OCTAGON_Plugin_API_User_Manual_ Eng_v1.05 (Jan 2011)

OCTAGON SF10x8 HD *(SF1008,SF1008SE,SF1018,SF1028P)*

Installation

To use the OCTAGON Plugin API,

- Install the C/C++ compiler of SH4
- You may use the any compiler which is made target for SH4 Linux but we recommend STLinux complier.
- STLinux compiler can be download in the website as below,
  - http://www.stlinux.com/
  - ftp://ftp.stlinux.com/pub/stlinux/

When compiler installation is done, create one Working directory then copy the OCTAGON API to it.

The directory structure of OCTAGON Plugin API is as below.

./plugapi/	<Plugin API Library>
./include/	<Include Files>
./samples/	<Sample Source codes>

Structure of OCTAGON Plugin

Octagon Plugin basically should use C++ language and make a source code file as “.cpp” format to compile /link correctly.

Definition

MAIN APPLICATION is embedded Basic Firmware Application, when STB boot on after it start automatically.

PLUGIN is sub application which process simultaneously with main application.

Example, user can make his own epg guide Plugin instead of original EPG guide.

Plugin use Message Queue method to communicate with main application.

If you want to know what is Message Queue, Refer to "Message Queue" part on this documents as below.

Contact Us

If you want to know more information or request function items to add which for Octagon Plugin API,

Please contact to www.octagon-forum.com

History

Version 1.05

: **Available Main Application build version – 4136 (1.08.75)**

Modified

1. ST71xx support
2. T_ServiceHandle - SERVICE_Rec3 support.

Version **1.04**

: **Available Main Application build version – 3681 (1.08.44)**

Modified

3. PCMD_SetRes -> PCMD_SetScreen
4. T_ApplState (plugin.h)

New

1. void **SetAlarmPtr**(int alarmId, S_Alarm *alarm);

Version **1.03**

: **Available Main Application build version – 3482**

Modified

1. **ST 7105** support.

New

1. bool **GetEventDataNow**(S_Service *svc,   int *eventCount, S_EventInfo **eventInfo);
2. bool **GetEventDataNow**(int svcType, int svcNum, int *eventCount, S_EventInfo **eventInfo);

Version **1.02**

: **Available Main Application build version - 3482**

Modified

1. Simultaneous Demux Filtering support.

New

1. New T_PluginCmd and T_PluginMsg related with Simultaneous Demux Filtering.

Version **1.01**

: **Available Main Application build version - 3311**

Modified

1. Minor IPC bug fix.

New

1. PLUGINMSG_StateChangeStart   <STATEIDX_Main>

Main Application send this message when starting change the main state and then send PLUGINMSG_StateChanged message after state change is done.

2. Bool **RemoveEventData**(int satIdx, int orgNetId, int tsId, int svcId);
3. int **WiTask_Priority**(T_TaskPriority prio);

Version 1.0

: **Available Main Application build version - 3291**

Modified

1. byte ***ImageCompress**(PIXELTYPE *source, int x, int y, int width, int height, int stride = 720, int *size = NULL);

   byte ***OsdCompress**(int x, int y, int width, int height, int *size = NULL);

int *size variable is added for get compressed size.

2. void **GetEventInfo**(S_Service *svc);

   void **GetEventInfo**(int svcType, int svcNum);

PCMD_GetEpg message is changed to message thrown type.

Use below function for get event data after receive APPLMSG_EPG_Data_Ack message

bool **ReadEventData**(int satIdx, int orgNetId, int tsId, int svcId, int *eventCount, S_EventInfo **eventInfo)

3. int **GetSvcIdxById**(int svcType, word orgNetId, word tsId, word svcId, int *index=NULL);

*index is slot number of SvcIdx array that is same list with live mode Service List.

New

1. New command : PCMD_WavStart ~ PCMD_GetAlarm

2. New Message : APPLMSG_EPG_Data_Ack
   APPLMSG_ALARM_DB_Changed

APPLMSG_ALARM_Set_Fail

3. int **GetSvcIndexCnt**(int svcType);
4. int **GetSvcIdxByIndex**(int svcType, int index);

5. int **FindMatchAlarmSlot**(word evtId, word svcId, word tsId, word orgNetId);
6. S_Alarm ***GetAlarmPtr**(int alarmId);

7. void **WavStart**();
8. void **WavWrite**(void *data, int size);
9. void **WavStop**();

10. void **SetProtData**(T_PtDataId id, dword idkey, S_PtSetData data);
11. bool **GetProtData**(T_PtDataId id, S_PtGetData *data);

12. Shared Lib

Write OCTAGON Plugin source code

Plugin start with ‘PluginMain()’ and end with ‘PluginClose()’ .

NOTE : Parameter of PluginMain() is not used at the present.

When you start the PluginMain() ,

13. Perform the initialization. (Refer to Initialization item on the next.)
14. Then repeat receiving and processing the message from WiMainQ which declare in global area.

When Plugin is stop, PluginClose() function will be automatically called so you DO NOT need to manually call it.

```
// *** Sample.cpp *****
#include “plugin.h”
int PluginMain( int argc, char *argv[] )
{
        dword evt[4];

        // *** Initial Code    ***
        // *******************

        while(1)
        {
                int ret = WiQueue_Receive( WiMainQ,    (u_int8 *)evt, 16, -1 );
                if( ret == 0 )
                {
                        switch( evt[0] )
                        {
                                case MESSAGES_XXX :
                                        // *** Message process    ****
                                        break;
                        }
                }
        }
}

void PluginClose()
{
        // *** Termination code ********
        // ******************************
}
```

Initialization

Before PluginMain() is called, basic initialization (OSD, Memory, Task, Demux, etc..) will be done but few additional functions will not be initialized.

If you want to use additional functions below, start to PluginMain() function then SHOULD call the initialization and termination functions.

- REMOTE CONTROLLER

int WiFront_Init();

Make the state to be able to receive the Remote Controller in Plugin.

int WiFront_Term();

End the state receiving Remote Controller.

Must call when off the Plugin.

- FONT

bool FontInit(char *fontFile = NULL);

Call when you want to print out the text by using Font in Plugin.

