *parameter* | const std::string screenID | 新的屏幕ID |
| *return* | ItemID | |

目前有效的屏幕ID有：scr0, scr1, scr2, scr3, service。其中scr0是HMIC测试用的屏幕编号，通常情况下不可使用。service代表一个不占用屏幕的服务进程，目前尚不支持向服务发送intent命令。

|  |  |  |
| --- | --- | --- |
| *declare* | ItemID ShiftScreenID(unsigned int id)const | |
| *description* | ItemID类的成员函数，用于切换屏幕ID为指定的值 | |
| *parameter* | unsigned int id | 新的屏幕ID |
| *return* | ItemID | |

scr0, scr1, scr2, scr3, service对应的数字ID分别为：0，1，2，3，0xF。使用其它值得到的Item ID是未定义的。请确保不要使用允许值之外的数字设置屏幕ID。

|  |  |  |
| --- | --- | --- |
| *declare* | std::string GetString()const | |
| *description* | 获取当前ItemID对象所表示的Item ID的字符串 | |
| *parameter* | type void |
| *return* | std::string | |

|  |  |  |
| --- | --- | --- |
| *declare* | void SetString(const std::string& itemID) | |
| *description* | 重新设置当前ItemID对象所代表的Item ID。 | |
| *parameter* | const std::string& |
| *return* | null | |

**Quiet Intent：**

根据实际使用中的具体需求，Intent机制的最新版中加入了Quiet Intent特性。

Quiet Intent是指在原有Intent机制基础上，不显示目标app的一种Intent方式。

常规Intent动作还会触发App的启动//显示。如果采用Quiet Intent方式，则不会将后台APP显示出来。而且使用Quiet Intent方式调用未启动的APP时，APP会启动并执行Intent语句，但是不会显示出来。

使用方法就是在原有的参数列表基础上，再加一个参数“true”。表明希望这次Intent是Quiet模式，尽量不显示APP。

相关API如下：

|  |  |  |
| --- | --- | --- |
| *declare* | void Intent(unsigned int app, const std::string &action, const std::string &data, bool isQuiet) | |
| *description* | 发送Intent命令 | |
| *parameter* | const std::string & | 被调用的APP的AMPID |
| *parameter* | const std::string & | 被调用APP提供的操作接口的名字 |
| *parameter* | const std::string & | 传递给操作接口的数据或者数据的地址 |
| *parameter* | bool isQuiet | 指定Intent是否为Quiet方式。True为Quiet方式，False为非Quiet方式 |
| *return* | null | |

|  |  |  |
| --- | --- | --- |
| *declare* | void Intent(const std::string &itemID, const std::string &action, const std::string &data, bool isQuiet) | |
| *description* | 发送Intent命令 | |
| *parameter* | const std::string & | 被调用的APP的Item ID |
| *parameter* | const std::string & | 被调用APP提供的操作接口的名字 |
| *parameter* | const std::string & | 传递给操作接口的数据或者数据的地址 |
| *parameter* | bool isQuiet | 指定Intent是否为Quiet方式。True为Quiet方式，False为非Quiet方式 |
| *return* | null | |

此外，ItemID类同样支持Quiet Intent。使用的方法也类似，都是在原本的Intent成员函数的基础上增加一个参数，“true”。

## 匿名AMPID

通常情况下，要使用AppFramework提供的服务必须是接受AppFramework管理的进程，这样的进程拥有AppFramework分配的AMPID。

但是部分功能可以提供给不接受AppFramework管理的进程的。我们为这样不受管理的进程提供了随机分配的AMPID，这样的进程只能使用AppFramework的部分功能。具体的可用功能列表还没有明确列出。以后会逐步更新。

目前已知可用的API族:

1. 简单的Intent
2. Popup

## 自我隐藏接口

|  |  |  |
| --- | --- | --- |
| *declare* | void HideMyself() | |
| *description* | 隐藏调用该函数的APP | |
| *parameter* | null |
| *return* | null | |

## 自我显示接口

|  |  |  |
| --- | --- | --- |
| *declare* | void ShowMyself() | |
| *description* | 将当前App显示出来 | |
| *parameter* | null | N/A |
| *return* | null | |

## 自我终结接口

|  |  |  |
| --- | --- | --- |
| *declare* | void StopMyself() | |
| *description* | 终止当前App的运行，让App正常退出 | |
| *parameter* | null | N/A |
| *return* | null | |

## App激活态请求接口

|  |  |  |
| --- | --- | --- |
| *declare* | void AppPowerAwakeRequest(bool isAwake); | |
| *description* | 请求进入或者退出激活态 | |
| *parameter* | bool isAwake | true:请求进入激活态 false：请求退出激活态 |
| *return* | null | |

# Refrences