RL78 BLE Dongle
仮想 UART 通信
チュートリアルガイド

Rev　: 1.0
TS-TUM03626
2015/12/04

目次

1　概要

テセラ・テクノロジー製 RL78 BLE Dongle と Android 端末を用いて、RL78 BLE Dongle と Android 端末間で、Bluetooth Low Energy の通信を利用した仮想 UART 通信により文字列の送受信を行うデモンストレーションの実施方法および実装方法を示します。

A)　動作確認デバイス
RL78 BLE Dongle（RL78/G1D）

B)　略語および略称の説明

略語/略称	フルスペル	備考
BLE	Bluetooth Low Energy	
SCP	Sample Custom Profile	仮想 UART 通信に使用する独自プロファイル
GUI アプリ	rBLE_UART_GUI	仮想 UART 通信に使用する PC 上で動作する GUI アプリケーション
Android アプリ	G1D uart	仮想 UART 通信に仕様する Android 上で動作する GUI アプリケーション

表 1-1 略語および略書の説明

2　開発環境

仮想 UART 通信ソフトウェアの Windows 用 GUI アプリケーション「rBLE_UART_GUI」の開発環境を以下に示します。

項目	説明
統合開発環境	Visual Studio 2013 Professional Update 5 ※Visual Studio Community 2013 with Update 5 でもビルド可
言語	Visual C++ 2013　(MFC 使用)
ライブラリ	ルネサス エレクトロニクス製 Bluetooth Low Energy プロトコルスタック V1.10

表 2-1 rBLE_UART_GUI の開発環境

3　動作環境

仮想 UART 通信ソフトウェアの Windows 用 GUI アプリケーション「rBLE_UART_GUI」の動作環境を以下に示します。

項目	説明
OS	Windows 7 Professional 日本語 64bit 版
ランタイム	Visual Studio 2013 の Visual C++ 再頒布可能パッケージ (32bit 版)

表 3-1 rBLE_UART_GUI の動作環境

4　参考資料

1. ネサス エレクトロニクス
Bluetooth Low Energy プロトコルスタック ユーザーズマニュアル
R01UW0095JJ0117

5　デモンストレーション説明書

5.1　デモンストレーション概要

本デモは、RL78 BLE Dongle(以下、Dongle と表記)および rBLE_UART_GUI(以下、GUI と表記)と Android 端末(以下、Android と表記)を用いて、Dongle と Android 間で、Bluetooth Low Energy の通信を利用した仮想 UART 通信により文字列の送受信を行います。

Android は Master として動作し、Dongle および GUI は Slave として動作します。

デモ構成は以下の通りです。

Dongle は PC の USB 端子に接続し、GUI から文字列の送受信を行います。Android は Android アプリケーション「G1D uart」を使用して文字列の送受信を行います。

仮想 UART 通信とは、Dongle および GUI と Android に実装されている Bluetooth Low Energy の独自プロファイルである Sample Custom Profile(SCP)を使用し、両者間 Bluetooth Low Energy の通信を仮想的に UART の通信に見立てた通信のことです。

SCP についての詳細は、7.3 Sample Custom Profile API を参照してください。

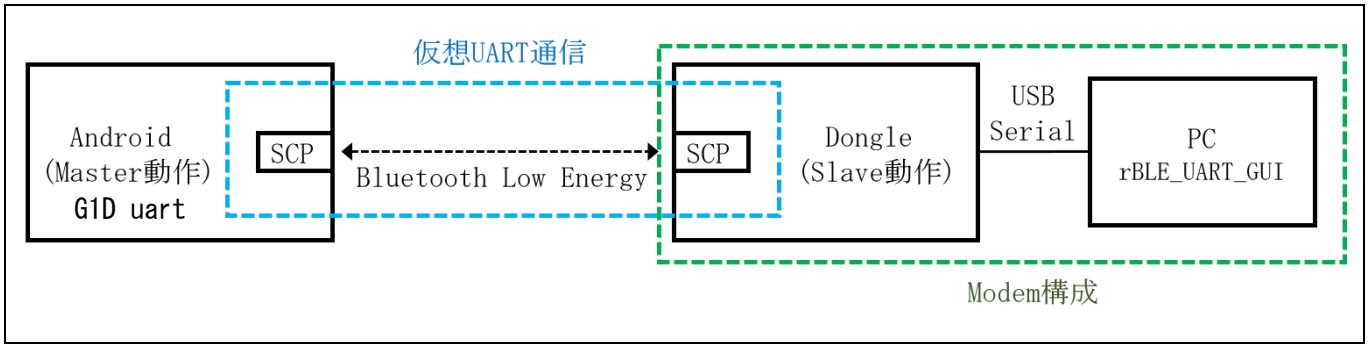


図 5-1 システムブロック図

5.2　ソフトウェア構成図

RL78 BLE Dongle と rBLE_UART_GUI は Modem 構成となっており、APP_MCU と BLE_MCU の 2 つのチップで動作します。

