WNS 概要设计说明书

Version: 1.0.0

DocCode: SW-SCXXXX-XX-000X

Date: 2015-06-11

www.spreadtrum.com

重要声明

版权声明

本文档中的任何内容受《中华人民共和国著作权法》的保护，**版权所有 © 2007，展讯通信有限公司，保留所有权利**，但注明引用其他方的内容除外。

商标声明

展讯通信有限公司和展讯通信有限公司的产品是展讯通信有限公司专有。在提及其他公司及其产品时将使用各自公司所拥有的商标，这种使用的目的仅限于引用。

不作保证声明

展讯通信有限公司不对此文档中的任何内容作任何明示或暗示的陈述或保证，而且不对特定目的的适销性及适用性或者任何间接、特殊或连带的损失承担任何责任。

保密声明

本文档（包括任何附件）包含的信息是保密信息。接收人了解其获得的本文档是保密的，除用于规定的目的外不得用于任何目的，也不得将本文档泄露给任何第三方。

前　言

文档说明

本文档为新增控件或者修改控件的说明，为应用熟悉 WNS 提供参考文档。

阅读对象

本文为控件开发人员、APP 开发人员以及 FAE 使用的参考文档。

内容介绍

本文档包括五个章节，分别为：

- 第一章：概述。简要介绍 WNS，并说明相关概念和缩写；
- 第二章：设计参考目标。介绍了 WNS 的功能需求和对外接口列表；
- 第三章：体系结构。介绍了 WNS 的体系结构；
- 第四章：模块概要设计。描述了 WNS 模块的概要设计；
- 第五章：与外部模块的耦合度分析。分析了 WNS 模块与 MMK 模块、APP 应用模块及其他控件模块的耦合度。

文档约定

本文档采用下面醒目标志来表示在操作过程中应该特别注意的地方。

注意：

提醒操作中应注意的事项。

说明：

说明比较重要的事项。

相关文档

目 录

第1章 概述

1.1 概述

本文主要讨论 BOCA 项目微软的 WNS 需求。微软给第三方应用提供了一个统一的推送通知的服务，我们主要是配合微软把相关模块移植到展讯平台上来。

1.2 相关概念

WNS 是 The Windows Push Notification Services 的缩写.

