# Grousenet UPnP Full プログラミングマニュアル 補足資料(Appendix)

## *Appendix-A* ： UPnP 開発の手引き - デバイス実装編

*Appendix-A* では、本 UPnP ミドルウェアを使用した UPnP デバイスの実装方法例に関して、弊社提供のサンプルコードを例に記載します。

### 01) 全体の流れ

本 UPnP ミドルウェアを使用して UPnP デバイスを実装する場合の基本手順は、以下の通りです。

| Step | アクション | 説明 |
|---|---|---|
| 1 | ディスクリプションドキュメントの準備 | 実装するデバイスに応じた<br>・ デバイスディスクリプションドキュメント<br>・ サービスディスクリプションドキュメント<br>を用意します。 |
| 2 | 起動･停止処理の実装<br>アドバタイズ定期送信処理の実装 | UPnP デバイスの起動･停止処理を実装します。<br>アドバタイズの定期送信処理を実装し、UPnP 接続を可能にします。 |
| 3 | デバイスイベントハンドラ関数の実装 | デバイスイベントハンドラ関数を実装し、コントロールポイントからの各種イベントに対応した制御を実装します。 |

### 02) ディスクリプションドキュメントの準備

ディスクリプションドキュメントは、UPnP 規定に則った形で記載して下さい。
ファイルフォーマットは、必ず UTF-8 で作成して下さい。
なお、提供サンプルコードで使用している RadioDevice は、UPnP フォーラムに規定されているデバイスではなく、オリジナルのデバイスです。
以下に、Radio Device の内容を記載します。

#### ① デバイスディスクリプションドキュメント(devdesc.xml)

下記が、RadioDevice のデバイスディスクリプションドキュメントです。
記載されている内容のうち、UDN はデバイス毎に変更しなければならないデータです。
UPnP を起動する前に、MAC アドレス等の確実に重ならない値で UDN を生成し、デバイスディスクリプションドキュメントを自動生成するような制御を用意しておくと、同一製品を同一ネットワーク上で UPnP を動作させる場合の競合を防ぐことが出来ます。

☆ 提供サンプルコードでは、デバイスディスクリプションドキュメントの自動生成処理を行っております。

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
   <specVersion>
      <major>1</major>
      <minor>0</minor>
   </specVersion>
```

```
    <device>
       <deviceType>urn:schemas-upnp-org:device:Radiodevice:1</deviceType>
       <friendlyName>UPnP for S1S6xK Sample Device</friendlyName>
       <manufacturer>SEC Corp.</manufacturer>
       <manufacturerURL>http://www.hotweb.or.jp/grouse/</manufacturerURL>
       <modelDescription>UPnP Sample Radio Device 1.0</modelDescription>
       <modelName>Grousenet UPnP Series</modelName>
       <modelNumber>1.0</modelNumber>
       <modelURL>http://www.hotweb.or.jp/grouse/</modelURL>
       <UDN>uuid:Upnp-RadioEmulator-1_0-0000000000001</UDN>
       <serviceList>
          <service>
             <serviceType>urn:schemas-upnp-org:service:RadioControl:1</serviceType>
             <serviceId>urn:upnp-org:serviceId:RadioControl1</serviceId>
             <SCPDURL>/RaCoSCDP.xml</SCPDURL>
             <controlURL>/upnp/control/RadioControl1</controlURL>
             <eventSubURL>/upnp/event/RadioControl1</eventSubURL>
          </service>
          <service>
             <serviceType>urn:schemas-upnp-org:service:RadioSetting:1</serviceType>
             <serviceId>urn:upnp-org:serviceId:RadioSetting1</serviceId>
             <SCPDURL>/RaSeSCDP.xml</SCPDURL>
             <controlURL>/upnp/control/RadioSetting1</controlURL>
             <eventSubURL>/upnp/event/RadioSetting1</eventSubURL>
          </service>
       </serviceList>
    <presentationURL>/index.htm</presentationURL>
    </device>
</root>
```

② **サービスディスクリプションドキュメント(RaCoSCDP.xml/RaSeSCDP.xml)**

下記が、RadioDevice のコントロールサービスのサービスディスクリプションドキュメント(RaCoSCDP.xml)です。アクション一覧などが定義されています。

☆ 提供サンプルコードでは、サービスディスクリプションドキュメントの自動生成処理を行っております。

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>ModulationAM</name>
      <argumentList>
        <argument>
          <retval />
          <name>Modulation</name>
          <relatedStateVariable>Modulation</relatedStateVariable>
          <direction>out</direction>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>ModulationFM</name>
      <argumentList>
        <argument>
          <retval />
          <name>Modulation</name>
          <relatedStateVariable>Modulation</relatedStateVariable>
          <direction>out</direction>
        </argument>
```

```
      </argumentList>
    </action>
    <action>
      <name>ModulationTV</name>
      <argumentList>
        <argument>
          <retval />
          <name>Modulation</name>
          <relatedStateVariable>Modulation</relatedStateVariable>
          <direction>out</direction>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SetBand</name>
      <argumentList>
        <argument>
        <name>Band</name>
          <relatedStateVariable>Band</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>NewBand</name>
          <relatedStateVariable>Band</relatedStateVariable>
          <direction>out</direction>
          <retval />
        </argument>
      </argumentList>
    </action>
    <action>
      <name>IncreaseBand</name>
      <argumentList>
        <argument>
          <name>Band</name>
          <retval />
          <relatedStateVariable>Band</relatedStateVariable>
          <direction>out</direction>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>DecreaseBand</name>
      <argumentList>
        <argument>
          <name>Band</name>
          <retval />
          <relatedStateVariable>Band</relatedStateVariable>
          <direction>out</direction>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="yes">
      <name>Modulation</name>
      <dataType>i4</dataType>
      <allowedValueRange>
        <minimum>1</minimum>
        <maximum>3</maximum>
      </allowedValueRange>
      <defaultValue>1</defaultValue>
    </stateVariable>
    <stateVariable sendEvents="yes">
      <name>Band</name>
      <dataType>i4</dataType>
      <allowedValueRange>
        <minimum>0</minimum>
        <maximum>200</maximum>
```

```
          <step>1</step>
        </allowedValueRange>
        <defaultValue>78</defaultValue>
      </stateVariable>
    </serviceStateTable>
</scpd>
```