Dongle と GUI のソフトウェア構成図を以下に示します。

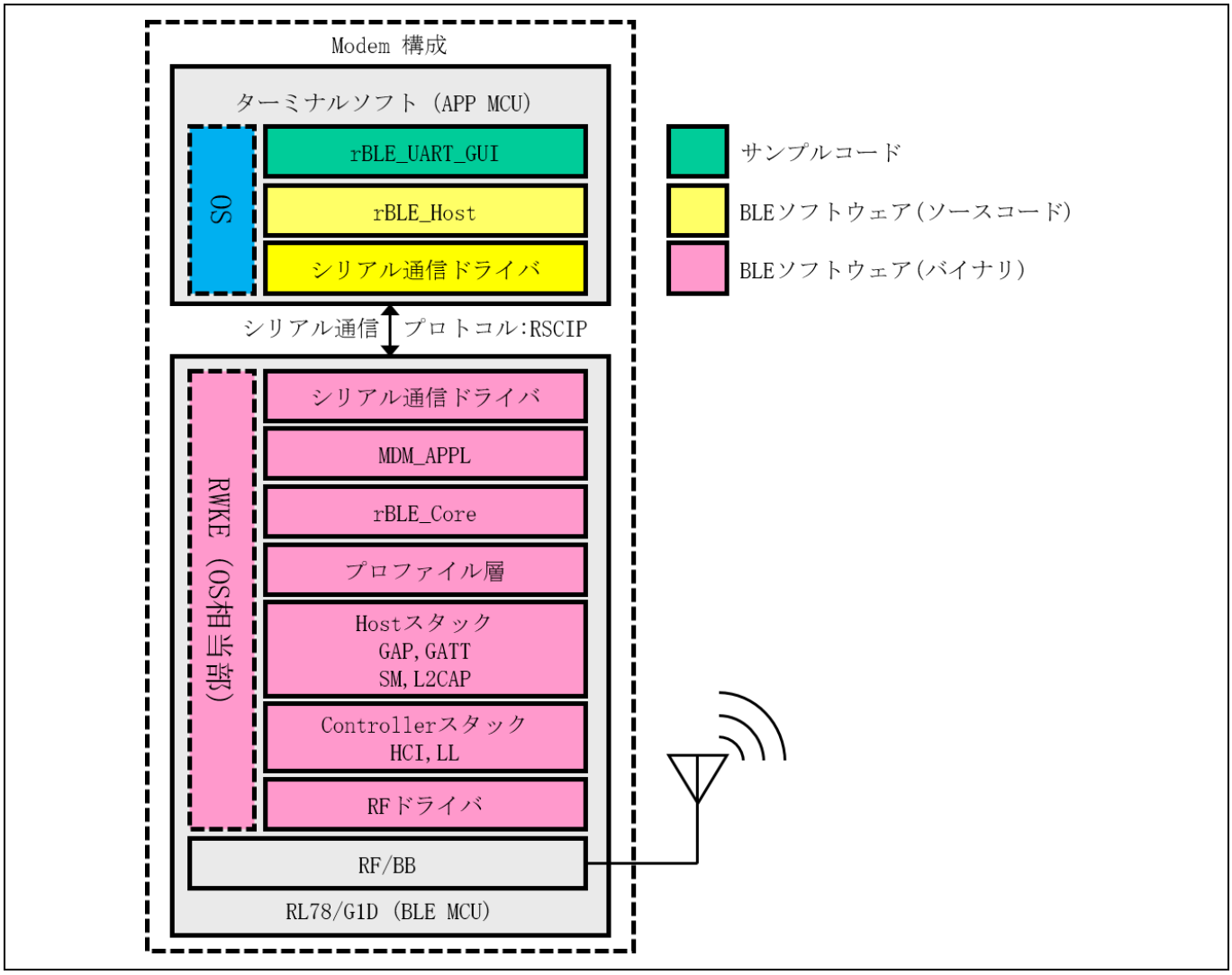


図 5-2 ソフトウェア構成図

5.3　デモセット部品一覧

デモンストレーションを実施するために必要な部品は以下の通りです。

部品名	数量	参考情報
RL78 BLE Dongle	1	テセラ・テクノロジー製
Windows7 以降を搭載したパソコン	1	「Visual Studio 2013 の Visual C++ 再頒布可能パッケージ」がインストールされた状態
rBLE_UART_GUI	1	GUI サンプルアプリケーション
BLE に対応した Android (Ver4.4.2 以降) (HUAWEI Ascend P7)	1	※4.4.2 以降には対応しておりますが、 すべての端末・バージョンでの動作は保証していません。
G1D uart	1	Android アプリケーション

表 5-1 デモセット部品一覧表

5.4　セットアップ

5.4.1　rBLE_UART_GUI のセットアップ

PC に接続した Dongle の制御に「rBLE_UART_GUI」を使用します。
「rBLE_UART_GUI」は製品に添付されていた文書記載の URL からダウンロードして下さい。
「rBLE_UART_GUI」を実行するには「Visual Studio 2013 の Visual C++ 再頒布可能パッケージ」(以下：ランタイムと表記)が必要です。
ランタイムを https://www.microsoft.com/ja-jp/download/details.aspx?id=40784 より入手してインストールします。
インストール方法は上記 URL の「インストール方法」を参照してください。

5.4.2　Android のセットアップ

① Android の[設定] - [セキュリティ] - 提供元不明アプリのインストールを許可にします。

② 下記の QR コードを読み取って「G1Duart_1.0.0.0.apk」 ファイルをダウンロードしてください。

QR コードが読み取れない場合は、下記よりダウンロードしてください。

http://www.tessera.co.jp/Download/Android/G1Duart_1.0.0.0.apk

③ ダウンロードした「G1Duart_1.0.0.0.apk」ファイルをタップすると[インストール]ボタンが表示されるので、タップしてください。

④ アプリケーションのインストールが終わると[アプリケーションをインストールしました]と表示されます。

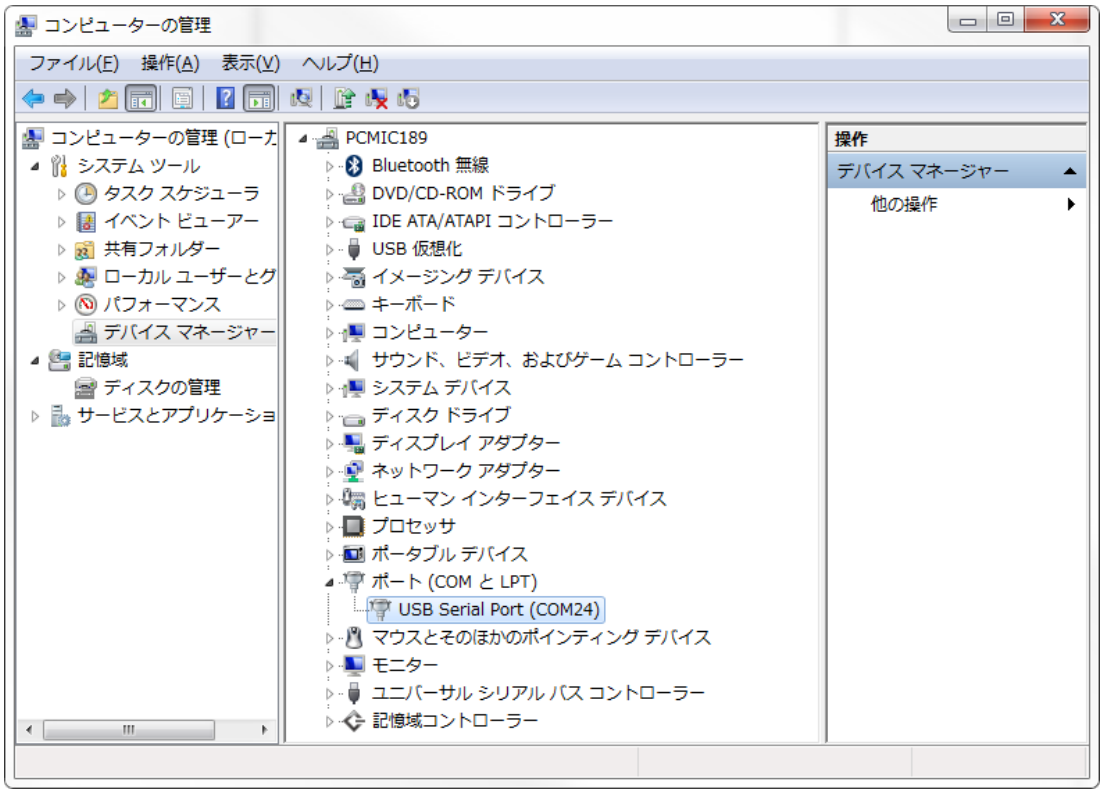
⑤ アプリケーションメニューに「G1Duart」のアイコンが登録されるので、アイコンをタップして「G1Duart」を起動します。