If you do not set the FontFile, then /var/etc/sfam.ttf file will be used.

void FontClose();

End of using Font.

Must call when off the Plugin.

Message Queue

POSIX Message Queue is method for Inter-Process Communication (IPC)

It is almost same with octagon Message Queue method..

Plugin repeats receiving and processing the message from WiMainQ which is shared memory area between Plugin and Main Application.

You SHOULD use the WiMainQ of global variable to get the message easily.

The prototype to receive message from the queue is as below.

```
int WiQueue_Receive(int id, u_int8 *buf, int length, int tmo);
```

There are 2 types of **RETURN VALUE** for Plugin API functions.

1. Error Code

In this case, API returns error code which means success process function or detect some kind of errors.

2. Message Code(Event ID)

In this case, the message code that received through return value will send to WiMainQ when correspond event occurred.

For example, WiDemux_FilterStart() which starts the demux PID filter. (It is also return the message code.)

1. It called by Plugin, API beginning Demux filter start and return value (message code) to Plugin.
2. When demux filter loaded, it sends message code which is same with API return value to WiMainQ.
3. Plugin compare API return value with message code from WiMainQ. If they are same, you can detect correspond filter is processed.

The important message uses in Plugin is as below.

You can check all about the messages in [3. Messages]

MSG_BASE_REMOCON

: When input remote key

APPLMSG_SI_CUR_PMT

: When PMT of present channel is loaded

APPLMSG_SI_PAT

: When PAT of present TP is loaded

APPLMSG_SI_NIT

: When NIT of present TP is loaded

APPLMSG_SI_SDT

: When SDT of present SDT is loaded

APPLMSG_SI_TDT

: When TDT of present TP is loaded

APPLMSG_SI_CAT

: When CAT of present TP is loaded

APPLMSG_SVC_Start

: When service started

APPLMSG_SVC_Stop

: When service stopped

APPLMSG_TUNER_LOCK(tuner)

: When correspond tuner locked

APPLMSG_TUNER_UNLOCK(tuner)

: When correspond tuner is unlocked

Send messages or commands to Main Application

To send the message or command to Main Application, use the function as below.

void SendCommand(dword cmd, ...);

Each command in Plugin.h is defined as following.

You can check the detail about each command in [4. Commands]

```
typedef enum
{
        PCMD_Key                = 0xf1230001,
        PCMD_SetConfig,         // idx, value
        PCMD_GetEpg,            // satIdx, orgNetId, tsId, svcId
        PCMD_SvcStop,           // mainSub
        PCMD_SvcStart,          // mainSub, svcType, svcNum
        PCMD_SvcMove,           // mainSub, upDown
        PCMD_ChannelSetIdx,     // tuner, tpIdx
        PCMD_ChannelSet,        // tuner, satIdx, freq, sr, polar
        PCMD_SetVolume,         // volume %
        PCMD_SetAudio,          // (0- manual, 1- isAuto, 2- off), pid, isAc3, soundMode
        PCMD_SetScreen,         // [1]- _display_resolution_ idx, [2]- _display_format_ idx,
                                // [3]- _screen_ratio_ idx : value will be applied if it is not (-1).
                                // [4]- decode_wh(width<<16|height), [5]- decode_xy(x<<16|y),
                                // [6]- view_wh(width<<16|height),
                                // [7]- view_xy(x<<16|y) : if _wh is 0 don't apply
        PCMD_BlockKey,          // value
        PCMD_ReleaseKey,        // value
        PCMD_SetState,          // mainState
        PCMD_Blit,              // x, y, w, h, image, mask, mode
        PCMD_Blit32,            // x, y, w, h, image, mask, swidth
        PCMD_SaveState,         // stateidx, state
        PCMD_RestoreState,      // stateidx
        PCMD_WavStart,
        PCMD_WavWrite,
        PCMD_WavStop,
        PCMD_NotUsed4,
        PCMD_NotUsed5,
        PCMD_NotifyMsg,         // send message.
        PCMD_UartCtrl,          // uart control
        PCMD_ProtectedData,     // protected data store.

        PCMD_NormalUnit,                  // block Live mode unit: msg[0]- PCMD_NormalUnit, msg[1]-
```

```
                                block(1)/release(0), msg[2]- UNIT_xxxx

    PCMD_DestroyAndSaveSubState,    // destroy all popup items and set substate to data1
    PCMD_SetAlarm,                  // (evtId<<16 | svcId), (tsId<<16 | orgNetId),
                                    (svcType<<16 | svcIdx), satIdx, sTime, eTime,
                                    (alarmType<<24 | alarmMode<<16 | alarmDays<<8 | alarmPwOff)
                                    // ok- APPLMSG_ALARM_DB_Changed,
                                    // not ok- APPLMSG_ALARM_Set_Fail

    PCMD_GetAlarm,                  // Ack- APPLMSG_ALARM_DB_Changed.
                                    main s/w Send APPLMSG_ALARM_DB_Changed message
                                    when alarm db is changed too.
    PCMD_FilterStart,
    PCMD_FilterStop,

    PCMD_SetControl,                // Register control.
    PCMD_ClearControl,              // Clear control.

    PCMD_EnableKey,                 // Internal use only for rcu ctrl
    PCMD_DisableKey,                // Internal use only for rcu ctrl

    PCMD_SvcStartInternalTest,      // Internal use only - mainSub, svcType, svcNum, service_tune_flag
    PCMD_SvcStopInternalTest,       // Internal use only

    PCMD_StopVideoStream,           // sel
    PCMD_SetVideoStream,            // sel, pid

    PCMD_GetSvcState,               // sel(main, sub)

} T_PluginCmd;
```

PCMD_NormalUnit param.

```
#define UNIT_InfoBox            0x01

#define UNIT_DirectBar          0x08    // simple service list.
```

Messages

All messages from WiMainQ is with 4 x 32bit.

MESSAGE_CODE	DATA	DATA	DATA

In this code, the first 32bit is message code which is with message type + private code.

