

ui_mode.cpp File Reference

```
#include "ui_mode.h"

#include "com_config.h"

#include "sys.h"

#include "ui.h"

#include "epp/counterTop.h"

#include "madk_pp_protocol.h"

#include "ipc/jsobject.h"

#include "ipc/notify.h"

#include "xmldoc.h"

#include "pathutil.h"

#include "pthreadutil.h"

#include "filesystem.h"

#include "_logf.h"

#include "main.h"

#include "version.h"

#include "csd-version.h"

#include "sdikeyservice.h"

#include "loadplugin.h"

#include "sdi_text.h"
```

```

#include "timestamp.h"

#include "lib_mADK_PP_Prot/mADK_PP_Prot.h"

#include <arpa/inet.h>

#include <ifaddrs.h>

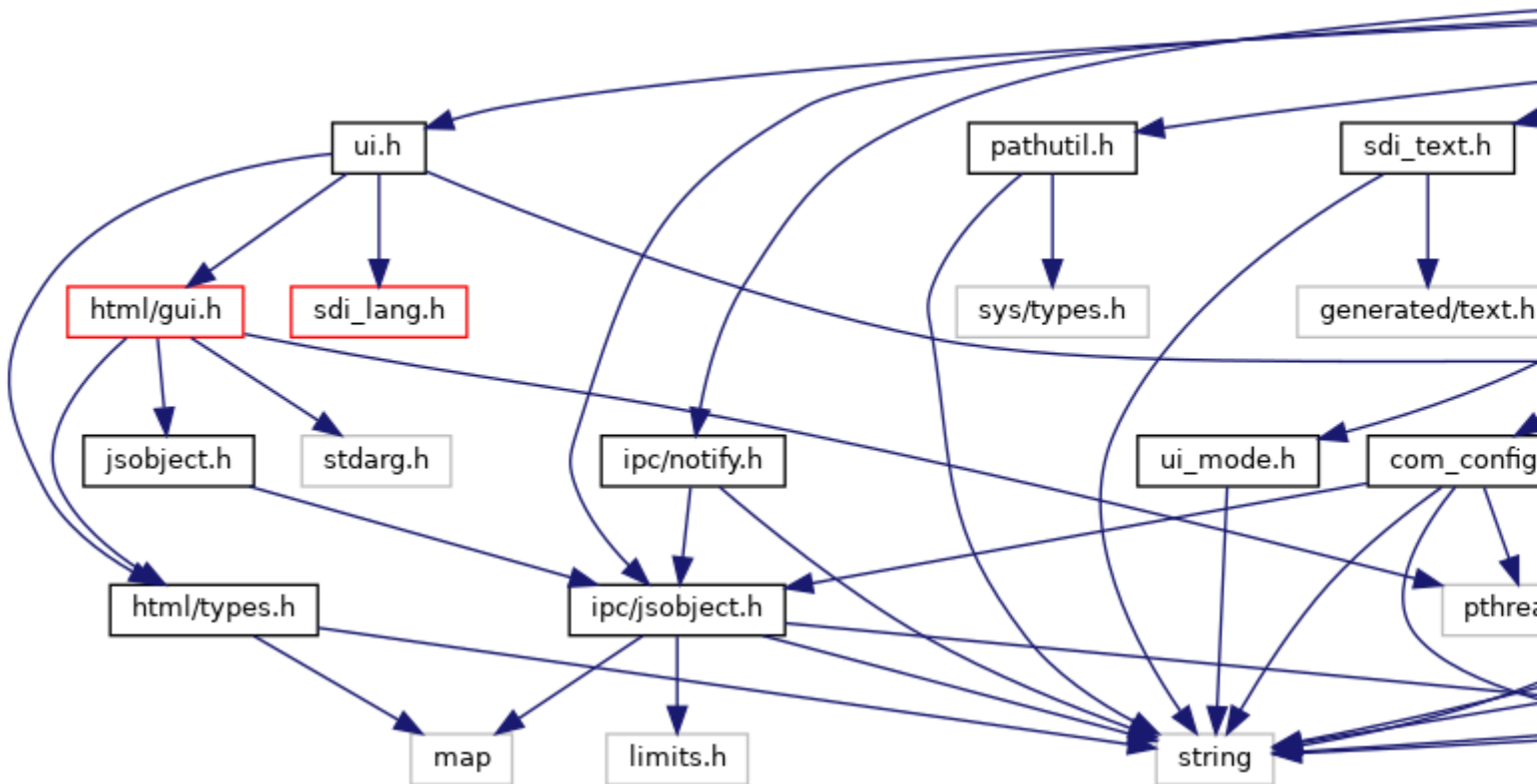
#include <sys/socket.h>

#include <netdb.h>

#include <net/if.h>

#include <sys/stat.h>
Include dependency graph for ui_mode.cpp:

```



Data Structures

```

struct ComIfTable
struct BtCom1aContext
struct IfInfo

```

Macros

```

#define FALL\_THROUGH

```

#define BT_PAIRING_TIMEOUT 120

Enumerations

CCPInterfaces {

CCP_INTERFACE_ETH0 = 0,
CCP_INTERFACE_ETH1 = 1,
CCP_INTERFACE_WLAN0 = 2,
CCP_INTERFACE_PPP_DIAL = 3,

CCP_INTERFACE_GPRS0 = 4,
CCP_INTERFACE_ETH_BT = 5,
CCP_INTERFACE_PPP_BT = 6,
CCP_INTERFACE_ETH_USB_GADGET = 7,

enum

CCP_INTERFACE_ETH_USB1_GADGET = 8,
CCP_INTERFACE_ETH_USB_HOST_GADGET = 9,
CCP_INTERFACE_PPP_USBD = 10,
CCP_INTERFACE_BRIDGE = 11,

CCP_INTERFACE_SERIAL_USBD = 12,
CCP_INTERFACE_ETH_USB0 = 13,
CCP_INTERFACE_BT = 14

}

ProfileSubType {

PST_NONE = 0,
PST_INTERNAL_ANDROID = 1,
PST_COM1A_BT_USB = 2,
PST_COM1A_USB = 3,

enum

PST_EPP_TLS = 4

}

```

    DisplayConnectStatus {
        DCS\_None = 0,
        DCS\_ComWaitOpen = 1,
        DCS\_ComOpened = 2,
        DCS\_ComConnected = 4,
enum
        DCS\_ComFailed = 8,
        DCS\_All = (DCS_ComWaitOpen | DCS_ComOpened | DCS_ComConnected | DCS_ComFailed)

    }
enum
    BTCom1AOpts { BTCom1A\_None = 0,
        BTCom1A\_Headless = 1,
        BTCom1A\_Reconnect = 2
    }
    BTPairingState {
        BT\_PairingStopped = 0,
        BT\_PairingStarted,
        BT\_ConfirmPIN,
        BT\_ConfirmPINDone,
enum
        BT\_PairingSuccess,
        BT\_PairingFailed,
        BT\_PairingTimeout,
        BT\_PairingCancelled

    }
}

```

Functions

```

void show\_idle\_connect\_status ()
void * io\_menu\_invoker (void *data)
void init\_ui\_mode ()
void * ccp\_thread\_func (void *data)
bool btStartPairing (unsigned discovery_tout_sec, enum com\_ErrorCodes *com_errno=0, bool ble=false)
void * com1a\_bt\_thread\_func (void *data)
bool select\_com\_profile (int comInterfaces, char **ComFileName)
void reset\_com\_profile ()
bool multi\_connection\_support\_enabled ()
bool comcfg\_file\_valid (const string &comcfg_file)
void protocol\_status\_ui\_update (const struct ProtStatus *status)

```

Variables

```
const struct ComIfTable comIfTable []  
enum CCPInterfaces ccp\_if\_type  
enum BTPairingState bt\_pairing\_state = BT\_PairingStopped  
enum com\_ErrorCodes bt\_pairing\_erno
```