5.5　操作方法

Dongle および rBLE_UART_GUI と Android 間で通信を行う場合の操作方法を記載します。

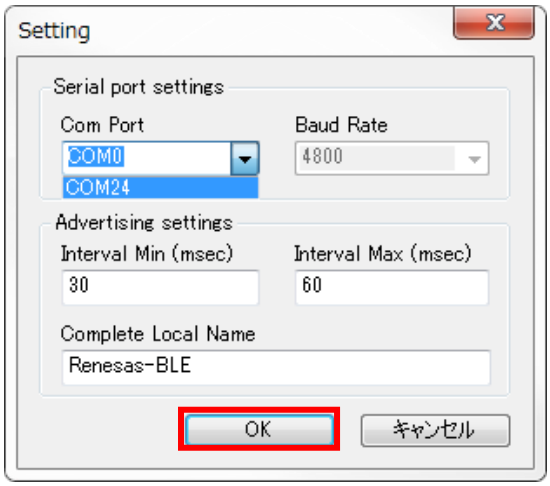
5.5.1　Dongle と Android 間での仮想 UART 通信を行う

① Dongle を PC の USB ポートに差し込み、PC が Dongle を認識したことをデバイスマネージャで確認します。



② rBLE_UART_GUI を起動します。

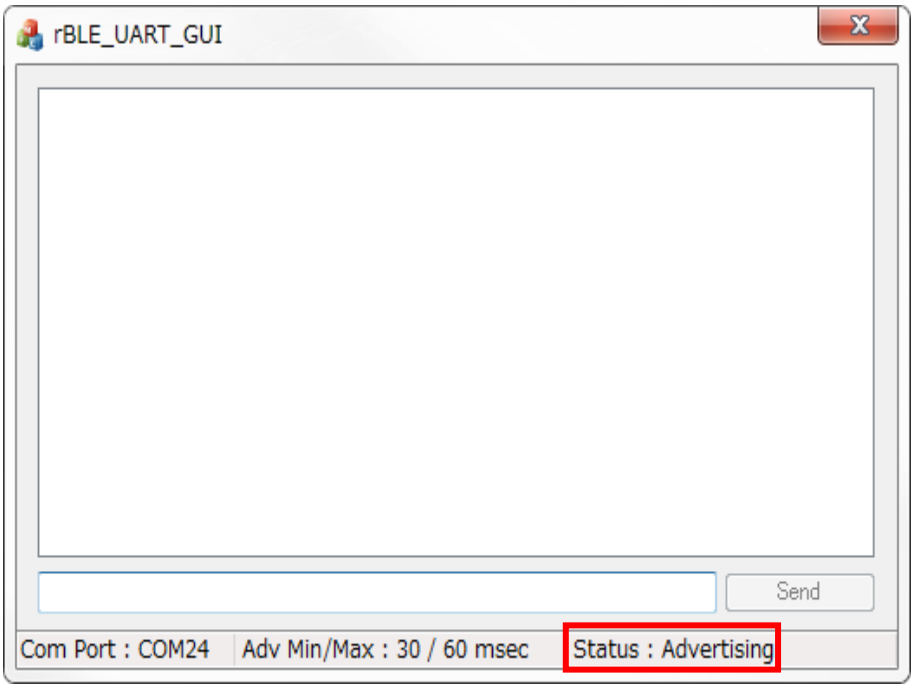
設定画面で、Dongle が接続されているポート名と Advertising 間隔および Complete Local Name を設定して OK ボタンを押下します。



項目名	デフォルト	説明
Com Port	COM0	Dongle が接続されている COM ポート名を選択します。
Baud Rate	4800	Dongle と GUI の通信速度を設定します。 (4800bps 固定)
Interval Min (msec)	30	Advertising 間隔の最小値を設定します。 (範囲：20～10240)
Interval Max (msec)	60	Advertising 間隔の最大値を設定します。 (範囲：20～10240)
Complete Local Name	Relesas-BLE	デバイス名を設定します。 (最大 26 バイト)