This message type uses high 8bit. The mask regarding the message type is defined as below;

```
#define MSG_BASE_KEY                    0x01000000
#define MSG_BASE_REMOCON                0x02000000


#define MSG_BASE_MASK                   0xff000000
```

By using MSG_BASE_MASK, you could easily recognize which type of message is incoming.

The rest 3 x 32bit is with data using each message

Message type that is using in WiMainQ is as below;

MSG_BASE_REMOCON <data>

The message type occurred by receiving the key from remote control

<data> is repeat code + key code and mask is defined as below;

```
#define MSG_MASK_KEY_REPEAT             0x00010000
#define MSG_MASK_KEY_CODE               0x000000ff
```

MSG_BASE_FILTER <data>

The message type when demux filter loading is successful or fail, the data compose as below;

MSG_BASE_FILTER | Private Code	Data	Num of section	Num of loaded section

```
if (data == NULL)
{
        Demux filter cannot loading data while a timeout.
}
else
{
        When loading section or table - loaded data's pointer array
        When CONTINOUS_MODE loading - loaded data's pointer
}
```

MSG_BASE_TIMER <data>

The message type occurred by timer function (WiTimer_XXX)

MSG_BASE_TUNER <data>

The message type occurred by tuner locking or unlocking.
LOCK = MSG_BASE_TUNER | Channel_CommandId, 0, NULL
UNLOCK = MSG_BASE_TUNER | Channel_CommandId, 1, NULL

Message code uses in WiMainQ is as below;

APPLMSG_SI_PAT , pid, <tuner>
: When PAT of present TP is loaded.
APPLMSG_SI_PMT, pid, <tuner>
: When PMT of present TP's temporary service is loaded.
APPLMSG_SI_NIT , pid, <tuner>
: When NIT of present TP is loaded.
APPLMSG_SI_SDT , pid, <tuner>
: When SDT of present TP is loaded.
APPLMSG_SI_TDT , pid, <tuner>
: When TDT of present TP is loaded.
APPLMSG_SI_TOT , pid, <tuner>
: When TOT of present TP is loaded.
APPLMSG_SI_BAT , pid, <tuner>
: When BAT of present TP is loaded.
APPLMSG_SI_CAT , pid, <tuner>
: When CAT of present TP is loaded.
APPLMSG_SI_CUR_PMT, <service handle>
: When PMT of the current watching or recording service is loaded.
APPLMSG_SI_ALL_PMT, <tuner>
: When all PMT of present TP is loaded.
APPLMSG_Positioner_Move
: When positional movement started.
APPLMSG_Positioner_Move_End
: When positional movement stopped.
APPLMSG_EPG_Data_Ack
: Ack of PCMD_GetEpg message.
APPLMSG_ALARM_DB_Changed
: Ack of PCMD_GetAlarm or when operation is ok as a PCMD_SetAlarm or Alarm data is changed on main s/w
APPLMSG_ALARM_Set_Fail
: Ack of PCMD_SetAlarm message if operation is failed.
APPLMSG_SVC_Start, svcHandle, svcType , svcNum
: when service started
<svcHandle> = { SERVICE_Main, SERVICE_Sub … }
<svcType> = { SVCTYPE_Tv, SVCTYPE_Radio }
<svcNum> = Index number of service
APPLMSG_SVC_NewSvcStart
: when new service started.
APPLMSG_SVC_Stop, svcHandle
: when service stopped.

APPLMSG_REC_START, (record id), (alarm id << 8 | tuner) , (svcType << 16 | svcNum)
: when recording started.
APPLMSG_REC_STOP, (record id), (alarm id) , (svcType << 16 | svcNum)
: When recording stopped

Messages related to the tuner are separately defined as below;

```
#define   APPLMSG_TUNER_LOCK( tuner )       \
                                        ( APPLMSG_TUNER_BASE | tuner )
#define   APPLMSG_TUNER_UNLOCK( tuner )  \
                                        ( APPLMSG_TUNER_BASE | 0x00010000 | tuner )
#define   APPLMSG_TUNER_SAME( tuner )       \
                                        ( APPLMSG_TUNER_BASE | 0x00020000 | tuner )
```

APPLMSG_Destroy,
: When request to Close for another operation to be work in application.
APPLMSG_PluginSelectedCmd, (process id),
: When operating program is selected in Plugin menu.

Commands

Information below about the define is the value of commands uses in ***SendCommand(dword cmd, ...);***

PCMD_Key, <Key Code>
Progress the <Key Code> in Main Application by remote control key.

PCMD_SetConfig <idx> <value>
Change the configuration value of Firmware staus.
<idx> = configuration index
<value> = configuration value
Configuration index is defined with T_Config in “plugin.h"
Use the function as below when get the value of Configuration.
int GetConfig(int cfgIdx);

PCMD_GetEpg, <satIdx> <orgNetId> <tsId> <svcId>
Request EPG data which is stored in memory.
DO NOT use this command immediately. Use the function as below.
void GetEventInfo(S_Service *svc);
void GetEventInfo(int svcType, int svcNum);

Get EPG data after receive APPLMSG_EPG_Data_Ack message using below.

bool ReadEventData(int satIdx, int orgNetId, int tsId, int svcId, int *eventCount, S_EventInfo **eventInfo);

PCMD_SvcStop, <service handle>

Stop the service.

<service handle > = { SERVICE_Main, SERVICE_Sub … }

PCMD_SvcStart, <service handle> <svcType> <svcNum>

Start the service.

<service handle> = { SERVICE_Main, SERVICE_Sub … }

<svcType> = { SVCTYPE_Tv, SVCTYPE_Radio }

<svcNum> = Index number of service

PCMD_SvcMove, <service handle> <move direction>

Direction Up/down to change the service.

<service handle> = { SERVICE_Main, SERVICE_Sub … }

<move direction> = +1 or -1

PCMD_ChannelSetIdx, <tuner> <tpIdx>

Tune to other exist TP using TP index.

<tuner> = tuner number

<tpIdx> = index of transponder

PCMD_ChannelSet, <tuner> <satIdx> <freq> <sr> <polar>

If channel data has no exact TP, You make STB tune a TP which you want using arguments.

