

LPRF BLE Porting Projects

Bluetooth Low Energy Main Page ^[1]

This wiki page explains necessary steps to port a BLE-Stack v1.x.x project running on a certain stack version to a newer one, and how to port between CC2540 and CC2541 projects.

Known Issues

This section lists the known issues and anomalies that impact the latest release. Issues that are resolved are listed in the release notes of the respective release and will be deprecated from this list.

- DMA Initialization of VLEN is not correct in the BIM function DMAExecCrc when supporting OAD. This can result in OAD CRC checksum failures in the BIM. To fix, add the following to line 345 of bim_main.c:

```
HAL_DMA_SET_VLEN( dmaCh0_p, HAL_DMA_VLEN_USE_LEN );
```

- Connections may drop after connecting with Android 7+ devices. This issues presents when SNV flash memory has high utilization. To fix, apply the workaround in the findItem() as described in this E2E forum post ^[2].
- A Slave initiated L2CAP Connection Parameter Update request via GAP_UpdateLinkParamReq will not be processed by the CC254x's BLE slave stack if the requested connection interval (CI) min/max range does not contain a range value that is a multiple of the current CI. The workaround is for the slave to request a min/max CI range that contains a multiple of the current CI. For example, if the current CI is 11.25ms, the request range should be min=7.5ms to max=X where X is >= than 22.5ms (2x 11.25ms).

Porting BLEv1.4.1 Projects to BLEv1.4.2

BLE v1.4.2 is a BLE-Stack maintenance release to the v1.4.1 release with all bug fixes applied in the protocol stack libraries. Refer to the release notes for a detailed list of changes in this release. Compared to the previous BLE v1.4.1 release, there are no API or SDK source code changes in BLE v1.4.2. Therefore, no application or project file porting is required when upgrading a project running BLE v1.4.1 to v1.4.2.

All documentation and examples from BLE v1.4.1 apply to BLE v1.4.2.

1. If the current project is based on a BLE-Stack SDK release prior to BLE v1.4.1 (for example, BLE v1.4.0), then follow the below porting instructions to first port your project to BLE v1.4.1. The following steps apply to projects already running BLE v1.4.1.
2. Copy all library files from the BLE v1.4.2 SDK to the existing BLE v1.4.1 SDK installation. The libraries are located at: \$1.4.2_INSTALL\$\Projects\ble\Libraries Where the default \$1.4.2_INSTALL\$ path is C:\Texas Instruments\BLE-CC254x-1.4.2.2. The entire v1.4.2 'Libraries' folder may be copy / pasted to the v1.4.1 installation.
3. Rebuild the project to link to the updated BLE v1.4.2 protocol stack libraries.

Porting BLEv1.4.0 Projects to BLEv1.4.1

Project Porting Directions

1. Move (or Copy) the project files from \$1.4.0_INSTALL\$\Projects\ble\PROJECT\$ to \$INSTALL\$\Projects\ble\PROJECT\$ where \$1.4.0_INSTALL\$ is the top level installation directory of the 1.4.0 stack, \$INSTALL\$ is the top level installation directory of the 1.4.1 stack, and \$PROJECT\$ is your project folder.
 2. If you modified any files from anywhere outside of the application folder, such as in the profiles at
-

\$1.4.0_INSTALL\$\Projects\Profiles, you will need to merge your changes with the new 1.4.1 version.

3. Open your project (now in the 1.4.1 folder) with IAR 9.10.3.

4. When prompted, choose yes to convert for use with new version.

5. In the Project options, in the Preprocessor options under the C/C++ Compiler tab, add "CC254x" to the controller include path: \$PROJ_DIR\$\\.\.\.\Components\ble\controller\CC254x\include

6. For any profiles, add the CC254x include path. Do not delete the original paths as they are still needed. Using SimpleBLEPeripheral as an example, add the following paths: \$PROJ_DIR\$\\.\.\Profiles\SimpleProfile\CC254x \$PROJ_DIR\$\\.\.\Profiles\Roles\CC254x However, do not delete the original paths: \$PROJ_DIR\$\\.\.\Profiles\SimpleProfile \$PROJ_DIR\$\\.\.\Profiles\Roles

7. Replace the .c files that were affected by the changes in step 6 by deleting the old reference and adding the new file. For example, delete peripheral.c from the SimpleBLEPeripheral project (under the PROFILES group). Then add peripheral.c from \$INSTALL\$\Projects\ble\Profiles\Roles\CC254x.