第2章 设计参考目标

2.1 功能需求规格列表

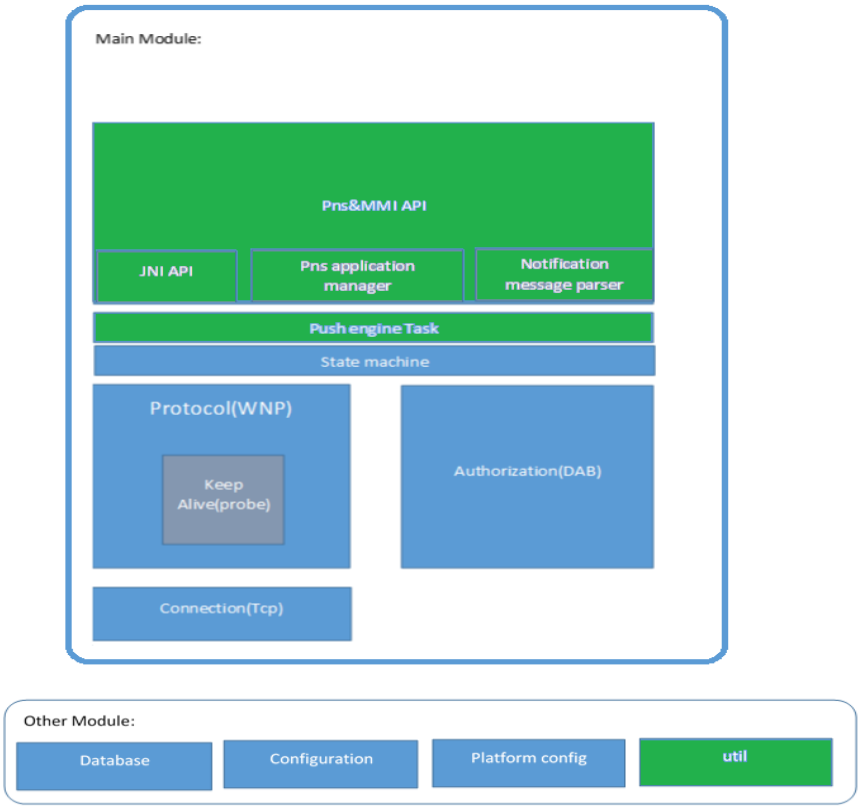
2.1.1 WNS 的需求规格描述

2.2 export 接口列表

第3章 体系结构

3.1 体系结构

3.1.1 WNS 的总体结构图



本文重点讨论绿色模块三大部分。

PNS(push notification service)&MMI API:

1. JNI api: 给第三方应用提供的接口。
2. Pns application manager: 管理 PNS 应用，决定是否使用 WNS 服务及是否开启网络连接。
3. Notification message parse：解析第三方应用服务器通过 WNS 服务给终端用户推送的通知消息。

Push Notification Task: 负责接受和分发来自其他 TASK 的消息. 包括 mmi_msg, network_msg 和 timer_msg.

Util component: 相关的底层支撑库. eg: tcp socket interface, json parser, http interface, xml parser, ssl.

其中，Util component 模块由接口人 xiaojun.zhang 联系底层同事统一提供。本文暂不讨论。

由于 Push Notification Task 模块微软也已经实现。所以本文重点讨论：**PNS(push notification service)&MMI API** 模块。

3.1.2 WNS 与其他模块的关系

3.2 设计结果示意图

第4章 模块概要设计

4.1 模块描述

PNS(push notification service)&MMI API:

1. JNI api: 给第三方应用提供的接口。
2. Pns application manager: 管理 PNS 应用，决定是否使用 WNS 服务及是否开启网络连接。
3. Notification message parser：解析第三方应用服务器通过 WNS 服务给终端用户推送的通知消息。

MMI Task 和 PNS Task 基于消息驱动交互。

4.2 功能描述

4.2.1 JNI api: 给第三方应用提供的接口

- typedef void (*get_push_notification_channel_cb)(char* appname, char * appkey ,push_notification_channel * channel);

  此回调函数是 SPRD 跟第三方应用约定，通道消息的传递函数，由第三方应用实现，作为 create_push_notification_channel_for_applicationasync 的参数传递给 SPRD。

- MSG_WNS_STATUS_E create_push_notification_channel_for_applicationasync(char* appname, char * appkey, get_push_notification_channel_cb *func_ptr)

  该接口提供给第三方应用用于创建推送通知的通道，此接口将第三方应用相关的信息：appname,appkey 及传递信息的回调函数指针发送给 Pns application manager 保存，然后 Pns application manager 通过给 WNS TASK 发消息来实现。

- BOOL close_push_notification_channel_for_applicationasync(char * appkey)

  该接口提供给第三方应用用于关闭推送通知的通道，通过向 WNS Task 发送 MSG_WNS_STOP_WNS 消息实现。

- wns_push_notification *get_notifcation_message_information (char * appkey, int index)

  第三方应用服务器通过 WNS 服务器发送过来的通知消息，Pns application manager 根据 appkey 保存到链表里，该接口通过扫描链表获取。

- int get_notification_message_count(char * appkey)

该接口给第三方应用提供当前应用的通知条数。

4.2.2 Pns application manager

数据结构：

推送通知通道信息：

```
typedef struct    push_notification_channel
{
      int channel_uri_length;
      char channel_id[CHANNEL_ID_MAX_LEN];
      char* channel_uri;
      char expiry_time[TIME_MAX_LEN];
}PUSH_NOTIFICATION_CHANNEL_T;
```

WNS Task 发给 MMI Task 的推送信息：

```
typedef struct    wns_push_notification
{
      char channel_id[CHANNEL_ID_MAX_LEN];
      char time[TIME_MAX_LEN];
      MSG_WNS_NOTIFICATION_TYPE_E notification_type;
      int content_length;
      char* contents;
}WNS_PUSH_NOTIFICATION_T;
```