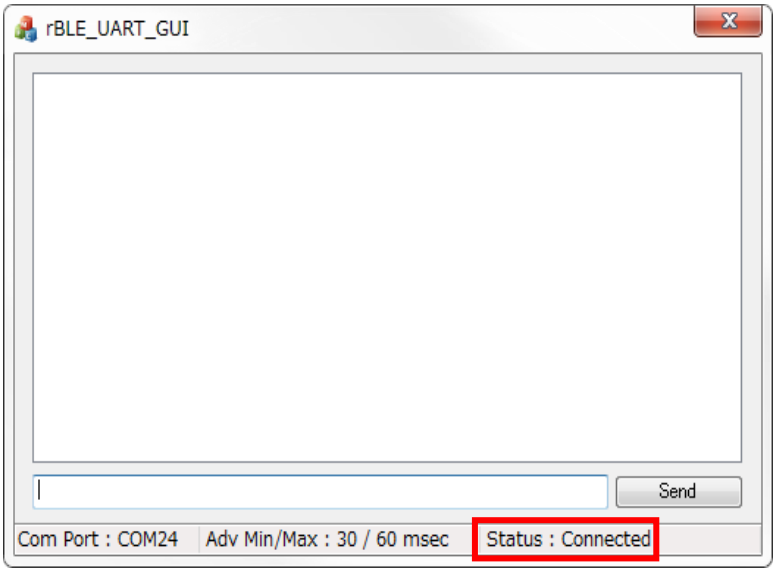
表 5-2 設定画面での設定項目

③ rBLE_UART_GUIのメイン画面が表示され、StatusがAdvertising状態になっていることを確認します。

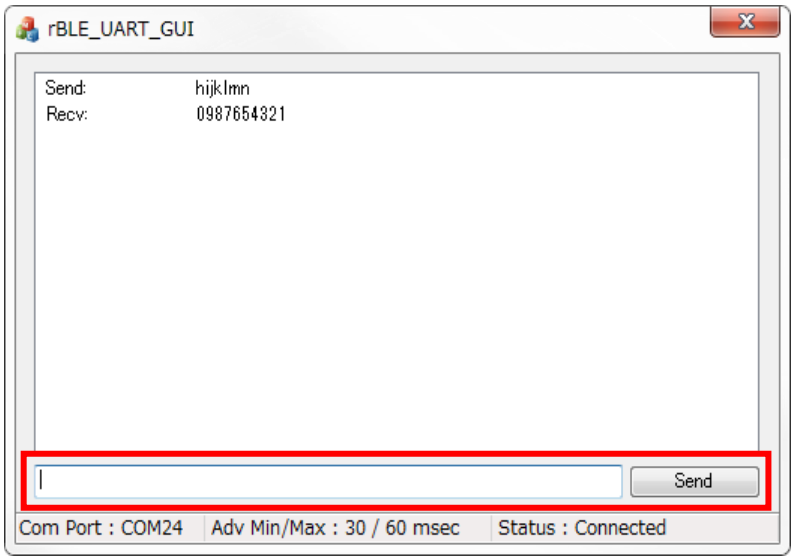


④ Android アプリケーション「G1D uart」を起動し、「DEVICE」ボタンをタップします。先程起動した Slave が発見され Complete Local Name が表示されるので、タップして接続を行います。

rBLE_UART_GUI のステータスが Connected になることを確認します。



⑤ Slave の rBLE_UART_GUI に任意の文字列を入力し、Send ボタンを押します。「G1D uart」に文字列が表示されることを確認します。「G1D uart」に任意の文字列を入力し、「Send」ボタンをタップします。Slave の rBLE_UART_GUI に文字列が表示されることを確認します。Dongle と Android 間で文字列の送受信が行えることを確認してください。



Master (Android)	Slave (Dongle)
送信した文字列が青色文字で、受信した文字列が赤色文字で表示されます。	送信した文字列が Send:として、受信した文字列が Recv:として表示されます。

6　プロファイル構成

6.1　プロファイル一覧

本デモを実現するにあたり、BLE の独自プロファイルを実装し、使用しています。

BLE のプロファイルには Client と Server の 2 つの役割がありますが、本デモでは Master を Client、Slave を Server として動作させます。各動作とそれに使用されているプロファイルを表 6-1 に示します。

動作	プロファイル名	備考
仮想 UART 通信動作	Sample Custom Profile (SCP)	Android は Master 動作のみ対応 Dongle は Slave 動作のみ対応

表 6-1 各動作とプロファイル

本デモに使用しているプロファイルの仕様を以下に示します。

6.1.1　Sample Custom Profile

Service 名	Sample Custom Service
UUID	0x80000000 00000000 00000000 00000001

表 6-2 Srvice 名

Characteristic 名	Notify Characteristic		
Properties	Notify		
UUID	0x80000000 00000000 00000000 00000002		
Member	Uint8_t	data[]	Client から Server への送信文字列(1～20byte)

表 6-3 Notify Characteristic

Characteristic 名	Write Characteristic		
Properties	Write		
UUID	0x80000000 00000000 00000000 00000003		
Member	Uint8_t	data[]	Client から Server への送信文字列(1～20byte)
Response	RBLE_ATT_ERR_NO_ERROR(0x00) or RBLE_ATT_ERR_APP_ERROR(0x80)		

表 6-4 Write Characteristic

7　ソフトウェア構成

7.1　ソフトウェア概要

Dongle の動作は rBLE_UART_GUI から起動することにより、Server 動作を行います。Dongle と rBLE_UART_GUI の構成は 5.2 ソフトウェア構成図を参照してください。

7.1.1　仕様概要

Dongle および rBLE_UART_GUI を動作させると Advertising が行われます。接続が確立されると「仮想 UART 通信動作」に移行します。

「仮想 UART 通信動作」の内容を以下に示します。

- 仮想 UART 通信動作
  - Advertising を行い、Master との接続を待ちます。
  - Master と接続確立後、仮想 UART 通信動作に移行します。
  - rBLE_UART_GUI から USB Serial 経由で受信したデータを、BLE により Master へ送信します。
  - Master から BLE により受信したデータを、rBLE_UART_GUI へ USB Serial 経由で送信します。

7.1.2　ソフトウェア動作

Modem 構成で動作するソフトウェアについて説明します。

① GUI 起動直後設定画面の「Setting」ダイアログを表示します。
② 「Setting」ダイアログで、OK ボタンを押下するとメイン画面、キャンセルボタンを押下すると終了します。
③ メイン画面が表示されると COM ポートをオープンして BLE の初期化を行い、Advertising を開始して Android からの接続を待ちます。
④ Android と接続確立後「仮想 UART 通信動作」を行います。
⑤ Android との接続が切断されると Advertising を再開します。