8. Compile and save your project.

API Changes

The following API's are now abstracted through gattservapp_util.c:

```
extern void GATTServApp_InitCharCfg( uint16 connHandle, gattCharCfg_t
*charCfgTbl );
```

```
extern uint16 GATTServApp_ReadCharCfg( uint16 connHandle, gattCharCfg_t
*charCfgTbl );
```

```
extern bStatus_t GATTServApp_ProcessCharCfg( gattCharCfg_t *charCfgTbl,
uint8 *pValue, uint8 authenticated, gattAttribute_t *attrTbl, uint16
numAttrs, uint8 taskId, pfnGATTReadAttrCB_t pfnReadAttrCB );
```

```
extern bStatus_t GATTServApp_ProcessCCCWriteReq( uint16 connHandle,
gattAttribute_t *pAttr, uint8 *pValue, uint8 len, uint16 offset, uint16
validCfg );
```

Therefore, the following file should be added to your project (preferably under the PROFILES group for consistency): \$INSTALL\$\Projects\ble\Profiles\GATT\ gattservapp_util.c

An additional "method" parameter has been added to profiles' read and write callback functions that are registered with GATTServApp_RegisterService() to indicate the type of read / write message.

```
typedef bStatus_t (*pfnGATTReadAttrCB_t)( uint16 connHandle,
gattAttribute_t *pAttr,
uint8 *pValue, uint8 *pLen,
uint16 offset,
uint8 maxLen, uint8 method );

typedef bStatus_t (*pfnGATTWriteAttrCB_t)( uint16 connHandle,
gattAttribute_t *pAttr,
uint8 *pValue, uint8 len,
uint16 offset,
uint8 method );
```

Also, the GAP_RegisterForHCIMsgs() command has been changed to GAP_RegisterForMsgs().

Typedef Changes

The Central Event Callback function in the Central GAPRole has changed from:

```
typedef void (*pfnGapCentralRoleEventCB_t)
(
    gapCentralRoleEvent_t *pEvent          //!< Pointer to event structure.
);
```

to:

```
typedef uint8 (*pfnGapCentralRoleEventCB_t)
(
    gapCentralRoleEvent_t *pEvent          //!< Pointer to event structure.
);
```

Structure Changes

Array Elements Changed to Pointers

In order to prepare for a possible future fragmentation implementation, GATT / ATT payload structures are no longer fixed sized arrays and are now pointers. Therefore, the payload is allocated based on its length. This means that the names of all payload elements in the structures in att.h and gatt.h have changed. A “p” has been added in front of the old array name and the next letter has been capitalized.

```
typedef struct
{
    uint16 handle;          //!< Handle of the attribute to be written (must be first field)
    uint16 offset;          //!< Offset of the first octet to be written
    uint8 len;              //!< Length of value
    uint8 value[ATT_MTU_SIZE-5]; //!< Part of the value of the attribute to be written
} attPrepareWriteReq_t
```

has been changed to:

```
typedef struct
{
    uint16 handle; //!< Handle of the attribute to be written (must be first field)
    uint16 offset; //!< Offset of the first octet to be written
    uint8 len;      //!< Length of value
    uint8 *pValue;  //!< Part of the value of the attribute to be written (0 to ATT_MTU_SIZE-5) - must be allocated
} attPrepareWriteReq_t;
```