PCMD_SetVolume, <%of volume>

Control the volume

PCMD_SetScreen, <resolution>, <format>, <ratio>

Control the screen display

<resolution> = index of _display_resolution_

<format> = index of _display_format_

<ratio> = index of _screen_ratio_

parameter value (-1) is ignore.

Ack message - PLUGINMSG_DisplayChanged.

PCMD_BlockKey, <key code>

Block the processing the key code in Main Application.

If you want the specific key to process in Plugin, you SHOULD use PCMD_BlockKey to prevent the duplication in Main Application.
NOTE : <Key code> is at 0xff, block all keys processing.

PCMD_ReleaseKey, <key code>

Given key code to be process again in Main Application.
If the specific key process is blocked by using PCMD_BlockKey, then you SHOULD release the block using the PCMD_ReleaseKey when Plugin or function is closed.
NOTE : <key code> is at 0xff, releases the all blocked key processing.

PCMD_SetState, <state index>

Change the state of Main Application.
Forcedly change to specific menu, EPG, Service List, etc…
<state index> = index of main firmware state

PCMD_Blit, <x> <y> <w> <h> <image> <mask> <pixel format>

Blit image to the screen using the hardware acceleration.
The image address for the Blit must be exist in the memory which is from via GetBlitBuf() function.

<mask> = 32bit bitmap mask. Default = 0xffffffff
<pixel format> = Pixel format of source image. It defined in Osd.h

PCMD_Blit32, <x> <y> <w> <h> <image> <mask> <stride>

Blit image to the screen using the hardware acceleration.
It SHOULD use when pixel format is 32bit ARGB.
Use this commend if the stride and width of the source is different.

PCMD_SaveState, <stateidx>, <state>

Store the state of the present Main Application and change the value of the state into new state number.
NOTE : the actual state would not change. Only the value would be change.
<stateidx> = { STATEIDX_Main, STATEIDX_Sub }

PCMD_RestoreState, <stateidx>

Return to the previous state number stored by PCMD_SaveState.

PCMD_WavStart,

Initialize wave function of Main Application. Use **WavStart()** function.

PCMD_WavWrite,

Send wave data to Main Application. Do not use this command to immediately.

Use **WavWrite(void *data, int size)** function.

NOTE: **PLUGINMSG_WavDone** should be sent on Main Application when data process is done.

PCMD_WavStop,

Close wave function of Main Application. Use **WavStop()** function.

PCMD_NotifyMsg, <message>

Send message to Main Application message queue. It should be sent to plugin process too if it is permitted message to plugin. Ex> APPLMSG_xxx

PCMD_ProtectedData,

Store data to protected Flash area. Do not use this command to immediately.

Use **SetProtData(T_PtDataId id, dword idkey, S_PtSetData data)** function

PCMD_NormalUnit, <state> <unit id>

Block or release Live mode unit.

<state> = 1: block, 0: release

<unit id> = UNIT_InfoBox / UNIT_DirectBar

PCMD_DestroyAndSaveSubState, <sub-state>

Destroy all popup items(STATEIDX_Sub) using APPLMSG_Destroy and store the sub-state of the present Main Application and change the value of the state into new sub-state number

Ack- PLUGINMSG_StateChanged <STATEIDX_Main> <STATEIDX_Sub>

PCMD_SetAlarm, <evtId<<16 | svcId>, <tsId<<16 | orgNetId>, <svcType<<16 | svcIdx>, <satIdx>, <sTime>, <eTime>, <alarmType<<24 | alarmMode<<16 | alarmDays<<8 | alarmPwOff>

> Set alarm as a parameter, if set is ok APPLMSG_ALARM_DB_Changed should be sent or not Main Application send APPLMSG_ALARM_Set_Fail message.

PCMD_GetAlarm,

> Ack- APPLMSG_ALARM_DB_Changed message.

PCMD_FilterStart, PCMD_FilterStop, PCMD_SetControl, PCMD_ClearControl,

> Internal use only.

Functions

- DEBUG

To print out the Debug message, use the function as below.

Debug functions would not print out if DEBUG is not defined as below

Printf(), DumpMemory(), Puts()

Usage to define DEBUG as below

#define DEBUG

void **Printf**(char *format, ...);

Print out the formatted text.

It used same format with **printf()** function which included ANSI C standard library.

Ex> *Printf("Test Printf : %d, %s\n", _num, _str);*

Result > *Test Printf : 5, good*

void **DumpMemory**(const void *mem, int size, char *name);

Read as **size** of the data located at **mem** after **name** string.

Ex> *DumpMemory(ptr, 0x20, "Dump1");*

Result >

```
[Dump1]

3c 73 63 72 69 70 74 3e 0a 0d 09 09 09 09 09 6c          <script>…….l

6f 63 61 74 69 6f 6e 2e 68 72 65 66 3d 27 68 74          ocation.herf='ht
```

void **Puts**(char *str);

Print out the text.

It always prints out without DEBUG definition.

- REMOTE CONTROLLER

int **WiFront_Init**();

Make the state able to receive the input of Remote Controller in Plugin.

int **WiFront_Term**();

Close the state of Remote controller input.

It SHOULD call when closing Plugin.

int **Front_IrRepeatConfig**(int wait, int repeat);

Set up the auto repeat IR code function.

<wait> : value of first delay. **<repeat>** : value of delay when repeat.

int **Front_IrSetCode**(int rCustomCode, int rPowerCode);

Set the custom code and power code of remote controller.

int **Front_IrGetCode**(int *rCustomCode, int *rPowerCode);

Get the current custom code and power code.