## 03) UPnP 起動関数･停止関数

ディスクリプションドキュメントの準備が出来たら、次はユーザアプリケーションを実装します。まずは UPnP デバイスの起動及び停止処理を実装します。

### 起動関数：RadioDeviceStart

起動処理は、デバイス起動時に一度だけ Call するように実装します。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | デバイスハンドルの準備 | /*****************************************************************<br>UPnP Device により供給されるデバイスハンドル<br>*****************************************************************/<br>UpnpDevice_Handle device_handle = -1; |
| 2 | config 構造体メンバ設定<br>(RadioDeviceStart 関数 Call 前) | /* config 構造体メンバ設定 */<br>memset(&config, 0, sizeof(tUpnp_Conf));<br>memcpy(config.Ipaddress,ip_address,sizeof(config.Ipaddress));<br>config.PortNo = port;<br>config.UseWebserver = DUPNP_OFF; // 既製 Web サーバ使用<br>config.TcpPortNo = TCP_PORTNO; // UPnP で使用する TCP ポート番号<br>config.UdpPortNo = UDP_PORTNO; // UPnP で使用する UDP ポート番号<br>config.UpnpTaskPri = UPNP_TASK_PRI; // UPnP タスクプライオリティ<br>config.ClsTimAcp = CLS_TIM_ACP; // TCP Connection クローズ待ちタイマー<br>config.IPTTL = DEF_IP_TTL; // SSDP にて使用する UDP TTL=4<br>config.UpnpMplSize = UPNP_BUFSIZ; // UPnP メモリプールサイズ設定 128KByte |
| 3 | ﾕｰｻﾞｱﾌﾟﾘｹｰｼｮﾝ内部データの初期化<br>(RadioDeviceStart 関数 Call 前) | // 本 APL の初期化<br>SampleUtil_Initialize();<br><br>☆SampleUtil_Initialize 関数の詳細は実際の関数実装を確認して下さい。 |
| 4 | **UpnpInit** 関数によるミドルウェアの初期化<br>(RadioDeviceStart 関数 Call 前) | ret = **UpnpInit**( &config );<br>if(ret != UPNP_E_SUCCESS )<br>{<br>//Upnp 初期化に失敗したなら<br>UpnpFinish();<br>return ret;<br>} |
| 5 | **UpnpRegisterRootDevice** 関数によるルートデバイスの登録 | // ポート番号が省略された場合の対処<br>if(port == 0)<br>{<br>port = UPNP_SAMPLE_PORT;<br>}<br>sprintf( (char *)desc_doc_url,"http://%d.%d.%d.%d:%d/%s",<br>ip_address[0],<br>ip_address[1],<br>ip_address[2],<br>ip_address[3],<br>port, desc_doc_name );<br>/* Root Device の登録 */<br>ret = **UpnpRegisterRootDevice**( (char *)desc_doc_name,<br>RadioDeviceCallbackEventHandler,<br>&device_handle, &device_handle ); |
| 6 | **UpnpRegisterRootDevice** の結果を確認。<br>**UpnpSendAdvertisement** 関数によるアドバタイズ送信の開始。 | if(ret != UPNP_E_SUCCESS )<br>{<br>//ルートデバイスの登録に失敗したなら<br>UpnpFinish(); //UPnP 終了<br>return ret;<br>}<br>else<br>{<br>//Radio デバイス状態テーブル初期化<br>RadioDeviceStateTableInit((char *)desc_doc_name, (char *)desc_doc_url );<br>☆Radio デバイス状態テーブル初期化処理の詳細は、実際の関数実装を確認して下さい。 |

| | | |
|---|---|---|
| | | // ディスクリプション(アドバタイズ送信)開始<br>ret = UpnpSendAdvertisement( device_handle, default_advr_expire );<br>if( ret != UPNP_E_SUCCESS )<br>{<br>//アドバタイズメント(広告)送信に失敗したなら<br>UpnpFinish( );//Upnp 終了<br>return ret;<br>}<br>}<br>return (ret);<br>} |

## 停止関数：RadioDeviceStop

停止処理は、デバイス終了時に一度だけ Call するように実装します。
IP アドレスの変更など、ネットワーク状態が変更になった場合や、別なデバイスを起動したい場合などは、一旦本関数でデバイスを終了し、再度デバイスを起動します。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **UpnpUnRegisterRootDevice** 関数 Call によるルートデバイスの削除 | UpnpUnRegisterRootDevice( device_handle ); //ルートデバイス登録解除 |
| 2 | **UpnpFinish** 関数による、ミドルウェアの終了 | UpnpFinish( ); //Upnp 終了 |
| 3 | ﾕｰｻﾞｱﾌﾟﾘｹｰｼｮﾝ内部データの開放 | SampleUtil_Finish( ); //サンプル終了<br>☆SampleUtil_Finish 関数の詳細は実際の関数実装を確認して下さい。 |

上記に示すとおり、UPnP デバイスの起動後は、結果を見て、**UpnpSendAdvertisement** 関数によりデバイスアドバタイズメントの定期送信機能を開始して下さい。
本関数 Call により、指定した時間間隔で、NOTIFY(ssdp:alive)が送信され、コントロールポイントにデバイスの存在を通知できるようになります。
**UpnpRegisterRootDevice** 関数で指定するデバイスイベントハンドラ関数(**RadioDeviceCallbackEventHandler**)は、下記のように実装しています。詳細はデバイスイベントハンドラの項を参照して下さい。
引数 EventType で何のイベントを受信したかを判定し、それぞれの処理関数を Call するような実装にしています。

```
ER RadioDeviceCallbackEventHandler( Upnp_EventType EventType,
                                    void *Event,
                                    void *Cookie )
{
    switch ( EventType )
    {//イベントのタイプ
        case UPNP_EVENT_SUBSCRIPTION_REQUEST: //サブスクリプション(購読)要求
        {
            RadioDeviceHandleSubscriptionRequest( ( struct
                                                  Upnp_Subscription_Request
                                                  * )Event );
            //Radio デバイスサブスクリプション要求操作
            break;
        }
        case UPNP_CONTROL_GET_VAR_REQUEST: //変数獲得要求
        {
            RadioDeviceHandleGetVarRequest( ( struct Upnp_State_Var_Request
```

```
                                        * )Event );
            //Radio デバイス変数獲得要求操作
            break;
        }
        case UPNP_CONTROL_ACTION_REQUEST: //動作要求
        {
            RadioDeviceHandleActionRequest( ( struct Upnp_Action_Request * )
                                        Event );
            //Radio デバイス動作要求操作
            break;
        }
        default: //不正なイベントタイプ
            // 無視する。
            break;
    }
    return ( E_OK );
}
```