Macro's have been added which can be used to access offsets from these pointer locations. For example,

```
typedef struct
    // Service found, store handles
    if ( pMsg->method == ATT_FIND_BY_TYPE_VALUE_RSP &&
        pMsg->msg.findByTypeValueResp.numInfo > 0 )
    {
        simpleBLESvcStartHdl =
pMsg->msg.findByTypeValueResp.handlesInfo[0].handle;
        simpleBLESvcEndHdl =
```

```
pMsg->msg.findByTypeValueRsp.handlesInfo[0].grpEndHandle;
}
```

has been changed to:

```
// Service found, store handles
if ( pMsg->method == ATT_FIND_BY_TYPE_VALUE_RSP &&
    pMsg->msg.findByTypeValueRsp.numInfo > 0 )
{
    simpleBLESvcStartHdl =
ATT_ATTR_HANDLE(pMsg->msg.findByTypeValueRsp.pHandlesInfo, 0);
    simpleBLESvcEndHdl =
ATT_GRP_END_HANDLE(pMsg->msg.findByTypeValueRsp.pHandlesInfo, 0);
}
```

where the macro's used are defined as:

```
#define ATT_ATTR_HANDLE( info, i )          ( BUILD_UINT16(
(info) [ATT_ATTR_HANDLE_IDX((i))], \

(info) [ATT_ATTR_HANDLE_IDX((i))+1] ) )
#define ATT_GRP_END_HANDLE( info, i )      ( BUILD_UINT16(
(info) [ATT_GRP_END_HANDLE_IDX((i))], \

(info) [ATT_GRP_END_HANDLE_IDX((i))+1] ) )
```

Additional Fields in Key Distribution Structre

The keyDist_t has several added fields which were previously reserved. These are needed for interoperability with Bluetooth 4.1 Security features. See an example of how to use this in the 1.4.1 HostTestRelease project.

```
typedef struct
{
    unsigned int sEncKey:1;    //!< Set to distribute slave encryption key
    unsigned int sIdKey:1;    //!< Set to distribute slave identity key
    unsigned int sSign:1;     //!< Set to distribute slave signing key
    unsigned int sLinkKey:1;  //!< Set to derive slave link key from slave LTK
    unsigned int sReserved:4; //!< Reserved for slave - don't use
    unsigned int mEncKey:1;    //!< Set to distribute master encryption key
    unsigned int mIdKey:1;    //!< Set to distribute master identity key
    unsigned int mSign:1;     //!< Set to distribute master signing key
    unsigned int mLinkKey:1;  //!< Set to derive master link key from master LTK
    unsigned int mReserved:4; //!< Reserved for master - don't use
} keyDist_t;
```

Default Value of HAL Components

The default values of HAL components, if not defined, have changed. See the `hal_board_cfg.h` file for a list of the default values. If you need to modify any of these, add a preprocessor definition. For example, `HAL_KEY` is now set to `TRUE` if not defined. Therefore, the `SimpleBLEPeripheral` project has added a new preprocessor definition: `HAL_KEY=FALSE`.

Allocating Memory for Over-the-Air Messages

As stated above, there have been changes made to prepare for a possible future fragmentation implementation. Therefore, it is now necessary to allocate memory for data sent over-the-air for ATT / GATT commands.

For example, a buffer must be allocated when sending a GATT_Notification. Note that this is done by the stack if the preferred method to send a GATT notification / indication is to be used. That is, using a profile's `SetParameter` function (i.e. `SimpleProfile_SetParameter()`) and calling `GATTServApp_ProcessCharCfg()`. See the `simpleGATTProfile.c` for an example of this.

If using `GATT_Notification()` or `GATT_Indication()` directly, this memory management will need to be added:

1. Attempt to allocate memory for the notification / indication using `GATT_bm_alloc()`.
2. If allocation succeeds, send notification / indication using `GATT_Notification()` / `GATT_Indication()`.
3. If the return value of the notification / indication is `SUCCESS (0x00)`, this means the memory was freed by the stack. If the return value is something other than `SUCCESS` (i.e. `blePending`), free the memory using `GATT_bm_free()`. There is an example of this in the `gattServApp_SendNotiInd()` function in `gattservapp_util.c`:

```

    noti.pValue = (uint8 *)GATT_bm_alloc( connHandle,
ATT_HANDLE_VALUE_NOTI,

                                     GATT_MAX_MTU, &len );

    if ( noti.pValue != NULL )
    {
        status = (*pfnReadAttrCB)( connHandle, pAttr, noti.pValue,
&noti.len,

                                     0, len, GATT_LOCAL_READ );

        if ( status == SUCCESS )
        {
            noti.handle = pAttr->handle;

            if ( cccValue & GATT_CLIENT_CFG_NOTIFY )
            {
                status = GATT_Notification( connHandle, &noti, authenticated );
            }
            else // GATT_CLIENT_CFG_INDICATE
            {
                status = GATT_Indication( connHandle, (attHandleValueInd_t
*)&noti,

                                     authenticated, taskId );
            }
        }

        if ( status != SUCCESS )
        {
            GATT_bm_free( (gattMsg_t *)&noti, ATT_HANDLE_VALUE_NOTI );

```

```

    }
}
else
{
    status = bleNoResources;
}

```

This will need to be done for other GATT messages also.