Modem 構成で動作するソフトウェアのフローチャートを図 7-1 に示します。

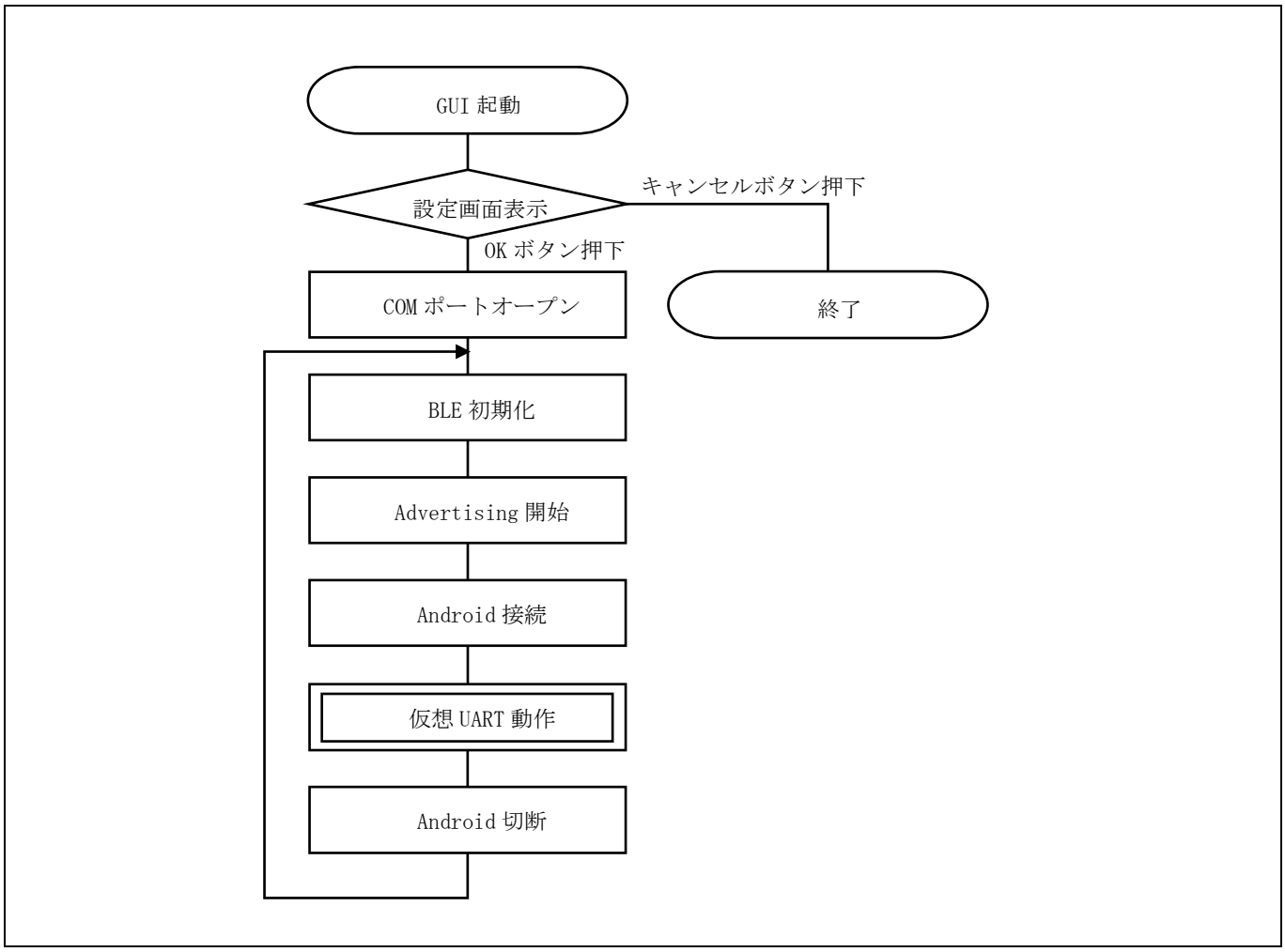


図 7-1 Modem 構成で動作するソフトウェアのフローチャート

7.1.3　仮想 UART 通信動作

「仮想 UART 通信動作」について説明します。「仮想 UART 通信動作」では、Master と Slave 間で仮想 UART 通信を行います。Dongle は PC と接続されていて、rBLE_UART_GUI から文字列を入力し、Send ボタンを押下することで、Android にその文字列を表示させます。1 度に送信できる文字列は最大で 20 文字です。

- 送信動作
  ①　rBLE_UART_GUI から送信する文字列を入力します。
  ②　入力した文字列を BLE により送信します。

- 受信動作
  ①　BLE により受信した文字列を rBLE_UART_GUI へ出力します。

「仮想 UART 通信動作」のフローチャートを図 7-2、図 7-3 に示します。

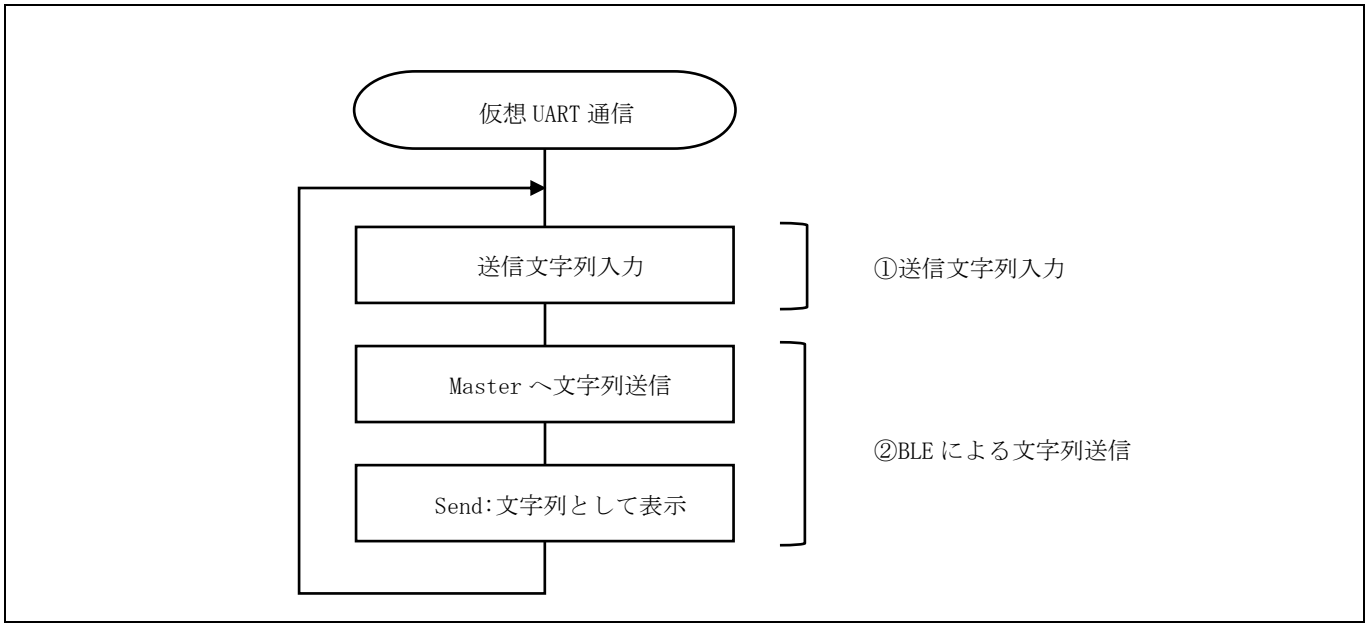


図 7-2 仮想 UART 通信動作フローチャート(送信動作)