以下に、それぞれのイベント毎の実装方法について記載します。

## 04) コールバック関数の実装 Part1－SUBSCRIBE 受信処理

SUBSCRIBE 受信時は、デバイスイベントハンドラの、★部分のルートを通ります。

```
ER RadioDeviceCallbackEventHandler( Upnp_EventType EventType,
                                    void *Event,
                                    void *Cookie )
{

    switch ( EventType )
    {//イベントのタイプ
★      case UPNP_EVENT_SUBSCRIPTION_REQUEST: //サブスクリプション(購読)要求
★      {
★         RadioDeviceHandleSubscriptionRequest( ( struct
★                                                 Upnp_Subscription_Request
★                                                 * )Event );
★          //Radio デバイスサブスクリプション要求操作
★          break;
★      }
```

RadioDeviceHandleSubscriptionRequest 関数では、以下のように、UpnpAcceptSubscription 関数を使用してサブスクリプション受付処理を実装しています。

コントロールポイントからは、サービス毎に SUBSCRIBE リクエストが来ますので、SUBSCRIBE リクエストで指定された UDN 及びサービス ID が、自分のデバイス情報と合致する場合のみ、サブスクリプション受付処理を行うようにして下さい。

```
ER RadioDeviceHandleSubscriptionRequest(
      struct Upnp_Subscription_Request *sr_event )
{

    ER_ID  ercd; // 戻り値
    UINT i ;

    ercd = wai_sem(dUPnP_SampleSemID); //セマフォ資源獲得
    if(ercd != E_OK)
    {
        return ercd; //セマフォ資源獲得失敗
    }

    for( i = 0; i < RADIO_SERVICE_SERVCOUNT; i++ )
    {
        // サービスの数(ここではコントロールとセッティング 2 つ分)だけループ
        if( ( strcmp( sr_event ->UDN, Radio_service_table[i].UDN ) == 0 ) &&
            ( strcmp( sr_event ->ServiceId, Radio_service_table[i].ServiceId )
              == 0 ) )
        { //UDN とサービス ID のチェックで問題なければ

            //サブスクリプションの受け入れ関数コール
            UpnpAcceptSubscription( device_handle,
                                    sr_event ->UDN,
                                    sr_event ->ServiceId,
                                    (char ** )Radio_service_table[i].
```

```
                                        VariableName,
                                        (char ** )Radio_service_table[i].
                                        VariableStrVal,
                                        Radio_service_table[i].VariableCount,
                                        sr_event ->Sid );
            }
        }
        ercd = sig_sem(dUPnP_SampleSemID); //セマフォ資源返却
        if(ercd != E_OK)
        {
            return ercd; //返却失敗
        }
        return 1;
    }
    /* RadioDeviceHandleSubscriptionRequest */
```

---

なお、最大サブスクリプション数を制限したい場合は **UpnpSetMaxSubscriptions** 関数、サブスクリプション有効期限を制限したい場合は **UpnpSetMaxSubscriptionTimeOut** 関数を使用します。

**UpnpSetMaxSubscriptions** 関数は、**UpnpRegisterRootDevice** 関数 Call 後、**UpnpSendAdvertisement** 関数 Call 前に本関数を Call するようにして下さい。

上記関数を使用しない場合、最大サブスクリプション数は無制限になり、サブスクリプション有効期限は基本的に、SUBSCRIBE リクエストの TIMEOUT ヘッダに設定されている値に従います。

## 05) コールバック関数の実装 Part2－SOAP ACTION 受信処理

SOAP ACTION 受信時は、デバイスイベントハンドラの、★部分のルートを通ります。

```
ER RadioDeviceCallbackEventHandler( Upnp_EventType EventType,
                                    void *Event,
                                    void *Cookie )
{
    switch ( EventType )
    {//イベントのタイプ
        case UPNP_EVENT_SUBSCRIPTION_REQUEST: //サブスクリプション(購読)要求
        {
            RadioDeviceHandleSubscriptionRequest( ( struct
                                                  Upnp_Subscription_Request
                                                  * )Event );
            //Radio デバイスサブスクリプション要求操作
            break;
        }
        case UPNP_CONTROL_GET_VAR_REQUEST: //変数獲得要求
        {
            RadioDeviceHandleGetVarRequest( ( struct Upnp_State_Var_Request
                                            * )Event );
            //Radio デバイス変数獲得要求操作
            break;
        }
    ★ case UPNP_CONTROL_ACTION_REQUEST: //動作要求
    ★ {
    ★    RadioDeviceHandleActionRequest( ( struct Upnp_Action_Request * )
    ★                                    Event );
    ★     //Radio デバイス動作要求操作
    ★     break;
    ★ }
        default: //不正なイベントタイプ
            // 無視する。
            break;
    }
    return ( E_OK );
}
```

RadioDeviceHandleActionRequest 関数以降、以下の手順で、

- SOAP ACTION の受信によるデバイスのコントロール制御。
- 状態変化に伴い、UpnpNotify 関数を使用した NOTIFY(ｲﾍﾞﾝﾄﾒｯｾｰｼﾞ) の送信。
- **UpnpAddToActionResponse** 関数を使用した SOAP ACTION レスポンスの生成及びデバイスイベントハンドラ関数の終了に伴うレスポンス送信処理。