Macro Definition Documentation

? [BT_PAIRING_TIMEOUT](#)

```
#define BT_PAIRING_TIMEOUT 120
```

? [FALL_THROUGH](#)

```
#define FALL_THROUGH
```

Enumeration Type Documentation

? [BTCom1AOpts](#)

```
enum BTCom1AOpts
```

Enumerator

[BTCom1A_None](#) no option

[BTCom1A_Headless](#) don't display wait screen to abort BT connect

[BTCom1A_Reconnect](#) force reconnect, even if device was already connected

? [BTPairingState](#)

```
enum BTPairingState
```

Enumerator

[BT_PairingStopped](#)

[BT_PairingStarted](#)

[BT_ConfirmPIN](#)

BT_ConfirmPINDone
BT_PairingSuccess
BT_PairingFailed
BT_PairingTimeout
BT_PairingCancelled

? **CCPInterfaces**

enum [CCPInterfaces](#)

Enumerator

CCP_INTERFACE_ETH0
CCP_INTERFACE_ETH1
CCP_INTERFACE_WLAN0
CCP_INTERFACE_PPP_DIAL
CCP_INTERFACE_GPRS0
CCP_INTERFACE_ETH_BT
CCP_INTERFACE_PPP_BT
CCP_INTERFACE_ETH_USB_GADGET
CCP_INTERFACE_ETH_USB1_GADGET
CCP_INTERFACE_ETH_USB_HOST_GADGET
CCP_INTERFACE_PPP_USBD
CCP_INTERFACE_BRIDGE
CCP_INTERFACE_SERIAL_USBD
CCP_INTERFACE_ETH_USB0
CCP_INTERFACE_BT

? **DisplayConnectStatus**

enum [DisplayConnectStatus](#)

Enumerator

DCS_None status display disabled for all, even for ComInterrupt, ComClose

DCS_ComWaitOpen show display updates for status ComWaitOpen

DCS_ComOpened show display updates for status ComOpened

DCS_ComConnected show display updates for status DCS_ComConnected

DCS_ComFailed show suppress display updates for status ComFailed

DCS_All

? ProfileSubType

enum [ProfileSubType](#)

Enumerator

PST_NONE no profile subtype

PST_INTERNAL_ANDROID allow this profile for Engage devices with internal Android only, e.g. CM5, M440, M424

PST_COM1A_BT_USB option to mark the profile to be used for COM1A (BT-SPP client/server and USB Host). Profile marked with PST_COM1A_BT_USB will skip profile with PST_COM1A_USB, which must be the next profile entry in table !!!

PST_COM1A_USB option to mark the profile to be used for COM1A (USB host only)

PST_EPP_TLS this profile is used for an external PINPad (EPP) using TLS over USB

Function Documentation

? btStartPairing()

```
bool btStartPairing ( unsigned                    discovery_tout_sec,  
                      enum com\_ErrorCodes * com_errno = 0,  
                      bool                        ble = false  
                      )
```

? ccp_thread_func()

```
void* ccp_thread_func ( void * data )
```

? com1a_bt_thread_func()

```
void* com1a_bt_thread_func ( void * data )
```

? comcfg_file_valid()

```
bool comcfg_file_valid ( const string & comcfg_file )
```

? init_ui_mode()

```
void init_ui_mode ( )
```

This module contains implementation, which is additionally required to support standard UI mode. Most of these implementations are UI related (e.g. to provide COM menus, idlescreen etc.) SDI variants using this module are compiled without define HEADLESS to add UI standard mode, in addition to headless mode. Finally, program parameter `-headless` will switch the SDI mode at startup. Recently supported platforms with UI support are: VOS/VOS2/VOS3 Note: This module also contains functions invoked for headless mode on these platforms, too. This is to cover the same behavior as in UI mode. (e.g. selection of a COM profile by COM_IF.CFG (COM settings file). function called once at startup to initialize UI mode of SDI

? io_menu_invoker()

```
void* io_menu_invoker ( void * data )
```

? multi_connection_support_enabled()

```
bool multi_connection_support_enabled ( )
```

For VOS/VOS2/VOS3 devices this function returns true, if SDI protocol with multi-connection support (using ADKIPC) was enabled by COM settings file. For other platforms this function just returns false.