Key value define

#define REM_UP 0x0 : Up key define

#define REM_DOWN 0x1 : Down key define

#define REM_RIGHT 0x2 : Right key define

#define REM_LEFT	0x3	: Left key define
#define REM_MENU	0x4	: Menu key define
#define REM_OK	0x1f	: OK Key define
#define REM_FAVORITE	0x41	: Fav key define
#define REM_MUTE	0xc	: Mute key define
#define REM_GUIDE	0x8	: Guide(epg) key define
#define REM_EXIT	0x1c	: Exit key define
#define REM_INFO	0x6	: info key define
#define REM_TVRADIO	0x1a	: TV, Radio convert key define
#define REM_PLAYLIST	0x40	: playfilelist menu key define
#define REM_AUDIOTRK	0x49	: audio setup key define
#define REM_SUBT	0x0b	: subtitle setup key define
#define REM_0	0x10	: Number 0 ~ 9 key define
#define REM_1	0x11	
#define REM_2	0x12	
#define REM_3	0x13	
#define REM_4	0x14	
#define REM_5	0x15	
#define REM_6	0x16	
#define REM_7	0x17	
#define REM_8	0x18	
#define REM_9	0x19	
#define REM_F1	0x4b	: red key(function number 1) define
#define REM_F2	0x4a	: green key(function number 2) define
#define REM_F3	0x49	: yellow key(function number 3) define (*same as REM*_AUDIOTRK)
#define REM_F4	0x48	: blue key(function number 4) define
#define REM_SLEEP	0x1e	: sleep setup key define

#define REM_TELETEXT	0x0d	: teletext key define
#define REM_RECALL	0x09	: recall key define
#define REM_PGUP	0x44	: page up key define
#define REM_PGDOWN	0x43	: page down key define
#define REM_PAUSE	0x07	: pause key define
#define REM_PREV	0x50	: prev(Move to the beginning) key define
#define REM_NEXT	0x4c	: next(Move to the end)key define

- OSD

OCTAGON HD pvr/stb uses 32bit (Real true color) ARGB mode in OSD.

#define PIXELTYPE **dword** : color type define

#define ARGB(a, r, g, b) (((a)<<24) | ((r)<<16) | ((g)<<8) | ((b)))

a (alpha(transparency)), r (red), g (green), b (blue) value to setup as 0~0xff(256) color value.

Width and height of OSD window is defined as below.

#define OSDWIDTH **720**

#define OSDHEIGHT **576**

dword ***GetOsdAddr**();

[Return Value] : Graphic device memory address to direct access.

void ***GetBlitBuf**();

Buffer allocates by the Main Application and get 720 x 576 x 4 bytes.

[Return Value] : memory buffer for Bit.

To use the Blit function, you SHOULD use the memory got from this function.

void **Blit**(int x, int y, int w, int h, void *image, dword mask, int pixel_format);

Blit image using H/W acceleration.

<mask> : 32bit bitmap mask which will be use in Blit. Default = 0xffffffff
<pixel_format> : pixel format of source image. Defained in Osd.h.

byte ***ImageCompress**

(PIXELTYPE *source, int x, int y, int width, int height, int stride = 720, int *size = NULL);

<source> : ARGB PIXEL image address.

[Return Value] : Image compression to RLE type

It SHOULD be free **return address** after using it.

***size** = compressed data size.

int **ImageDecompress**

(PIXELTYPE *dest, byte *source, int x, int y, int width, int height, int stride = 720);

Decompress the RLE compressed data to **dest**.

<dest> : OSD target memory buffer.

<source> : RLE compressed data by ***ImageCompress()*** function.

byte ***OsdCompress**(int x, int y, int width, int height, int *size = NULL);

[Return Value] : Memory buffer which compressed the sub-area on the screen.

It SHOULD be free **return address** after using it.

***size** = compressed size.

int **OsdDecompress**(byte *source, int x, int y, int width, int height);

Decompress the source data to the screen.

<source> : compression source.

void **FillBox**(int x, int y, int w, int h, dword color);

Draw color box on the screen.

<color> : 32bit true type color

void **DrawRect**(int x, int y, int w, int h, dword color, int t = 1);

Draw framed(outline)-box which is non-filled.

<t> : thickness of frame.

void **GetPixmap**(int x, int y, int w, int h, dword *buf);

Copy screen data to memory buffer.

<buf> : Memory buffer which allocated by Plugin.

NOTE : Not use to blit of H/W acceleration.

void **PutPixmap**(int x, int y, int w, int h, dword *buf);

Copy the buffer to screen. (*using s/w blit*)

NOTE : Not use to blit of H/W acceleration.

- OSD COLOR MODE DEFINE

#define OSD_MODE_4ARGB	0
#define OSD_MODE_4AYUV	1
#define OSD_MODE_8ARGB	2
#define OSD_MODE_8AYUV	3
#define OSD_MODE_16ARGB	4
#define OSD_MODE_16AYUV	5
#define OSD_MODE_16RGB	6
#define OSD_MODE_16YUV655	7
#define OSD_MODE_16YUV422	8
#define OSD_MODE_16ARGB1555	9
#define OSD_MODE_16AYUV2644	10

#define OSD_MODE_32ARGB 11

#define OSD_MODE_32AYUV 12

- FONT

bool **FontInit**(char *fontFile = NULL);

Call when you want to print out the text by using the Font in Plugin.

If you do not set the FontFile then ***/var/etc/sfam.ttf*** file will be used.

void **FontClose**();

End of using Font.

It SHOULD call when off the Plugin.

void **SetFontSize**(int size);

Set the text size of front.

void **SetFontAttr**(int setFlag, int clrFlag);

NOT USED.

void **DrawText**

(int x, int y, int width, int height, char *text, PIXELTYPE fColor = ARGB(0xff,0xff,0xff,0xff), PIXELTYPE bColor = 0, int al = 0, int val = 0, int indent = 0);

Draw text function.

<fColor> : Foreground color,

<bColor> : Background color,

<al> : Set align of horizontal line, (LEFT, CENTER, RIGHT)

<val> : Set align of vertical line, (TOP, CENTER, BOTTOM)

<indent> : Left indentations

void **GetStrWidth**(const char *text, int cc, int flags, int *width, int *height, int *base);

Calculate the height and width of the text which will be printed.

<cc> : Text length.

<flags> : NOT USED.

<base> : It is distance from vertical base to the top point of the text.

It is NOT USED but just need to pass argument.

- PLUGIN CONTROL

In the Plugin, to transfer the command to Main Application, use the function as below.

void **SendCommand**(dword *cmd);

Send the message or command to Main Application.

<cmd> : array of command and data

void **SendCommand**

(dword cmd, dword data1=0, dword data2=0, dword data3=0,

dword data4=0, dword data5=0, dword data6=0, dword data7 =0);

Send the message or command together with the argument (data1~data7) to Main Application.