Allocation of Client Characteristic Configuration Table

The client characteristic configuration descriptors (CCCD's) are now initialized as pointers which must be allocated. For example, the CCCD for simpleProfileCharacteristic 4 is declared as a pointer in simpleGATTprofile.c:

```
static gattCharCfg_t *simpleProfileChar4Config
```

This must then be allocated when the profile is added to the application (in SimpleProfile_AddService()):

```

bStatus_t SimpleProfile_AddService( uint32 services )
{
    uint8 status;

    // Allocate Client Characteristic Configuration table
    simpleProfileChar4Config = (gattCharCfg_t *)osal_mem_alloc(
sizeof(gattCharCfg_t) *

linkDBNumConns );
    if ( simpleProfileChar4Config == NULL )
    {
        return ( bleMemAllocError );
    }
}

```

Porting BLEv1.3.2 Projects to BLEv1.4.0

1. Move (or Copy) the project files from *C:\Texas Instruments\BLE-CC254x-1.3.2\Projects\ble\PROJECT\$* to *C:\Texas Instruments\BLE-CC254x-1.3\Projects\ble\PROJECT\$* where *PROJECT\$* is your project folder.
2. If you modified any files from *C:\Texas Instruments\BLE-CC254x-1.3.2\Projects\Profiles*, you will need to merge your changes with the new 1.4.0 version.
3. Open your project (now in the 1.4.0 folder) with IAR 8.20.
4. When prompted, choose yes to convert for use with new version.
5. Compile and save your project.

You should be aware of two project-specific changes:

- The functionality of the PLUS_BROADCASTER define has changed so that the peripheralBroadcaster.c and peripheralBraodcaster.h files are no longer needed. See the modified simpleBLEperipheral project for how to accomodate this.
- Advertising will now begin again after a connection is dropped by default. To remove this, comment out line 1021 of peripheral.c (shown below):

```
VOID osal_set_event( gapRole_TaskID, START_ADVERTISING_EVT );
```

- All UUID's are now stored in centralized locations: gatt_uuid.h and gatt_profile_uuid.h

Also, the following API's have changed:

- extern bStatus_t GAP_TerminateLinkReq(uint8 taskID, uint16 connectionHandle, uint8 reason);
- the third parameter "reason" has been added to allow the application to indicate the termination reason to the connected device
- the GAPBondMgr_ProcessGAPMsg(gapEventHdr_t *pMsg) function is no longer a void-type function. It returns a uint8 which indicates TRUE if safe to deallocate the incoming GAP message and FALSE otherwise