Returns

true, if the IPC variant of the SDI protocol library shall be used on VOS/VOS2/VOS3, else false (e.g. disabled by configuration or wrong platform).

? protocol_status_ui_update()

```
void protocol_status_ui_update ( const struct ProtStatus * status )
```

? reset_com_profile()

```
void reset_com_profile ( )
```


Function to reset selected COM profile and clear current COM configuration. This will force reading of COM configuration and selection of COM profile with next call of [select_com_profile\(\)](#).

[? select_com_profile\(\)](#)

```
bool select_com_profile ( int      comInterfaces,  
                        char ** ComFileName  
                        )
```

Function invoked by SDI protocol to select and specify the used COM profile. The function passes the available COM interfaces in parameter *comInterfaces* as bitmask with values of ADKCOM enum *com_FeatureMask1*, so that SDI COM settings menu is displayed with corresponding entries with configuration options. With first invocation at startup with default settings, SDI shows up a COM setting wizard (in standard UI mode) from which the user is able to select the COM profile to use. Once the profile and settings are applied the function knows the stored settings for further SDI startups. On success, the function returns full path of the COM profile assigned to pointer of parameter *ComFileName*. The supplied buffer of this parameter is static, thus, the caller (SDI protocol) doesn't need to care about to release resources of it.

Parameters

[in] *comInterfaces* available COM interfaces, bitmask of ADKCOM enum *com_FeatureMask1*
[out] *ComFileName* full path to COM profile to use

Returns

true on success, else false on error

[? show_idle_connect_status\(\)](#)

```
void show_idle_connect_status ( )
```

function to read the connect status from the SDI protocol and to display with the idlescreen. This is usually triggered by the running SDI protocol via callback, but in special situations this is required to be triggered by SDI (e.g. in EPP mode during SDI startup).

Variable Documentation

[? bt_pairing_errno](#)

enum [com_ErrorCodes](#) *bt_pairing_errno*

[? bt_pairing_state](#)

enum [BTPairingState](#) bt_pairing_state = [BT_PairingStopped](#)

? ccp_if_type

enum [CCPIInterfaces](#) ccp_if_type

? comIfTable

const struct [ComIfTable](#) comIfTable[]

Initial value:

```
={
    {COM_LAN_1, PST_NONE, "ETH (LAN)",
     COM_PREFIX CONNECT_LAN_FILE, eth_menu, {COM_BLE,
"BT PAN", COM_PREFIX CONNECT_BT_FILE, bt_pan_menu, {COM_WIFI,
"WiFi", COM_PREFIX CONNECT_WIFI_FILE, wifi_menu, {COM_USB_SERIAL,
"USB (serial)", COM_PREFIX CONNECT_USB_SER_FILE, 0,
, {COM_LAN_USBD, PST_NONE,
"USB (LAN)", COM_PREFIX CONNECT_USB_LAN_FILE, usb_eth_menu, {COM_COM1A,
"COM1A", COM_PREFIX CONNECT_COM1A_FILE, com1a_menu, {COM_COM1A_USB,
"COM1A (USB)", COM_PREFIX CONNECT_COM1A_FILE, 0,
, {COM_BLE, PST_NONE,
"BLE", COM_PREFIX CONNECT_BLE_FILE, ble_menu, {COM_COM1,
"COM1", COM_PREFIX CONNECT_COM1_FILE, 0,
, {0,
, PST_NONE, "ANY (LAN)",
, COM_PREFIX CONNECT_ANY_FILE, eth_menu, {0,
, PST_NONE, "LOCALHOST",
, COM_PREFIX CONNECT_LO_FILE, 0, {0,
, PST_INTERNAL_ANDROID, "ANDROID (intern)",
, COM_PREFIX CONNECT_ANDROID_FILE, 0, {0,
, PST_EPP_TLS, ""
, EPP_PREFIX CONNECT_EPP_TLS, 0, }}
}
```