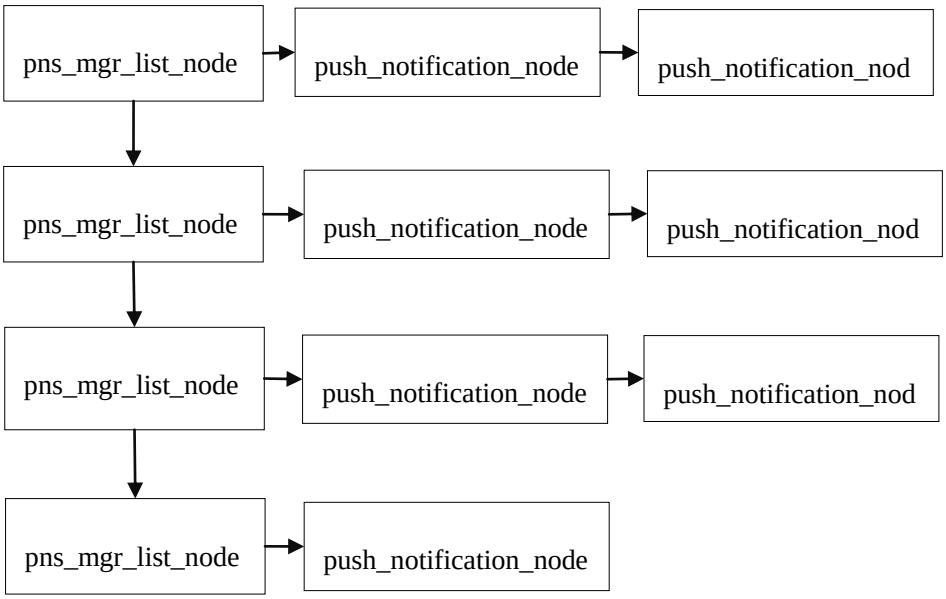
Push notification manager 保存通知信息的节点：

```
typedef struct    wns_push_notification_node_tag
{
```

```
    char time[TIME_MAX_LEN];
    MSG_WNS_NOTIFICATION_TYPE_E notification_type;
    int content_length;
    char* contents;
    get_push_notification_channel_cb *get_channel_info_func_cb;
    struct    wns_push_notification_node_tag *next;
}WNS_PUSH_NOTIFICATION_NODE_T;
```

Push notification manager 保存所有第三方应用通道信息以及通知信息的链表：

```
typedef struct    pns_mgr_list_tag
{
    char* appname;
    char* appkey;
    PUSH_NOTIFICATION_CHANNEL_T    *push_notification_channel_ptr;
    WNS_PUSH_NOTIFICATION_NODE_T *push_notification_node_ptr;
    int push_notification_count;
    struct    pns_mgr_list_tag *next_app;
}PNS_MGR_LIST_T;
```



Pns application manager 维护以上链表，处理 WNS Task 发送过来的消息。链表保存在 NV 中。

开机后 Pns application manager 先加载 NV 中保存的第三方应用相关的 channel 及 notification 信息。如果链表为空，则不需要启动 WNS 服务。

否则注册 WNS 服务，给 WNS TASK 发 MSG_WNS_REG_WNS 消息，只需要给 WNS TASK 发一次注册消息。注册消息带上 MMI Task 的 ID。

typedef struct

{

SIGNAL_VARS

BLOCK_ID task_id;//the task_id indicate which task the wns will communicate with

}WNS_REG_SIGNAL_T;

然后判断 WNS 服务是否连接 WNS 服务器，如果未连接，给 WNS TASK 发 MSG_WNS_CONNECT_WNS 消息连接服务器。消息里需要带上操作系统相关信息及网络 ID。