の制御を行います。SOAP ACTION 受信時は、上記 3 つの制御を行う必要があります。

### ① 受信した SOAP ACTION の内容を解析し、デバイスのコントロール制御を行います。

```
ER RadioDeviceHandleActionRequest( struct Upnp_Action_Request *ca_event )
{
    int action_found = 0; //動作発見フラグ
    int i = 0;
```

```
int service = -1;
int retCode = 0;
char *errorString = NULL;

ca_event ->ErrCode = 0;
ca_event ->ActionResult = NULL;

if((strcmp(ca_event ->DevUDN, Radio_service_table[RADIO_SERVICE_CONTROL].UDN ) == 0) &&
   (strcmp(ca_event ->ServiceID, Radio_service_table[RADIO_SERVICE_CONTROL].ServiceId) == 0))
{
    // UDN とサービス ID のチェックを行いコントロールサービスであれば
    // Radio デバイスコントロールサービスの動作を要求する
    service = RADIO_SERVICE_CONTROL; //サービス名(コントロール)の設定
}
else if((strcmp(ca_event ->DevUDN, Radio_service_table[RADIO_SERVICE_SETTING].UDN ) == 0)&&
        ( strcmp(ca_event ->ServiceID, Radio_service_table[RADIO_SERVICE_SETTING].ServiceId ) == 0))
{
    // UDN とサービス ID のチェックを行いセッティングであれば
    // Radio デバイスセッティングサービスの動作を要求する
    service = RADIO_SERVICE_SETTING; //サービス名(セッティング)の設定
}

/*************************************************************/
/* ACTION 名称に従って、それぞれの動作を行う関数を Call します。*/
/* 関数へのポインタは、初期化処理で設定しています。          */
/*************************************************************/
// 最大動作数を超えるか、ACTION 名に NULL が出現するまでループ
for(i=0;((i<RADIO_MAXACTIONS)&&(Radio_service_table[service].ActionNames[i]!=NULL));i++ )
{
    if(!strcmp(ca_event ->ActionName,Radio_service_table[service].ActionNames[i]))
    {
        // アクション名が合致した。
        // Power が ON のときの場合、または OFF のときだが PowerOn を指示された場合
        if((!strcmp(Radio_service_table[RADIO_SERVICE_CONTROL].VariableStrVal[RADIO_CONTROL_POWER],"1")) ||
            (!strcmp( ca_event ->ActionName, "PowerOn")))
        {
            // Mute が ON のときは音量設定無効
            if(((!strcmp( ca_event ->ActionName, "SetVolume")) ||
                (!strcmp( ca_event ->ActionName, "IncreaseVolume")) ||
                (!strcmp( ca_event ->ActionName, "DecreaseVolume"))) &&
               (!strcmp(Radio_service_table[RADIO_SERVICE_CONTROL].VariableStrVal[RADIO_CONTROL_MUTE],"1")))
            {
                // Mute 中ですエラーメッセージ設定
                errorString = "Mute is On. Volume can't set.";
                retCode = UPNP_E_INTERNAL_ERROR;
            }
            else
            {
                //アクション実行
                retCode = Radio_service_table[service].actions[i](
                            ca_event ->ActionRequest,
                            &ca_event ->ActionResult,
                            &errorString );
            }
        }
```

事前に ACTION 毎に関数へのポインタ登録をしたテーブルを用意しており、該当した ACTION の関数 Call をしています。各関数ともに同じインタフェースにしており、ActionRequest に応答メッセージが設定されリターンされるような実装になっています。

：<略>

以下は、上記のアクション実行が、仮に IncreaseVolume(ボリュームを１上げる)だった場合のアクション関数です。

```
ER RadioDeviceIncreaseVolume(  IXML_Document * in,
                               IXML_Document ** out,
                               char **errorString )
{

    return UpdateVolume( 1, in, out, errorString );
    //音量値更新関数 Call して音量を上げる。

}
```

```
ER UpdateVolume(int incr, IXML_Document * in, IXML_Document ** out, char **errorString )
{
    INT curvolume, newvolume;
    char *actionName = NULL;
    char value[RADIO_MAX_VAL_LEN];
    ER_ID  ercd; // 戻り値

    if( incr > 0 ) {
        actionName = "IncreaseVolume";
    } else {
        actionName = "DecreaseVolume";
    }

    ercd = wai_sem(dUPnP_SampleSemID); //セマフォ資源獲得
    if(ercd != E_OK) {
        return ercd; //セマフォ資源獲得失敗
    }

    // 現在の値を格納
    curvolume = atoi( Radio_service_table[RADIO_SERVICE_CONTROL]. VariableStrVal[RADIO_CONTROL_VOLUME] );

    ercd = sig_sem(dUPnP_SampleSemID); //セマフォ資源返却
    if(ercd != E_OK) {
        return ercd; //返却失敗
    }
    // 新値 = 現在値＋変更値
    newvolume = curvolume + incr;

    if( newvolume < MIN_VOLUME  || newvolume > MAX_VOLUME ) {
        /* 値不正 */
        ( *errorString ) = "Invalid Volume";
        return UPNP_E_INVALID_PARAM;
    }

    /* Volume 設定に関するベンダー固有処理はここで記載します。*/

    /*********************************************/
    /* Radio デバイスサービステーブル値の更新       */
    /* この関数で、SUBSCRIBER へ NOTIFY を送信します */
    /*********************************************/
    sprintf( value, "%d", newvolume );
    ercd = RadioDeviceSetServiceTableVar( RADIO_SERVICE_CONTROL, RADIO_CONTROL_VOLUME, value );
    if( ercd ) {
```