- QUEUE

Plugin can create new Message Queue and Task for Multi Tasking Plugin.

int **WiQueue_Create**(int elements, int size, int type, char *name);

Creating the queue.

[Return Value] : Return Queue ID.

<elements> : Maximum number of message which will be processed

<size> : Message buffer size

int **WiQueue_Rx4**(int id, u_int32 *data, int tmo);

int **WiQueue_Receive**(int id, u_int8 *buf, int length, int tmo);

Receive the message from Queue.

[Return Value] : Zero is Success, Non-zero is Fail.

<id> : queue id, **<buf>** : message buffer address,

<length> : messsage length, **<tmo>** : timeout time

int **WiQueue_Tx4**(int id, u_int32 a0, u_int32 a1, u_int32 a2, u_int32 a3);

int **WiQueue_Send**(int id, u_int8 *buf);

Send message to Queue

[Return Value] : Zero is Success, Non-zero is Fail.

<id> : queue id, **<buf>** : message buffer address,

int **WiQueue_Delete**(int id);

Delete the created queue.

int **WiQueue_GetMaxElements**(int id);

[Return Value] : Maximum elements number of queue.

void WiQue_DeleteElement(int id, int msg0, int msg1, int msg2, int msg3);

Delete the specific element from queue.

- TASK

Plugin can create new Message Queue and Task for Multi Tasking Plugin.

Octagon Task used by POSIX Pthread in the Linux System.

int **WiTask_Priority**(T_TaskPriority prio);

[Return Value] : task priority

<prio> : TPrio_Min, TPrio_Mid, TPrio_Max,

It is desirable for Plugin Main Task to use under TPrio_Mid value.

int **WiTask_Create**(PFNTASK entry, void *arg, int stack_size, int prio, char *name);

Function shall create a new Task. It created within process.

The task is created executing **entry** with **arg** as its sole argument.

[Return Value] : task id

<entry> : Start-routine of created new task.

<arg> : Argument for entry.

<stack_size> : If set zero, it makes stack size to 16k bytes as default.

<priority> : Task priority. RECOMMAND set to LOW.

*** PRIORITY OF TASK IS DEFINED AS BELOW.**

```
enum task_priority {
        PRIO_LOWEST    = 1,
        PRIO_MID       = 20,
        PRIO_HIGHEST = 40
};
```

int **WiTask_Delete**(int id);

Delete the task and release all resource.

int **WiTask_Suspend**(int id);

Make a task status to be wait mode.

int **WiTask_Resume**(int id);

Make a task status to be running mode.

int **WiTask_Sleep**(int milli_sec);

Delay for a specified amount of time.

<milli_sec> : millisecond time of unit

int **WiTask_Id**();

Return the present task ID.

Below is function about semaphore.

Support to Semaphore to solve the "Mutual Exclusion" problem in the Multi Task Programming.

It is same concept with POSIX semaphore.

int **WiSem_Create**(int init_value); // POSIX sem_init()

Initializes the semaphore.

[Return Value] : Return allocated semaphore address.

<Init_value> : initial value for the semaphore.

int **WiSem_Delete**(int id);

Deletes the semaphore.

int **WiSem_Take**(int id, int tmo);

Decrements (locks) the semaphore pointed number.

<tmo> : timeout (-1) Wait to get semaphore until it will be available.

NOTE : refer to man page of ***sem_wait()***

timeout (zero) No wait function. Return to fail…

NOTE : refer to man page of ***sem_trywait()***

int **WiSem_Release**(int id);

Increments (unlocks) the semaphore pointed number.
NOTE : refer to man page of ***sem_post()***

- TIMER

To use for periodically process or process relate to time.

int **WiTimer_AfterCallback**(int ticks, void (*callback)(int, int), int arg);

CALLBACK function is called on an appointed time elapsed.

[Return value] : timer ID

<tick> : 10ms time of unit

<callback> : CALLBACK function that will be call.

Parameter 1 : message code,

parameter 2 : argument for ***callback()***

<arg> : The value which send to second parameter when function is called.

int **WiTimer_EveryCallback**(int ticks, void (*callback)(int, int), int arg);

CALLBACK function is called on every appointed time.

[Return value] : timer ID

int **WiTimer_AfterQueue**(int ticks, int queue);

Send the message code to queue when appointed time is elapsed.

If you want to send **message code** to Main Application, **<queue>** set to ***WiMainQ.***

[Return value] : timer ID

int **WiTimer_EveryQueue**(int ticks, int queue);

Send the message code to queue on every appointed time.

If you want to send **message code** to Main Application, **<queue>** set to ***WiMainQ.***

[Return Value] : timer ID

unsigned long **GetTick**(void);

unsigned int **WiTimer_Ticks**();

[Return Value] : the 10ms unit counter. It begins after boot on system.

long long **WiTimer_TicksMs**();

[Return Value] : the 100ms unit counter. It begins after boot on system.

- DEMUX FILTER

It SHOULD be careful to use Demux PID Filter, it caused serious problem on the Main Application.

- ***WiDemux_FilterStart(),***
- ***WiDemux_FilterStop()***

HIGHLY RECOMMAND : These 2 functions use DEMUX_CHANNEL3 to start Demux Filter. This channel is not used on Main Application part to filter PID.

int **WiDemux_FilterStart**

(int sel, int pid, int size, u_int8 *filter, u_int8 *mask, int sections, int continues, int queue, int callback, int timeout, void (*fcallback)(unsigned char *, void *context) = NULL , void *context = NULL);

Start the demux filter after setting.

[Return Value] : message code

<callback> : It will be called when data is loaded or not for a timeout.

<sel> : demux channel

Use the ***GetCurSvcDmxChannel()*** function for demux channel.

<filter> : filter value

<mask> : filter mask

<Sections> zero : table loading, other : number of sections to load

<queue> : zero : WiMainQ , other : Que id

<callback> : zero : not use callback, 1 : use callback function

<timeout> : 100ms unit of time (10 = 1sec)

int **WiDemux_FilterStop**(int msgCode);

Stop the started demux filter.