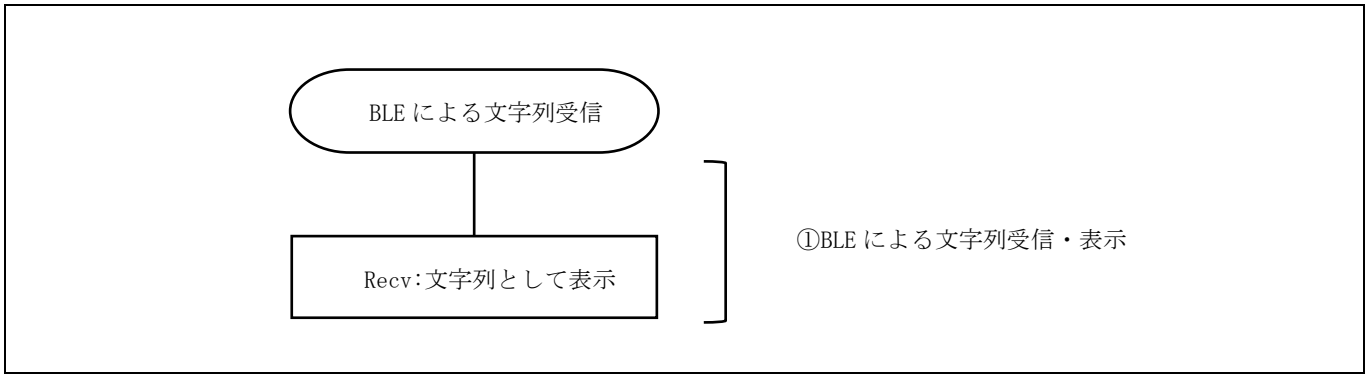


図 7-3 仮想 UART 通信動作フローチャート(受信動作)

7.2　ソースコード解説

本デモのソースコードについて解説します。
本デモに関連する MCU APP のファイルについて、その動作内容を表 7-1 に示します。
次節より、ソースコードを示しながら動作を説明します。

ファイル	動作内容
rBLE_Core.c	BLE ソフトウェア用 API の C++ラッパー関数
SettingsDlg.cpp	設定ダイアログ
rBLE_UART_GUIDlg.cpp	仮想 UART 通信動作 APP
scps.c	Sample Custom Profile Server 用 API

表 7-1 MCU APP ファイルの動作内容

7.2.1　Advertising 開始

初期化後、設定ダイアログの設定内容に従って Advertising を開始します。

行数	rBLE_UART_GUIDlg.cpp
838	`// コネクションハンドルのクリア`
839	`m_uiConnectionHandle = ILLEGAL_CONNECTION_HANDLE;`
840	
841	`// 設定情報のパラメータを取得`
842	`CrBLE_Params::GetBroadCastEnableParam(`
843	`&prm,`
844	`theApp.m_pSettingInfo->m_settingInfo.uiAdvMin,`
845	`theApp.m_pSettingInfo->m_settingInfo.uiAdvMax,`
846	`theApp.m_pSettingInfo->m_settingInfo.szCompLocalName,`
847	`strlen(theApp.m_pSettingInfo->m_settingInfo.szCompLocalName));`
848	
849	`// ブロードキャスト開始`
850	`WP_RBLE_GAP_Broadcast_Enable(prm.disc_mode, prm.conn_mode, &prm.adv_info);`

7.2.2　接続/仮想 UART 通信同作用プロファイルの有効化

接続が確立されたら仮想 UART 通信動作用プロファイル(Server)の有効化を行います。

行数	rBLE_UART_GUIDlg.cpp
896	`// コネクションハンドルを保持`
897	`Con_Info_p = &event->param.conn_comp.connect_info;`
898	`m_uiConnectionHandle = Con_Info_p->conhdl;`
899	
900	`// SCPサーバーの有効化`
901	`/* Valid Connection Handle ? */`
902	`if (ILLEGAL_CONNECTION_HANDLE != m_uiConnectionHandle)`
903	`{`
904	`::SendMessage(CrBLE_UART_GUIDlg::m_hWnd, WM_APPEND_LOG_FILE, NULL, (LPARAM)&log);`
905	`log.log.Empty();`
906	
907	`::SendMessage(m_hWnd, WM_CLEAR_DATA_LOG, NULL, NULL);`
908	
909	`/* API Call */`
910	`/* SCPサーバー有効化 */`
911	`ret_api = WP_RBLE_SCP_Server_Enable(m_uiConnectionHandle, RBLE_APP_SCPS_CallBack);`
912	`::SendMessage(m_hWnd, WM_UPDATE_STATUS, NULL, (LPARAM)EStatus::STATUS_CONNECT);`
913	
914	`log.log.AppendFormat(_T("WP_RBLE_SCP_Server_Enable(conhdl=%d)=0x%04X"),`
915	`m_uiConnectionHandle,`
916	`ret_api);`
917	`}`

7.2.3　文字列の送信

Sample Custom Profile を有効化したら、文字列の送信を行います。ソースコードを以下に示します。任意の文字を入力し、Send ボタンを押下することで入力した文字列を対応デバイスに送信します。

行数	rBLE_UART_GUIDlg.cpp
320	`UpdateData(TRUE);`
321	
322	`uint8_t data[MAX_SEND_DATA_LEN];`
323	`uint8_t len = m_ctrlValSendData.GetLength();`
324	
325	`if (ILLEGAL_CONNECTION_HANDLE != m_uiConnectionHandle)`
326	`{`
327	`for (int i = 0; i < m_ctrlValSendData.GetLength(); i++)`
328	`{`
329	`data[i] = (uint8_t)m_ctrlValSendData[i];`
330	`}`
331	
332	`// データ送信`
333	`WP_RBLE_SCP_Server_Send_String(m_uiConnectionHandle, data, len);`
334	
335	`// Logデータ生成`
336	`datalog.direction = CLogList::EDataDirection::Send;`
337	`datalog.log = m_ctrlValSendData.GetBuffer();`
338	`SendMessage(WM_APPEND_DATA_LOG, NULL, (LPARAM)&datalog);`
339	
340	`log.category = ELogCategory::BleApi;`
341	`log.log.AppendFormat(_T("WP_RBLE_SCP_Server_Send_String(conhdl=%d, data='%s', len=%d)"),`
342	`m_uiConnectionHandle,`
343	`m_ctrlValSendData,`
344	`m_ctrlValSendData.GetLength());`
345	`::SendMessage(CrBLE_UART_GUIDlg::m_hWnd, WM_APPEND_LOG_FILE, NULL, (LPARAM)&log);`
346	
347	`m_ctrlValSendData.ReleaseBuffer();`
348	
349	`m_ctrlValSendData.Empty();`
350	`UpdateData(FALSE);`
351	`}`