Pns application manager 用状态机实现与 WNS engine task 连接，

例如可以有如下状态：{ disconnect，connecting，connect ，etc… } ，初始状态为 disconnect

那么当第一次发了 MSG_WNS_CONNECT_WNS 后，进入 connecting， 等待 MSG_WNS_RESPONSE，如果 MSG_WNS_RESPONSE 不是 WNS_STATUS_SUCCESS，可以设置状态为 disconnect，以便继续重连。

当收到 MSG_WNS_CONNECTION_CLOSED，如果状态是 connect， 可以设置为 disconnect，

通过检测状态，如果是 disconect，才可以继续连。

避免连续发多次 MSG_WNS_CONNECT_WNS。

/*Indicate request message structure of MSG_WNS_CONNECT_WNS*/

typedef struct

{

SIGNAL_VARS

WNS_AGENT_INFO_T agent_info;

TCPIP_NETID_T net_id;

}WNS_CONNECT_SIGNAL_T;

如果 Pns application manager 收到第三方应用发过来的需要创建通道的消息，首先判断是否注册，如果没有就先注册，然后判断是否连接，如果没有连接，则先连接。启动定时器，等连接好后

给 WNS TASK 发 MSG_WNS_CREATE_CHANNEL 消息，第一次发送的时候，channel_id 设置为空，channel_id_length 设置为 0。如果曾经申请过通道，需要更新，则需要在消息体里带上 channel_id 和 channel_id_length。

/*Indicate request message structure of MSG_WNS_CREATE_CHANNEL*/

typedef struct

{

SIGNAL_VARS

char app_key[APP_KEY_MAX_LEN];

uint32 channel_id_length;//Note: If channel_id is null or channel_id_length is 0, the server will response a new channel id. If channel_id and channel_id_lengthhas value, that means renew the channel uri using current channel id

char* channel_id;//wns need to free this memory.

}WNS_CREATE_CHANNEL_SIGNAL_T;

如果 Pns application manager 收到第三方应用发过来的需要销毁通道的消息，直接给 WNS TASK 发 MSG_WNS_REVOKE_CHANNEL 消息。

消息处理流程：

```
PUBLIC void MMIWNS_DispatchSignal(WNS_STATUS_RESPONSE_SIGNAL_T* signal_ptr)
{
    if(WNS_MSG_RESPONSE == signal_ptr->SignalCode)
    {
        switch (signal_ptr->response_which_message)
        {
            case MSG_WNS_REG_WNS:
            {
                //处理 WNS 服务注册的响应消息。
                //设置 WNS 服务的注册状态。
                break;
            }
            case MSG_WNS_CONNECT_WNS:
            {
                //处理 WNS 服务连接服务器的响应消息。
                //设置 WNS 服务连接状态。
                break;
            }
            case MSG_WNS_CREATE_CHANNEL:
            {
```

```
            //处理 WNS 服务创建推送通知通道的响应消息。
            //将通道信息通过回调函数传递给第三方应用。
            break;
        }
        case MSG_WNS_STOP_WNS:
        {
            //处理关闭 WNS 服务的响应消息。
            break;
        }
        case MSG_WNS_DISCONNECT_WNS:
        {
            //处理断开 WNS 服务连接的响应消息。
            break;
        }
        case MSG_WNS_REVOKE_CHANNEL:
        {
            //处理销毁 WNS 服务通道的响应消息。
            break;
        }

        default:
            break;

}
else if(WNS_MSG_NOTIFICATION == signal_ptr->SignalCode)
{
```

```
            //处理收到的第三方应用服务器通过 WNS 服务器推送的通知信息。
            //将通知信息传递给 Idle Screen 显示。
        }
    }
}
```

当前疑问点：

1.目前 WNS Task 发给 MMI Task 的 signal_ptr->SignalCode 这个值定义还不完全。

4.2.3 Notification message parser

1.通知消息的类型，目前我们可能只支持第一种。

```
/*Indicate the push notification type*/
typedef enum
{
    WNS_TYPE_TOAST,
    WNS_TYPE_TILE,
    WNS_TYPE_BADGE,
    WNS_TYPE_RAW
}MSG_WNS_NOTIFICATION_TYPE_E;
```