Even demux filter is loaded or timeout is occurred, you SHOULD free the allocated resource by calling this function.

<msgCode> : Returned message code from WiDemux_FilterStart() function.

void **WiDemux_SetCallback**(int evtCode, void (*callback)(unsigned char *, void *context) , void *context);

Set the CALLBACK function which will be called when demux filter is loaded.

It SHOULD be call after start ***WiDemux_FilterStart()*** to get msgCode.

- SERVICE AND STB DATA

S_Service * **GetCurService**(int serviceHandle);

Get the S_Service data of current watching service or recording.

[Return Value] : pointer of struct S_Service

<serviceHandle> : activated service which you want.

“Plugin.h” is defined as following.

It is different function to ***GetCurServiceInfo().***

Please, refer to ***GetCurServiceInfo()*** as below.

```
typedef enum
{
        SERVICE_Main = 0,
        SERVICE_Sub,
        SERVICE_Rec1,
        SERVICE_Rec2,
        SERVICE_Rec3,
        N_ServiceHandle,
} T_ServiceHandle;
```

```
 typedef enum
 {
        SVCTYPE_Tv,
        SVCTYPE_Radio,
        SVCTYPE_Default,
 } T_SvcType;
```

Refer to the “stbdata.h” for struct S_Service.

NOTE : DO NOT revise or free the contents of received data as the service information will be changed incorrectly.

S_Service ***GetSvc**(byte tvRadio, word svcIdx);

[Return Value] : The service data if TV or radio list which is correspond to svcIdx.

S_Service* **GetSvcById**(int svcType, word orgNetId, word tsId, word svcId);

[Return Value] : The service data which is correspond to information as below.

<svcType> : service type [TV / Radio]

<orgNetId> : original network ID

<tsId> : TS ID

<svcId> : Service ID

int **GetSvcIdxById**(int svcType, word orgNetId, word tsId, word svcId, int *index=NULL);

[Return Value] : The service index which is correspond to information as below.

<svcType> : service type [TV / Radio]

<orgNetId> : original network ID

<tsId> : TS ID

<svcId> : Service ID

<index> : slot number of index array that is same list with live mode Service List.

int **GetSvcIndexCnt**(int svcType);

[Return Value] : Count of available slot which is index array.

<svcType> : service type [TV / Radio]

NOTE : The value of index array is svcIdx (Service Index)

int **GetSvcIdxByIndex**(int svcType, int index);

[Return Value] : Service Index

<svcType> : service type [TV / Radio]

<index> : slot number of index array

NOTE : If index is same or bigger than **GetSvcIndexCnt(svcType)** return (-1);

byte ***GetSvcName**(byte tvRadio, word svcIdx, bool shortName);

[Return Value] : The service name pointer.

<svcIdx> : Service Index.

<shortName> : TRUE – short name, FALSE – full name

byte ***GetSvcName**(S_Service *svc, bool shortName);

[Return Value] : The service name pointer.

<svc> : Service pointer.

<shortName> : TRUE – short name, FALSE – full name

S_ServiceInfo ***GetCurServiceInfo**(int serviceHandle);

Get the Service Information (SI) of activated service which current watching or recording.

If return value is NULL, it means that service is not activated.

Basic information such as PAT, PMT, SDT, TDT, etc…, all will be automatically loaded after then start service. So, user don’t need to any action to collect SI information.

Refer to the **stbdata.h** for S_*ServiceInfo*

NOTE :

1. You SHOULD need to check if return value is NULL. (To prevent segment fault.)

2. DO NOT directly use the pointer for *audioInfo*, *subInfo*, *pmtData* in S_*ServiceInfo*. It is cause to crash memory, so please use the function as below.

GetAudioInfo(), GetSubtInfo(), GetPmtData()

S_ServiceInfo ***GetServiceInfo**(S_Service *svc);

S_ServiceInfo ***GetServiceInfo**(int svcType, int svcNum);

Get the Service Information (SI) of given service.

To get the service information which you want, that service must be in the activated TP (In other words, it must be in the TP which locked by tuner.)

NOTE : S_*ServiceInfo* structure can be complicated when all SI information is collected, so please check if the return value is NULL.

S_SiCat ***GetCatData**(int svcHandle);

Get CAT information given activated service.

Refer to the **stbdata.h** for S_SiCat

S_SiPmt ***GetPmtData**(int svcHandle);

Get PMT information given activated service.

Refer to the **stbdata.h** for S_SiPmt

S_AudioInfo ***GetAudioInfo**(int svcHandle);

Get Audio information given activated service.

Refer to the **stbdata.h** for S_AudioInfo

S_SubtInfo ***GetSubtInfo**(int svcHandle);

Get Subtitle information given activated service.

Refer to the **stbdata.h** for S_SubInfo

word **GetSvcId**(byte tvRadio, word svcIdx);

Get service ID which is correspond to **svcIdx** of TV or radio list.

word **GetSvcPmtPid**(byte tvRadio, word svcIdx);

Get pmt PID which is correspond to **svcIdx** of TV or radio list.

word **GetSvcOrgNetId**(byte tvRadio, word svcIdx);

Get original network ID which is correspond to **svcIdx** of TV or radio list.

word **GetSvcTsId**(byte tvRadio, word svcIdx);

Get TS ID which is correspond to **svcIdx** of TV or radio list.

word **GetSvcTpIdx**(byte tvRadio, word svcIdx);

Get TP index which is correspond to **svcIdx** of TV or radio list.

word **GetSvcSatIdx**(byte tvRadio, word svcIdx);

Get Satellite index which is correspond to **svcIdx** of TV or radio list.

word **GetSvcSatIdx**(S_Service *svc);

Get satellite index of **svc**.

int **GetCurSvcType**(int svcHandle);

Get the type of the activated service.

int **GetCurSvcTuner**(int svcHandle);

Get the ID of the tuner using for activated service.

int **GetCurSvcNum**(int svcHandle);

Get the index of the activated service.

int **GetCurSvcDmxChannel**(int svcHandle);

Get the demux channel of the activated service.

You SHOULD call this function to get demux channel correctly in the Plugin.