7.2.4　文字列の受信

対向デバイスから文字列の受信を行います。文字列受信部分のソースコードを以下に示します。

行数	rble_sample_app.c
996	/* クライアントからデータを受信 */
997	case RBLE_SCP_EVENT_SERVER_CHG_CHAR_IND:
998	log.log.Append(_T(" RBLE_SCP_EVENT_SERVER_CHG_CHAR_IND"));
999	PRINTF("RBLE_SCP_EVENT_SERVER_CHG_CHAR_IND¥n");
1000	
1001	datalog.direction = CLogList::EDataDirection::Recv;
1002	
1003	for (int loop_cnt = 0; loop_cnt < event->param.write_char.data_len; loop_cnt++)
1004	{
1005	buff.AppendChar((char_t)event->param.write_char.data[loop_cnt]);
1006	}
1007	
1008	datalog.log = buff.GetBuffer();
1009	::SendMessage(m_hWnd, WM_APPEND_DATA_LOG, NULL, (LPARAM)&datalog);
1010	
1011	log.log.AppendFormat(_T(", '%s'"), buff);
1012	
1013	buff.ReleaseBuffer();
1014	break;

7.3 Sample Custom Profile API

Sample Custom Profile (SCP)の API について記載します。Sample Custom Profile を使用することで、仮想 UART 通信を行うことができます。

7.3.1 Definitions

Sample Custom Profile の API で使用される定義について記載します。

● SCP イベントタイプ列挙型宣言

```
enum RBLE_SCP_EVENT_TYPE_enum {
  RBLE_SCP_EVENT_SERVER_ENABLE_COMP          = 0x01,   // Server 有効設定完了イベント
  RBLE_SCP_EVENT_SERVER_DISABLE_COMP         = 0x02,   // Server 無効設定完了イベント
  RB LE_SCP_EVENT_SERVER_SEND_NOTIFY_COMP    = 0x03,   // Server 文字列送信完了イベント
  RBLE_SCP_EVENT_SERVER_CHG_CHAR_IND         = 0x04,   // Server 文字列受信イベント
  RBLE_SCP_EVENT_CLIENT_ENABLE_COMP          = 0x81,   // Client 有効設定完了イベント
  RBLE_SCP_EVENT_CLIENT_DISABLE_COMP         = 0x82,   // Client 無効設定完了イベント
  RBLE_SCP_EVENT_CLIENT_NOTIFY               = 0x83,   // Client 文字列受信イベント
  RBLE_SCP_EVENT_CLIENT_WRITE_CHAR_RESPONSE  = 0x84    // Client 文字列送信応答イベント
};
```

● SCP イベントタイプ型宣言

```
typedef uint8_t RBLE_SCP_EVENT_TYPE;
```

● SCP イベントタイプ型宣言

```
typedef uint8_t  RBLE_SCP_EVENT_TYPE;
```

● SCP Server イベントコールバック関数型宣言

```
typedef void ( *RBLE_SCPS_EVENT_HANDLER )( RBLE_SCPS_EVENT *event );
```

● SCP Client イベントコールバック関数型宣言

```
typedef void ( *RBLE_SCPC_EVENT_HANDLER )( RBLE_SCPC_EVENT *event );
```

● 仮想 UART 通信サービス内容構造体

```
typedef struct RBLE_SCS_CONTENT_t {
  uint16_t      shdl;                   // 仮想 UART 通信開始ハンドル
  uint16_t      ehdl;                   // 仮想 UART 通信終了ハンドル
  uint16_t      notify_char_hdl;        // Server 文字列特性ハンドル
  uint16_t      notify_val_hdl;         // Server 文字列特性値ハンドル
  uint8_t       notify_prop;            // Server 文字列特性プロパティ
  uint8_t       reserved1;              // 予約
  uint16_t      write_char_hdl;         // Client 文字列特性ハンドル
  uint16_t      write_val_hdl;          // Client 文字列特性値ハンドル
  uint8_t       write_prop;             // Client 文字列特性プロパティ
  uint8_t       reserved2;              // 予約
}RBLE_SCS_CONTENT;
```

● 仮想 UART 通信サービス定義

```
#define RBLE_SCP_WRITE_CHAR_MAX    (0x14)    // 最大送信文字数
```

● SCP Server イベントパラメータ構造体

```
typedef struct RBLE_SCPS_EVENT_t {
  RBLE_SCP_EVENT_TYPE              type;                                  // SCP イベントタイプ
  uint8_t                          reserved;                              // 予約
  union Event_Scps_Parameter_u {
    RBLE_STATUS                    status;                                // ステータス

    // Server 有効設定完了イベント
    struct RBLE_SCP_Server_Enable_t{
      uint16_t                     conhdl;                                // コネクションハンドル
      RBLE_STATUS                  status;                                // ステータス
      uint8_t                      reserved;                              // 予約
    }server_enable;

    // Server 無効設定完了イベント
    struct RBLE_SCP_Server_Disable_t {
       uint16_t                    conhdl;                                // コネクションハンドル
       RBLE_STATUS                 status;                                // ステータス
       uint8_t                     reserved;                              // 予約
    }server_disable;

    // Server 文字列送信完了イベント
    struct RBLE_SCP_Server_Send_Notify_t {
       uint16_t                    conhdl;                                // コネクションハンドル
       RBLE_STATUS                 status;                                // ステータス
       uint8_t                     reserved;                              // 予約
    }send_notify;

    // Server 文字列受信イベント
    struct RBLE_SCP_Server_Write_Chara_Ind_t{
       uint16_t                    conhdl;                                // コネクションハンドル
       uint8_t                     data_len;                              // 文字数
       uint8_t                     reserved;                              // 予約
       uint8_t                     data[RBLE_SCP_WRITE_CHAR_MAX];         // 文字列
    }write_char;
  }param;
}RBLE_SCPS_EVENT;
```