：<略>

### ② 正常に状態変化を行った場合は、UpnpNotify 関数で、状態変化を通知します。

```
ER RadioDeviceSetServiceTableVar(UINT service, UINT variable, char *value )
{
    ER_ID ercd;
    if( ( service >= RADIO_SERVICE_SERVCOUNT )
        || ( variable >= Radio_service_table[service].VariableCount )
        || ( strlen( value ) >= RADIO_MAX_VAL_LEN ) )
    {
        // サービス数が定義数以上または、サービス内変数数以上または、
        // 動作値が定義数以上の場合は終了
        return ( 0 );
    }
    ercd = wai_sem(dUPnP_SampleSemID); //セマフォ資源獲得
    if(ercd != E_OK)
    {
        return ercd; //セマフォ資源獲得失敗
    }
    // 指定のサービスの値変更
    strcpy( Radio_service_table[service].VariableStrVal[variable], value );

    // NOTIFY送信
    UpnpNotify( device_handle,
                Radio_service_table[service].UDN,
                Radio_service_table[service].ServiceId,
                (char **)&Radio_service_table[service].VariableName[variable],
                (char **)&Radio_service_table[service].VariableStrVal[variable],
                1 );
    ercd = sig_sem(dUPnP_SampleSemID); //セマフォ資源返却
    if(ercd != E_OK)
    {
        return ercd; //返却失敗
    }

    return ( 1 );
}
/* RadioDeviceSetServiceTableVar */
```

変数の値はユーザ側で管理して下さい。

状態変化を NOTIFY(ｲﾍﾞﾝﾄﾒｯｾｰｼﾞ)で通知します。

### ③ 結果が正常な場合は、SOAPACTION レスポンスイベントハンドラ関数引数の ErrCode、ErrStr、ActionResult に結果を保存し、結果が正常な場合は、SOAPACTION レスポンスイベントハンドラ関数を終了します。

```
ER UpdateVolume(int incr, IXML_Document * in, IXML_Document ** out, char **errorString )
{
    ：<略>
    /*********************************************/
    /* Radio デバイスサービステーブル値の更新       */
    /* この関数で、SUBSCRIBER へ NOTIFY を送信します */
    /*********************************************/
    sprintf( value, "%d", newvolume );
    ercd = RadioDeviceSetServiceTableVar( RADIO_SERVICE_CONTROL, RADIO_CONTROL_VOLUME, value );
    if( ercd ) {
        /**** Action Response追加 ****/
        ercd = UpnpAddToActionResponse( out, actionName,
```

SOAP ACTION レスポンスを生成しています。

```
                    RadioServiceType[RADIO_SERVICE_CONTROL],"Volume", value );
        if( ercd != UPNP_E_SUCCESS ) {
            ( *out ) = NULL;
            ( *errorString ) = "Internal Error";
            return UPNP_E_INTERNAL_ERROR;
        }
        return UPNP_E_SUCCESS;
    } else {
        ( *errorString ) = "Internal Error";
        return UPNP_E_INTERNAL_ERROR;
    }
}
```

---

```
ER RadioDeviceHandleActionRequest( struct Upnp_Action_Request *ca_event )
{
    :<略>
                    //アクション実行
                    retCode = Radio_service_table[service].actions[i](
                                ca_event ->ActionRequest,
                                &ca_event ->ActionResult,
                                &errorString );
                }
            }
            :<略>
            action_found = 1; //動作発見
            break;
        }
    }
    if( !action_found ) { //ACTION が無い
        //エラー設定
        ca_event ->ActionResult = NULL;
        strcpy( ca_event ->ErrStr, "Invalid Action" );
        ca_event ->ErrCode = 401;
    } else {
        // 処理成功？
        if( retCode == UPNP_E_SUCCESS ) {
            // 正常完了
            ca_event ->ErrCode = UPNP_E_SUCCESS;
        } else {
            //異常終了ならエラーメッセージ設定
            strcpy( ca_event ->ErrStr, errorString );
            switch ( retCode )
            {
                case UPNP_E_INVALID_PARAM:
                {
                    ca_event ->ErrCode = 402;
                    break;
                }
                default:
                {
                    ca_event ->ErrCode = 501;
                    break;
                }
            }
        }
    }
    return ( ca_event ->ErrCode );
```

ActionRequestに応答メッセージが設定されリターンされるような実装になっています。

Action名不一致の場合の処理です。

Action 正常終了の場合のルートです。ErrStr は設定不要。ActionRequestはアクション実行時に設定されているため、ここではErrCode に UPNP_E_SUCCESS を設定するだけです。

Action 異常終了の場合のルートです。

```
}
/* RadioDeviceHandleActionRequest */
```

## 06) コールバック関数の実装 Part3－SOAP QUERY 受信処理

SOAP QUERY 受信時は、デバイスイベントハンドラの、★部分のルートを通ります。

```
ER RadioDeviceCallbackEventHandler( Upnp_EventType EventType,
                                    void *Event,
                                    void *Cookie )
{
    switch ( EventType )
    {//イベントのタイプ
        case UPNP_EVENT_SUBSCRIPTION_REQUEST: //サブスクリプション(購読)要求
        {
            RadioDeviceHandleSubscriptionRequest( ( struct
                                                    Upnp_Subscription_Request
                                                    * )Event );
            //Radio デバイスサブスクリプション要求操作
            break;
        }
    ★  case UPNP_CONTROL_GET_VAR_REQUEST: //変数獲得要求
    ★  {
    ★      RadioDeviceHandleGetVarRequest( ( struct Upnp_State_Var_Request
    ★                                      * )Event );
    ★      //Radio デバイス変数獲得要求操作
    ★      break;
    ★  }
        case UPNP_CONTROL_ACTION_REQUEST: //動作要求
        {
           RadioDeviceHandleActionRequest( ( struct Upnp_Action_Request * )
                                           Event );
            //Radio デバイス動作要求操作
            break;
        }
        default: //不正なイベントタイプ
            // 無視する。
            break;
    }
    return ( E_OK );
}
```

SOAP QUERY 受信時のデバイスイベントハンドラ関数では、引数 Event->CurrentVal に、現在の変数値を設定し、デバイスイベントハンドラ関数をリターンすると、ミドルウェア内で SOAP QUERY レスポンスを送信します。