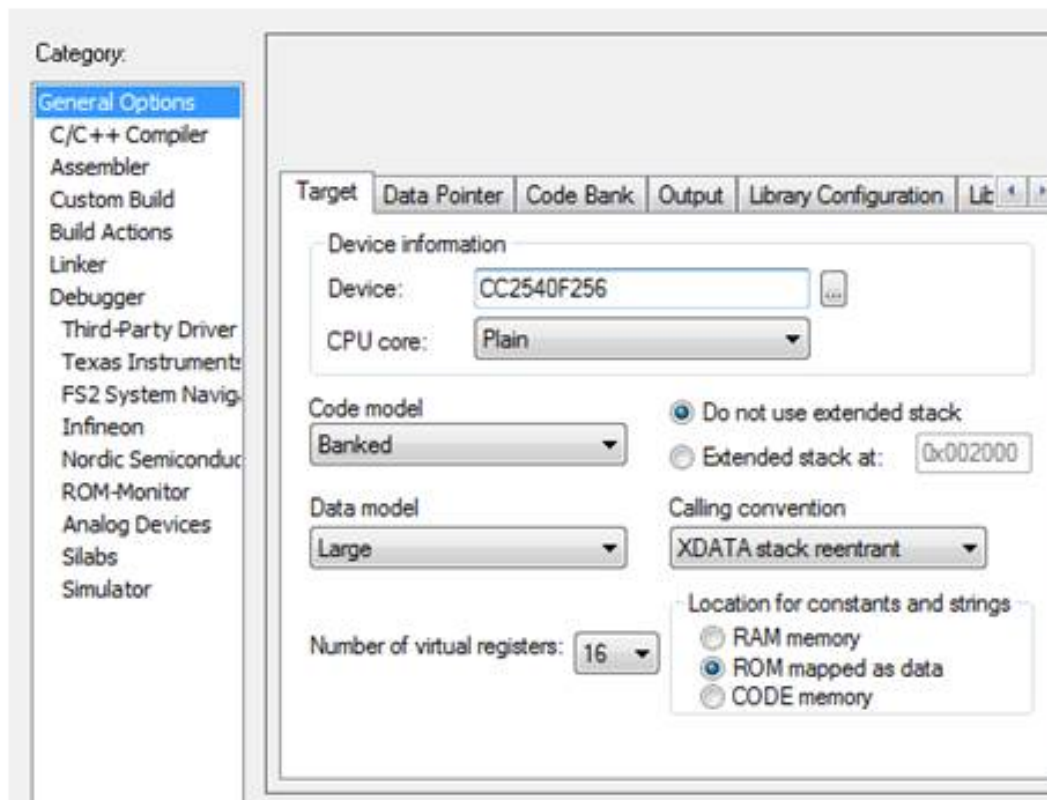
Porting BLEv1.2 Projects to BLEv1.3

1. Move (or Copy) the project files from *C:\Texas Instruments\BLE-CC254x-1.2\Projects\ble* to *C:\Texas Instruments\BLE-CC254x-1.3\Projects\ble*
2. Move (or Copy) the necessary profiles from *C:\Texas Instruments\BLE-CC254x-1.3\Projects\ble\Profiles* to *C:\Texas Instruments\BLE-CC254x-1.3\Projects\ble\Profiles*
3. Open the project and in the LIB group of the project.
 1. Remove all files
 2. Add CC254x_BLE.lib (or GAP role specific, for a more memory optimized library)
 3. Add CC254x_BLE_HCI_TL_None.lib for SoC solutions
4. Add NPI group (Right click in the workspace > "Add" > "Add Group..."). Add files np_i.h, np_i.c (np_i_uart.c and np_i_spi.c as well if needed)
5. Go to "Project" > "Options" > "General Options" > "Target" > "Number of virtual registers": Set number to **16**
6. Go to "Project" > "Options" > "C/C++ Compiler" > "Additional include directories": Add the line **\$PROJ_DIR\$\..\common\npi\npi_np**
7. Done