● SCP Client イベントパラメータ構造体

```
typedef struct RBLE_SCPC_EVENT_t {
  RBLE_SCP_EVENT_TYPE              type;                                  // SCP イベントタイプ
  uint8_t                          reserved;                              // 予約
  union Event_Scpc_Parameter_u {
       RBLE_STATUS                 status;                                // ステータス

       // Client 有効設定完了イベント
       struct RBLE_SCP_Client_Enable_t{
          uint16_t                 conhdl;                                // コネクションハンドル
          RBLE_STATUS              status;                                // ステータス
          uint8_t                  reserved;                              // 予約
          RBLE_SCS_CONTENT         scs;                                   // 仮想 UART 通信サービス情報
       }client_enable;

       // Client 無効設定完了イベント
       struct RBLE_SCP_Client_Disable_t {
         uint16_t                  conhdl;                                // コネクションハンドル
         RBLE_STATUS               status;                                // ステータス
         uint8_t                   reserved;                              // 予約
       }client_disable;

       // Client 文字列受信イベント
       struct RBLE_SCP_Client_Notify_Ind_t {
         uint16_t                  conhdl;                                // コネクションハンドル
         uint8_t                   data_len;                              // 文字数
         uint8_t                   reserved;                              // 予約
         uint8_t                   data[RBLE_SCP_WRITE_CHAR_MAX];         // 文字列
       }notify;

       // Client 文字列送信応答イベント
       struct RBLE_SCP_Client_Write_Char_Response_t {
         uint16_t                  conhdl;                                // コネクションハンドル
         uint8_t                   att_code;                              // ステータス
         uint8_t                   reserved;                              // 予約
       }wr_char_resp;
  }param;
}RBLE_SCPC_EVENT;
```

7.3.2 Functions

以下に、SCP 機能で定義されている API 関数を表 7-2 にまとめ、その API 関数の詳細について説明します。

API 名	概要
RBLE_SCP_Server_Enable	Server Role を有効にする
RBLE_SCP_Server_Disable	Server Role を無効にする
RBLE_SCP_Server_Send_String	文字列を送信する

表 7-2 SCP 機能 API 関数一覧

7.3.2.1 RBLE_SCP_Server_Enable

RBLE_STATUS RBLE_SCP_Server_Enable(uint16_t conhdl, RBLE_SCPS_EVENT_HANDLER call_back)		
このファンクションは、SCP 機能の Server Role を有効にします。 結果は Server Role 有効設定完了イベント RBLE_SCP_EVENT_SERVER_ENABLE_COMP で通知されます。		
Parameters:		
	conhdl	コネクションハンドル
	call_back	SCP のイベントを通知するコールバックファンクションを指定
Return:		
	RBLE_OK	正常終了
	RBLE_PARAM_ERR	パラメータ異常
	RBLE_STATUS_ERROR	状態エラー

7.3.2.2 RBLE_SCP_Server_Disable

RBLE_STATUS RBLE_SCP_Server_Disable(uint16_t conhdl)		
このファンクションは、SCP 機能の Server Role を無効にします。 結果は Server Role 無効設定完了イベント RBLE_SCP_EVENT_SERVER_DISABLE_COMP で通知されます。		
Parameters:		
	conhdl	コネクションハンドル
Return:		
	RBLE_OK	正常終了
	RBLE_PARAM_ERR	パラメータ異常
	RBLE_STATUS_ERROR	状態エラー

7.3.2.3 RBLE_SCP_Server_Send_String

RBLE_STATUS RBLE_SCP_Server_Send_String(uint16_t conhdl, uint8_t *write_value, uint8_t write_length)		
このファンクションは、Client に対して文字列を送信します。最大送信可能文字数は 20 です。 結果は文字列送信完了イベント RBLE_SCP_EVENT_SERVER_SEND_NOTIFY_COMP で通知されます。		
Parameters:		
	conhdl	コネクションハンドル
	write_value	文字列
	write_length	文字数
Return:		
	RBLE_OK	正常終了
	RBLE_PARAM_ERR	パラメータ異常
	RBLE_STATUS_ERROR	状態エラー

7.3.3 Events

以下に、SCP 機能で定義されているイベントを表 7-3 にまとめ、イベントの詳細について説明します。

API 名	概要
RBLE_SCP_EVENT_SERVER_ENABLE_COMP	Server 有効設定完了イベント
RBLE_SCP_EVENT_SERVER_DISABLE_COMP	Server 無効設定完了イベント
RBLE_SCP_EVENT_SERVER_SEND_NOTIFY_COMP	Server 文字列送信完了イベント
RBLE_SCP_EVENT_SERVER_CHG_CHAR_IND	Server 文字列受信イベント

表 7-3 SCP 機能イベント一覧

7.3.3.1 RBLE_SCP_EVENT_SERVER_ENABLE_COMP

RBLE_SCP_EVENT_SERVER_ENABLE_COMP		
このイベントは、Server Role の有効設定(RBLE_SCP_Server_Enable)結果を通知します。		
Parameters:		
	status	RBLE_OK
	conhdl	コネクションハンドル

7.3.3.2 RBLE_SCP_EVENT_SERVER_DISABLE_COMP

RBLE_SCP_EVENT_SERVER_DISABLE_COMP		
このイベントは、Server Role の無効設定(RBLE_SCP_Server_Disable)結果を通知します。		
Parameters:		
	status	RBLE_OK
	conhdl	コネクションハンドル

7.3.3.3 RBLE_SCP_EVENT_SERVER_SEND_NOTIFY_COMP

RBLE_SCP_EVENT_SERVER_SEND_NOTIFY_COMP		
このイベントは、文字列の送信(RBLE_SCP_Server_Send_String)完了を通知します。		
Parameters:		
	status	RBLE_OK
	conhdl	コネクションハンドル

7.3.3.4 RBLE_SCP_EVENT_SERVER_CHG_CHAR_IND

RBLE_SCP_EVENT_SERVER_CHG_CHAR_IND		
このイベントは、文字列の受信を通知します。		
Parameters:		
	data_len	受信文字数
	data[]	受信文字列
	conhdl	コネクションハンドル