RadioDeviceHandleGetVarRequest 関数では、

- 該当する変数値の確定。
- 現在の変数値の取得及び引数 Event->CurrentVal への設定

の制御を行います。

```
ER RadioDeviceHandleGetVarRequest(struct Upnp_State_Var_Request *cgv_event )
{
    UINT i,j;
    INT getvar_succeeded = 0; //変数獲得成功フラグ
    ER_ID ercd;
```

```
    cgv_event->CurrentVal = NULL; //現在値
    ercd = wai_sem(dUPnP_SampleSemID); //セマフォ資源獲得
    if(ercd != E_OK)
    {
        return ercd; //セマフォ資源獲得失敗
    }

    for( i = 0; i < RADIO_SERVICE_SERVCOUNT; i++ )
    { //サービスの数だけループ
        if((strcmp( cgv_event->DevUDN, Radio_service_table[i].UDN ) == 0 ) &&
           ( strcmp( cgv_event->ServiceID, Radio_service_table[i].ServiceId ) == 0))
        {
            for( j = 0; j < Radio_service_table[i].VariableCount; j++ )
            { //各サービスの機能数だけループ
                if( strcmp( cgv_event->StateVarName,
                            Radio_service_table[i].VariableName[j] ) == 0 )
                {
                    // 変数取得成功フラグ
                    getvar_succeeded = 1;
                    // 指定変数の現在値取得
                    cgv_event->CurrentVal = ixmlCloneDOMString( Radio_service_table[i].
                                                VariableStrVal[j] );
                    break;
                }
            }
        }
    }
    if( getvar_succeeded )
    {
        cgv_event->ErrCode = UPNP_E_SUCCESS;
    }
    else
    {
        // エラーコード(404)設定
        cgv_event->ErrCode = 404;
        strcpy( cgv_event->ErrStr, "Invalid Variable" );
    }
    ercd = sig_sem(dUPnP_SampleSemID); //セマフォ資源返却
    if(ercd != E_OK)
    {
        return ercd; //返却失敗
    }
    return ( cgv_event->ErrCode == UPNP_E_SUCCESS );
}
/* RadioDeviceHandleGetVarRequest */
```

CurrentVal への変数値の設定は、ixmlCloneDOMString 関数を使用します。

変数値異常などの場合は、このように ErrCode と ErrStr へエラー情報を設定します。

# *Appendix-B* ： UPnP 開発の手引き - コントロールポイント実装編

*Appendix-B* では、本 UPnP ミドルウェアを使用した UPnP コントロールポイントの実装方法例に関して、弊社提供のサンプルコードを例に記載します。

## 01) 全体の流れ

本 UPnP ミドルウェアを使用して UPnP コントロールポイントを実装する場合の基本手順は、以下の通りです。

| Step | アクション | 説明 |
|---|---|---|
| 1 | 起動･停止処理の実装 | UPnP コントロールポイントの起動･停止処理を実装します。 |
| 2 | M-SEARCH 送信処理の実装 | M-SEARCH 送信処理を実装し、UPnP デバイス検索を可能にします。 |
| 3 | SUBSCRIBE 送信処理の実装 | SUBSCRIBE 送信処理を実装し、状態変更通知登録を可能にします。 |
| 4 | UNSUBSCRIBE 送信処理の実装 | SUBSCRIBE 送信処理を実装し、状態変更通知解除を可能にします。 |
| 5 | ACTION 送信処理の実装 | ACTION 送信処理を実装し、コントロールを可能にします。 |
| 6 | QUERY 送信処理の実装 | QUERY 送信処理を実装し、状態取得を可能にします。 |
| 7 | SSDP NOTIFY 受信処理の実装 | SSDP NOTIFY 受信処理を実装し、UPnP デバイス存在確認を可能にします。 |
| 8 | SSDP BYEBYE 受信処理の実装 | SSDP BYEBYE 受信処理を実装し、UPnP デバイス消滅確認を可能とします。 |
| 9 | GENA NOTIFY 受信処理の実装 | GENA 受信処理を実装し、UPnP デバイスの状態変更通知を受信可能にします。 |
| 10 | SUBSCRIBE RENEWAL 失敗時処理の実装 | SUBSCRIBE RENEWAL 失敗の判断を可能にします。 |

## 02) 起動関数･停止関数

ユーザアプリケーションの実装を行うにあたって、まずは UPnP コントロールポイントの起動及び停止処理を実装します。

### 起動関数：CtrlPointStart

起動処理は、デバイス起動時に一度だけ Call するように実装します。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | コントロールポイントハンドルの準備 | `UpnpClient_Handle cp_handle = -1;` |
| 2 | config 構造体メンバ設定 (CtrlPointStart 関数 Call 前) | `/* config 構造体メンバ設定 */`<br>`memset(&config, 0, sizeof(tUpnp_Conf));`<br>`memcpy(config.Ipaddress,ip_address,sizeof(config.Ipaddress));`<br>`config.PortNo = port;`<br>`config.UseWebserver = DUPNP_OFF;  // 既製 Web サーバ使用` |

| | | |
|---|---|---|
| | | config.TcpPortNo = TCP_PORTNO; // UPnP で使用する TCP ポート番号<br>config.UdpPortNo = UDP_PORTNO; // UPnP で使用する UDP ポート番号<br>config.UpnpTaskPri = UPNP_TASK_PRI; // UPnP タスクプライオリティ<br>config.ClsTimAcp = CLS_TIM_ACP; // TCP Connection クローズ待ちタイマー<br>config.IPTTL = DEF_IP_TTL; // SSDP にて使用する UDP TTL=4<br>config.UpnpMplSize = UPNP_BUFSIZ; // UPnP メモリプールサイズ設定 128KByte |
| 3 | ﾕｰｻﾞｱﾌﾟﾘｹｰｼｮﾝ内部データの初期化<br>(CtrlPointStart 関数 Call 前) | // 本 APL の初期化<br>SampleUtil_Initialize();<br><br>☆SampleUtil_Initialize 関数の詳細は実際の関数実装を確認して下さい。 |
| 4 | **UpnpInit** 関数によるミドルウェアの初期化<br>(CtrlPointStart 関数 Call 前) | ret = **UpnpInit**( &config );<br>if(ret != UPNP_E_SUCCESS )<br>{<br>//Upnp 初期化に失敗したなら<br>UpnpFinish();<br>return ret;<br>} |
| 5 | **UpnpRegisterClient** 関数によるコントロールポイントの登録 | ret = UpnpRegisterClient(CtrlPointCallbackEventHandler, &cp_handle, &cp_handle);<br>// ↑callback 関数 ↑cookie ↑ハンドル<br><br>if(ret != UPNP_E_SUCCESS){<br>UpnpFinish(); //UPnP 終了<br>return ret;<br>} |