S_Tp * **GetTp** (word tpIdx);

S_Tp ***GetTp**(int satIdx, word orgNetId, word tsId);

S_Tp ***GetTp**(int svcHandle);

S_Tp ***GetTp**(S_Service *service);

Get TP data with various prototype.

int **GetTpIdx**(int satIdx, word orgNetId, word tsId);

Get TP index.

char* **GetTpName**(word tpIdx);

Get TP name.

dword **GetTpFreq**(word tpIdx);

Get TP frequency.

word **GetTpSr**(word tpIdx);

Get symbol rate.

word **GetTpPolar**(word tpIdx);

Get Polarity (Vertical or Horizontal).

word **GetTpOrgNetId**(word tpIdx);

Get original network ID

word **GetTpTsId**(word tpIdx);

Get TS ID.

word **GetTpSatIdx**(word tpIdx);

Get satellite index of TP.

bool **IsSameTp**(const S_Tp &tp1, const S_Tp &tp2);

Compare tp1 and tp2 to check if they are same TP.

bool **IsSameTp**(const S_Tp *tp1, const S_Tp *tp2);

Compare tp1 and tp2 to check if they are same TP.

S_Tp ***FindTpInSat**(word satIdx, dword freq, byte polar);

Find the TP using argument as above.

int **FindTpIdx**(word satIdx, dword freq, dword sr, byte polar);

Find the TP index using argument as above.

word **GetTpNumInSat**(word satIdx);

Get number of satellite total TP.

- EPG

void **GetEventInfo**(S_Service *svc);

void **GetEventInfo**(int svcType, int svcNum);

Request event data using PCMD_GetEpg message.

bool **ReadEventData**(int satIdx, int orgNetId, int tsId, int svcId, int *eventCount, S_EventInfo **eventInfo);

bool **ReadEventData**(int svcType, int svcNum , int *eventCount, S_EventInfo **eventInfo);

bool **ReadEventData**(S_Service *svc, int *eventCount, S_EventInfo **eventInfo);

Get the event count and event info data of service.

Call this function after receive APPLMSG_EPG_Data_Ack message

bool **GetEventDataNow**(S_Service *svc,   int *eventCount, S_EventInfo **eventInfo);

bool **GetEventDataNow**(int svcType, int svcNum, int *eventCount, S_EventInfo **eventInfo);

Get the event count and event info data of service.

This function will be returned after receive APPLMSG_EPG_Data_Ack message.

bool **RemoveEventData**(int satIdx, int orgNetId, int tsId, int svcId);

Remove temporary event data from memory.

After receive APPLMSG_EPG_Data_Ack message, have to call this function if don't want to read event data.

char* **GetEventName**(S_EventInfo* evt);

Get the event name of **evt**.

char* **GetShortText**(S_EventInfo* evt);

Get the event short event text of **evt**.

char* **GetExtendedText**(S_EventInfo* evt);

Get extended information text of **evt**.

- ALARM

int **FindMatchAlarmSlot**(word evtId, word svcId, word tsId, word orgNetId);

Return matched alarm id, when not found return (-1).

S_Alarm ***GetAlarmPtr**(int alarmId);

Return alarm pointer.

If alarmId is same or bigger than ALARM_NUM return NULL

If alarmId is 0 return NULL. zero idx is not used.

void **SetAlarmPtr**(int alarmId, S_Alarm *alarm);

Copy alarm value to the original alarmId slot.

- ETC

Data error inspection function (CRC value calculation)

unsigned short ***GetCrc16****(unsigned char *data, int size);*

unsigned long ***GetMpegCrc32****(unsigned char *data, int size);*

char * **GetStrBuf**(void);

Get buffer which store temporary text memory. (Maximum size of buffer is 256 characters.)

```
static char _curStrBuf = 0;
static char _strBuf[N_StrBuf][STR_BUF_SIZE] = {{0,},};
```

int **GetConfig**(int cfgIdx);

Get config value. (cfgIdx – T_Config enum index)

void **GetTunerState**(int tuner, int *isLock, int *strength, int *quility);

Get the state of tuner locking, signal strength, signal quality.

int **GetState**(T_StateIdx stateIdx);

void **SaveState**(T_StateIdx stateIdx, T_ApplState state);

void **RestoreState**(T_StateIdx stateIdx);

Please refer to Commands part.

```
PCMD_SetState,          // mainState
PCMD_SaveState,                   // stateidx, state
PCMD_RestoreState,                // stateidx
```

void **Mjd2Date**(int MJD, int *year, int *mon, int *day, int *dayOfWeek);

Change the MJD (Modified Julian Date) format to YY/MM/DD.

int **Date2Mjd**(int year, int month, int day);

[Return Value] : Modified Julian Date which calculated by **year, month, day**.

dword **GetCurTime**(int *hour = NULL, int *min = NULL, int *sec = NULL, int *mjd = NULL);

[Return Value] : The current system time and date as HH/MM/SS with MJD.

void **WavStart**();

Initialize Main Application Wave function.

void **WavWrite**(void *data, int size);

Send data to Main Application.

If data process is done **PLUGINMSG_WavDone** message should be sent.

void **WavStop**();

Close Main Application Wave function.

void **SetProtData**(T_PtDataId id, dword idkey, S_PtSetData data);

Save data to protected Flash area

<id> : Unique data id.

PtData_ReservedStart = 0xFD257000,

PtData_ReservedEnd = 0xFD257040,

PtData_Max = 0xFD257080,

<idkey> : Same id exception control. For change exist id data set idkey identical with previous idkey.

<data> : 20 Bytes.

bool **GetProtData**(T_PtDataId id, S_PtGetData *data);

<data> : dword val[8]: '0'- id, '1'- reserved, '2'~'6'- data, '7'- crc('2'~'6')

- ADDITIONAL FUNCTIONS

- *.ico

An icon will be displayed when you add *.ico icon file to the plugin/bin/

- *.descr

: Maximum up to 40 letters of explanation will be displayed when you add *.descr text file to the plugin/bin/

- shared lib

: Modify /etc/ld.so.conf

Def value
/lib
/usr/lib
/var/lib

: run ldconfig

: conf applied to /etc/ld.so.cache

www.octagon-forum.com