处理函数：

```
MMIWNS_HandleRecvNotification(WNS_PUSH_NOTIFICATION_SIGNAL_T *signal_ptr);
```

将接受到的通知信息首先保存到 Push notification manager 保存所有第三方应用通道信息以及通知信息的链表里，然后调用 Idle Screen 提供给通知应用的接口，将通知信息在 Idle Screen 上显示出来。

```
/*****************************************************************************/
//  Description : 提供给应用的接口，保存 IDLE 界面显示的通知信息
//  Global resource dependence :
//  Author: grant.peng
//  Note:
```

```
/*****************************************************************************/
PUBLIC BOOLEAN MMIIDLE_AppendPushMsg(IDLE_PUSH_MSG_INFO *push_msg_info_ptr)

//Push notification message info
typedef struct idle_push_msg_info_tag
{
    MMI_IDLE_PUSH_MSG_TYPE_E e_push_msg_type;
    MMI_IDLE_ICON_DATA_TYPE_E icon_data_type;
    MMI_IDLE_ICON_DATA_U icon_data;
    uint32 app_id;
    char* appkey;
    wchar *title;
    uint32 title_len;
    wchar *contents;
    uint32 contents_len;
    uint32 msg_count;
    void * user_data1;
    uint32 user_data_len1;
    void * user_data2;
    uint32 user_data_len2;
    IDLE_APP_ENTRY_FUNC_T app_entry_func_ptr;
    IDLE_APP_ENTRY_FUNC_T app_left_focus_action_func_ptr;
    IDLE_APP_ENTRY_FUNC_T app_right_focus_action_func_ptr;
    struct idle_push_msg_info_tag *next_node_ptr;
} IDLE_PUSH_MSG_INFO;
```

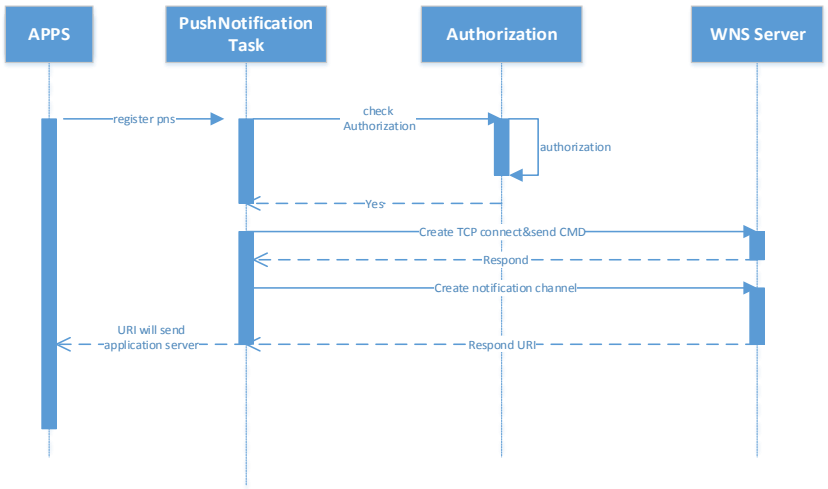
清除 Idle Screen 消息的接口：

PUBLIC void FreePushMsgByType(MMI_IDLE_PUSH_MSG_TYPE_E e_push_msg_type， uint32 app_id)