### 停止関数：CtrlPointStop

停止処理は、コントロールポイント終了時に一度だけ Call するように実装します。
IP アドレスの変更など、ネットワーク状態が変更になった場合や、別なデバイスを起動したい場合などは、一旦本関数でデバイスを終了し、再度デバイスを起動します。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **UpnpUnRegisterClient** 関数によるコントロールポイントの削除 | UpnpUnRegisterClient(cp_handle); |
| 2 | **UpnpFinish** 関数による、ミドルウェアの終了<br>(CtrlPointStop 関数 Call 後) | UpnpFinish( ); //Upnp 終了 |
| 3 | ﾕｰｻﾞｱﾌﾟﾘｹｰｼｮﾝ内部データの開放 | SampleUtil_Finish( ); //サンプル終了<br>☆SampleUtil_Finish 関数の詳細は実際の関数実装を確認して下さい。 |

## 03) M-SEARCH 送信処理の実装 Part1－送信関数処理

### 送信関数：UpnpSearchAsync

関数 Call の度に M-SEARCH が送信されます。引数 SEARCH ターゲットを変更することができます。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **UpnpSearchAsync**<br>M-SEARCH 送信 | ret = UpnpSearchAsync(cp_handle, timeout, "upnp:rootdevice", NULL);<br>//ﾊﾝﾄﾞﾙ //ﾀｲﾑｱｳﾄ値 //ｻｰﾁﾀｰｹﾞｯﾄ |

## 04) M-SEARCH 送信処理の実装 Part2－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

M-SEARCH 後、応答があった場合または受信タイムアウトとなった場合、UpnpRegisterClient 関数にて登録した関数（サンプルでは CtrlPointCallbackEventHandler）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。また、本コールバック関数で、UPnPDownloadXmlDoc 関数にて、ディ

スクリプションドキュメントの取得を行い、デバイス情報の収集を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType = UPNP_DISCOVERY_SEARCH_RESULT**<br>M-SEARCH 応答受信 | case UPNP_DISCOVERY_SEARCH_RESULT:<br>{<br>struct Upnp_Discovery *d_event = //デバイス応答情報取得<br>( struct Upnp_Discovery * )Event;<br><br>//ここで応答受信時の処理を記述します<br>//(例)デバイスディスクリプションドキュメントの取得<br>IXML_Document *DescDoc = NULL;//と宣言しているものとする<br>//XML 取得<br>ret = **UpnpDownloadXmlDoc**(d_event ->Location,&DescDoc);<br>//応答情報の LocationURL<br>break;<br>} |
| 2 | **EventType = UPNP_DISCOVERY_SEARCH_RESULT**<br>M-SEARCH 受信タイムアウト | case UPNP_DISCOVERY_SEARCH_TIMEOUT:<br>//ここでタイムアウト時の処理を記述します<br>break; |

## 05) SUBSCRIBE 送信処理の実装 Part1－送信関数処理

### 送信関数：UpnpSubscribeAsync

引数で指定する EventURL に対して SUBSCRIBE します。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **UpnpSubscribeAsync**<br>SUBSCRIBE 送信 | ret = UpnpSubscribeAsync(cp_handle,SvcTbl[sn].EventURL,<br>//ﾊﾝﾄﾞﾙ //EventURL<br>timeout,CtrlPointCallbackEventHandler,NULL);<br>//更新ﾀｲﾑ値 //ｺｰﾙﾊﾞｯｸ関数 |

## 06) SUBSCRIBE 送信処理の実装 Part2－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

SUBSCRIBE 送信後、応答があった場合、UpnpSubscribeAsync 関数にて登録した関数（※注※サンプルでは、UpnpRegisterClient 関数と同じ CtrlPointCallbackEventHandler を登録していますが、専用のコールバック関数を登録することも可能です。）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType = UPNP_EVENT_SUBSCRIBE_COMPLETE**<br>SUBSCRIBE 応答受信 | case UPNP_EVENT_SUBSCRIBE_COMPLETE:<br>{<br>struct Upnp_Event_Subscribe *es_event = //応答情報取得<br>( struct Upnp_Event_Subscribe * )Event;<br>//ここで応答受信時の処理を記述します<br>break;<br>} |

## 07) UNSUBSCRIBE 送信処理の実装 Part1－送信関数処理

### 送信関数：UpnpUnSubscribeAsync

登録 SID のサービスに対して UNSUBSCRIBE します。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **UpnpUnSubscribeAsync**<br>UNSUBSCRIBE 送信 | ret = UpnpUnSubscribeAsync(cp_handle, SvcTbl[sn].SID,<br>//ﾊﾝﾄﾞﾙ //SID |

| | | |
|---|---|---|
| | | CtrlPointCallbackEventHandler,NULL);<br>//ｺｰﾙﾊﾞｯｸ関数 |

## 08) UNSUBSCRIBE 送信処理の実装 Part2－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

UNSUBSCRIBE 送信後、応答があった場合、UpnpUnSubscribeAsync 関数にて登録した関数（※注※サンプルでは、UpnpRegisterClient 関数と同じ CtrlPointCallbackEventHandler を登録していますが、専用のコールバック関数を登録することも可能です。）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType =**<br>**UPNP_EVENT_UNSUBSCRIBE_COMPLETE**<br>UNSUBSCRIBE 応答受信 | case UPNP_EVENT_UNSUBSCRIBE_COMPLETE:<br>{<br>struct Upnp_Event_Subscribe *es_event = //応答情報取得<br>( struct Upnp_Event_Subscribe * )Event;<br>//ここで応答受信時の処理を記述します<br>break;<br>} |

## 09) ACTION 送信処理の実装 Part1－送信関数処理

### 送信関数：UpnpSendActionAsync

SOAP XML を生成し送信します。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **UpnpAddToAction**<br>ACTION SOAP XML 生成 | ret = UpnpAddToAction(&actionNode, actionname, psvctype,<br>//SOAP XML //ｱｸｼｮﾝ名 //ｻｰﾋﾞｽﾀｲﾌﾟ<br>param_name[param],param_val[param] )<br>//変数名 //変数値 |
| 2 | **UpnpSendActionAsync**<br>ACTION SOAP XML 送信 | ret = UpnpSendActionAsync(cp_handle,<br>//ﾊﾝﾄﾞﾙ<br>SvcTbl[svcno].ControlURL,psvctype,NULL,<br>//ControlURL //ｻｰﾋﾞｽﾀｲﾌﾟ<br>actionNode,CtrlPointCallbackEventHandler,<br>//SOAP XML //ｺｰﾙﾊﾞｯｸ関数<br>NULL ); |

## 10) ACTION 送信処理の実装 Part2－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

ACTION 送信後、応答があった場合、UpnpSendActionAsync 関数にて登録した関数（※注※サンプルでは、UpnpRegisterClient 関数と同じ CtrlPointCallbackEventHandler を登録していますが、専用のコールバック関数を登録することも可能です。）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType =**<br>**UPNP_CONTROL_ACTION_COMPLETE**<br>ACTION 応答受信 | case UPNP_CONTROL_ACTION_COMPLETE:<br>{<br>struct Upnp_Action_Complete *a_event = //応答情報取得<br>( struct Upnp_Action_Complete * )Event;<br>//ここで応答受信時の処理を記述します<br>break;<br>} |

## 11) QUERY 送信処理の実装 Part1－送信関数処理

### 送信関数：UpnpGetServiceVarStatusAsync

QUERY を送信します。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **UpnpGetServiceVarStatusAsync**<br>QUERY 送信 | ret = UpnpGetServiceVarStatusAsync(cp_handle,<br>//ﾊﾝﾄﾞﾙ<br>SvcTbl[svcno].ControlURL,<br>//ControlURL<br>VarName,<br>//変数名<br>CtrlPointCallbackEventHandler,<br>//ｺｰﾙﾊﾞｯｸ関数<br>NULL ); |

## 12) QUERY 送信処理の実装 Part2－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

QUERY 送信後、応答があった場合、UpnpGetServiceVarStatusAsync 関数にて登録した関数（※注※サンプルでは、UpnpRegisterClient 関数と同じ CtrlPointCallbackEventHandler を登録していますが、専用のコールバック関数を登録することも可能です。）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType = UPNP_CONTROL_GET_VAR_COMPLETE**<br>QUERY 応答受信 | case UPNP_CONTROL_GET_VAR_COMPLETE:<br>{<br>struct Upnp_State_Var_Complete *sv_event = //応答情報取得<br>( struct Upnp_State_Var_Complete * )Event;<br>//ここで応答受信時の処理を記述します<br>break;<br>} |

## 13) SSDP NOTIFY 受信処理の実装－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

SSDP NOTIFY を受信した場合、UpnpRegisterClient 関数にて登録した関数（サンプルでは CtrlPointCallbackEventHandler）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType = UPNP_DISCOVERY_ADVERTISEMENT_ALIVE**<br>SSDP NOTIFY 受信 | case UPNP_DISCOVERY_ADVERTISEMENT_ALIVE:<br>{<br>struct Upnp_Discovery *d_event = //受信情報取得<br>( struct Upnp_Discovery * )Event;<br>//ここで受信時の処理を記述します<br>break;<br>} |

## 14) SSDP BYEBYE 受信処理の実装－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

SSDP BYEBYE を受信した場合、UpnpRegisterClient 関数にて登録した関数（サンプルでは CtrlPointCallbackEventHandler）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType = UPNP_DISCOVERY_ADVERTISEMENT_BYEBYE**<br>SSDP BYEBYE 受信 | case UPNP_DISCOVERY_ADVERTISEMENT_BYEBYE:<br>{<br>struct Upnp_Discovery *d_event = //受信情報取得<br>( struct Upnp_Discovery * )Event;<br>//ここで受信時の処理を記述します<br>break;<br>} |

## 15) GENA NOTIFY 受信処理の実装－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

SUBSCRIBE 登録済み UPnP デバイスの状態変化時に送信される GENA NOTIFY を受信した場合、UpnpRegisterClient 関数にて登録した関数（サンプルでは CtrlPointCallbackEventHandler）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType = UPNP_EVENT_RECEIVED**<br>GENA NOTIFY 受信 | case UPNP_EVENT_RECEIVED:<br>{<br>struct Upnp_Event *e_event = //受信情報取得<br>( struct Upnp_Event * )Event;<br>//ここで受信時の処理を記述します<br>break;<br>} |

## 16) SUBSCRIBE RENEWAL 失敗時処理の実装－コールバック関数処理

### コールバック関数：CtrlPointCallbackEventHandler

SUBSCRIBE RENEWAL に失敗した場合、UpnpRegisterClient 関数にて登録した関数（サンプルでは CtrlPointCallbackEventHandler）が Call されます。関数内で引数 Upnp_EventType EventType の値に応じた処理を行います。

| Step | 実装 | ソースコード記載 |
|---|---|---|
| 1 | **EventType = UPNP_EVENT_AUTORENEWAL_FAILED**<br>SUBSCRIBE RENEWAL 失敗 | case UPNP_EVENT_AUTORENEWAL_FAILED:<br>{<br>struct Upnp_Event_Subscribe *es_event = //受信情報取得<br>( struct Upnp_Event_Subscribe * )Event;<br>//ここで受信時の処理を記述します<br>break;<br>} |