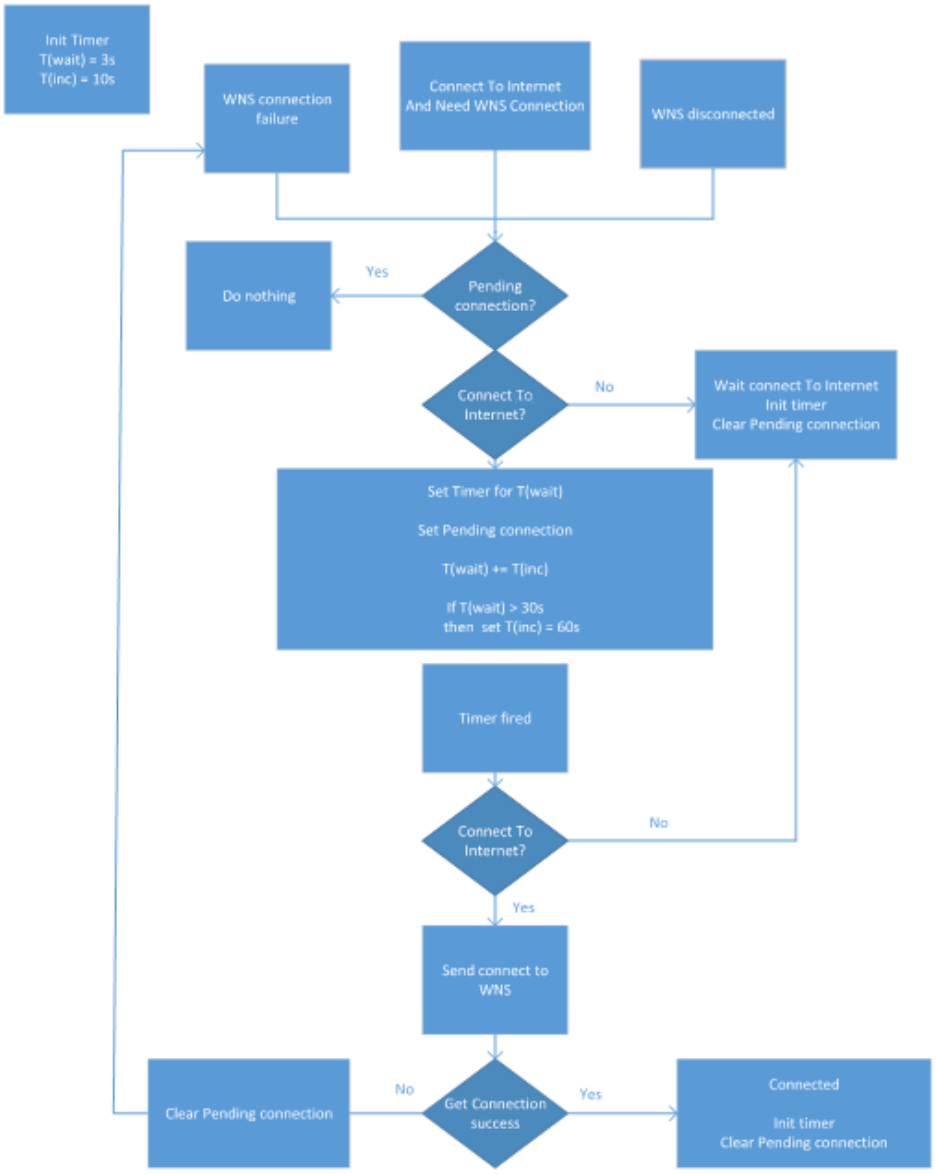
```
//Push notification message type
typedef enum
{
    MMI_IDLE_PUSH_MSG_SMS = 0,
    MMI_IDLE_PUSH_MSG_MISSED_CALL,
    MMI_IDLE_PUSH_MSG_ONGOING_CALL,
    MMI_IDLE_PUSH_MSG_WNS,
    MMI_IDLE_PUSH_MSG_TYPE_MAX
}MMI_IDLE_PUSH_MSG_TYPE_E;
```

4.3 处理逻辑流程图/状态迁移图/消息序列图

注册 WNS 服务及创建通道流程：



Pns application manager 的网络重连机制：



4.3.1 消息派发流程

说明：

可根据具体应用添加对应的说明

4.4 数据结构设计说明

4.5 export 接口描述

本章描述 WNS 主要对外接口

第5章 与外部模块间耦合度分析

5.1 WNS 与其他应用之间的关系

附录A Revision History

Version	Date	Owner	Notes
0.0.1	2015-06-11	Grant.peng	Created