Porting from CC2540 to CC2541 Project

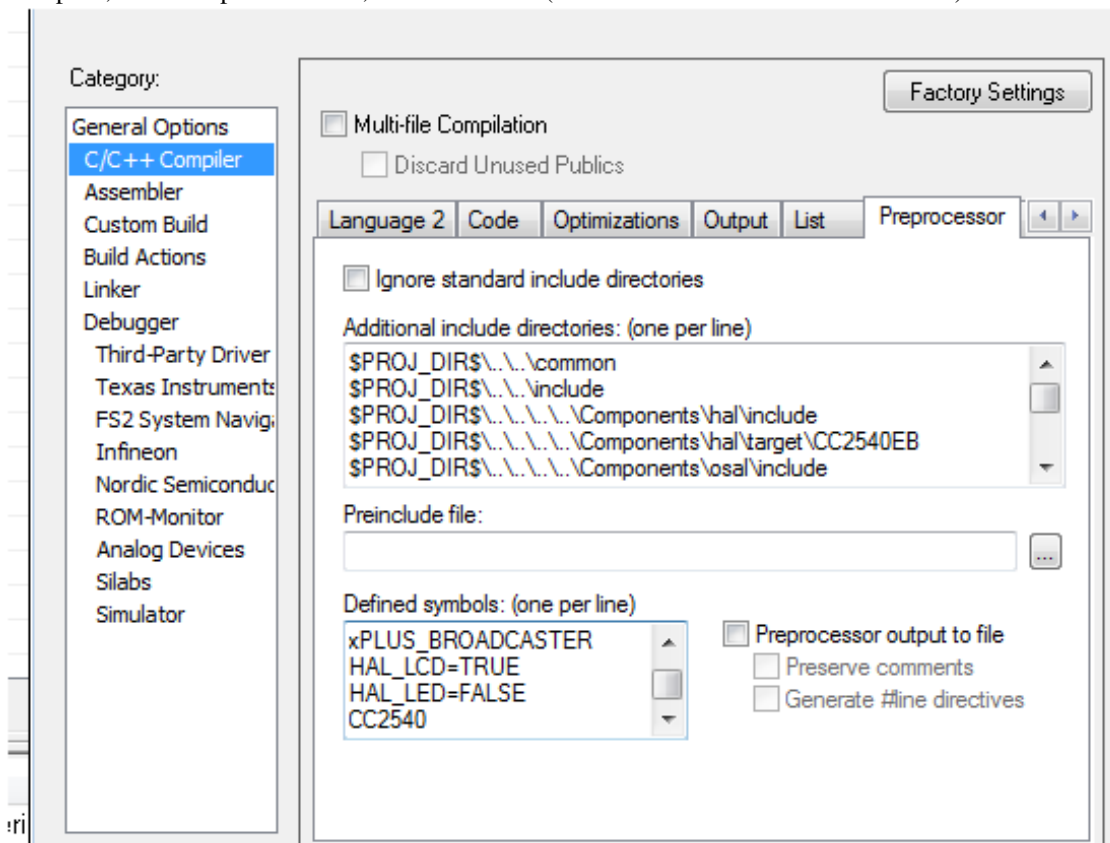
This section will describe how to port a CC2540 Project to work on a CC2541. Similar steps can be taken for the reverse direction.

1. Create and select a new CC2540 project configuration based on the CC2541 configuration under Project -> Edit Configurations:
2. In the project options, under general options, in the target tab, change the "Device" to CC2540F256:



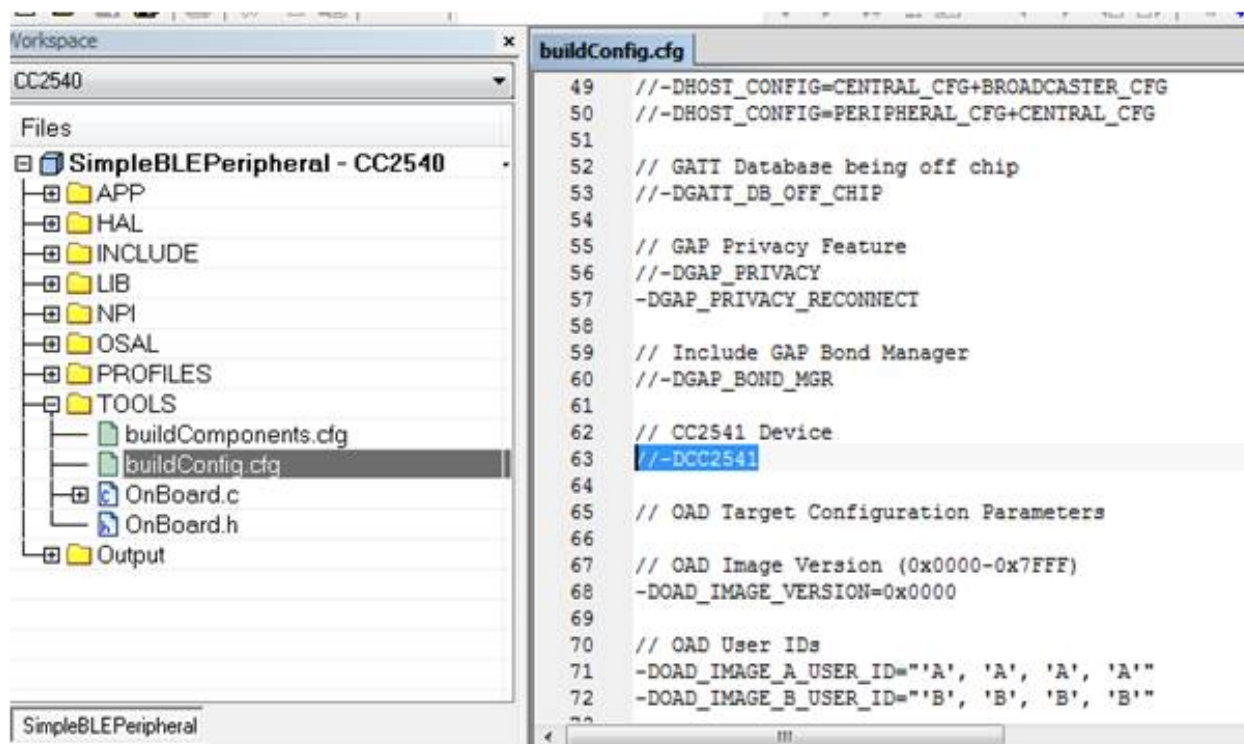
3. Under C/C++

Compiler, in the Preprocessor tab, define CC2540 (and make sure CC2541 is not defined):



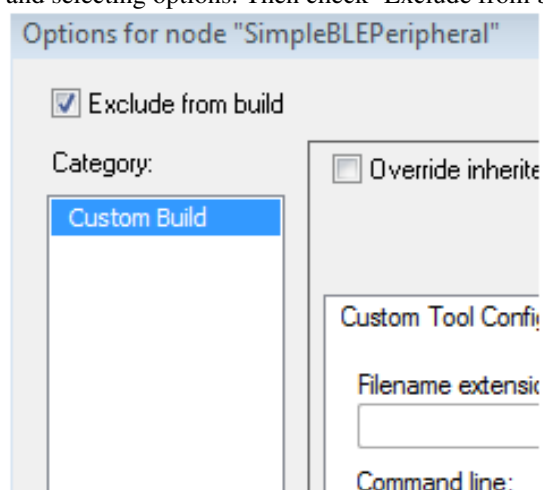
4. In the

Workspace pane, under the TOOLS group, open buildConfig.cfg, and comment out the CC2541 define:

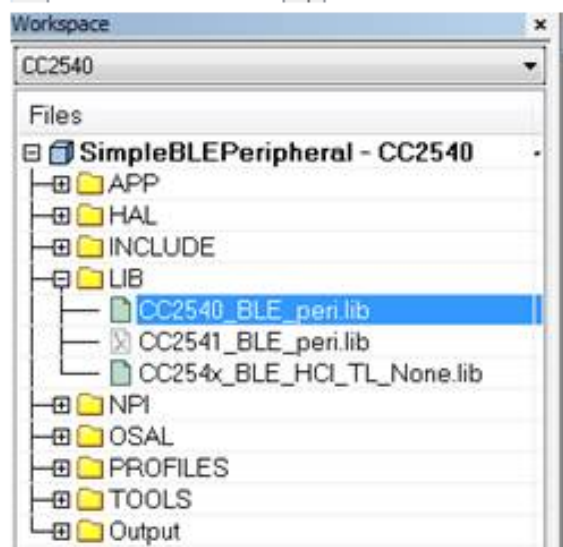


5.

In the Workspace pane, under the LIB group, exclude the 2541 library by right clicking on CC2541_BLE_peri.lib and selecting options. Then check "Exclude from build:"



6. Add the CC2540_BLE_peri.lib to the library group:



7. You also have to ensure to exclude the CC2540 library

from the other CC2541 configurations.

References

- [1] <http://processors.wiki.ti.com/index.php/Category:BluetoothLE>
- [2] https://e2e.ti.com/support/wireless_connectivity/bluetooth_low_energy/f/538/t/457958

Article Sources and Contributors

LPRF BLE Porting Projects *Source:* <http://processors.wiki.ti.com/index.php?oldid=227540> *Contributors:* Gstewart, JLindh, JXS, T Camise

Image Sources, Licenses and Contributors

File:Step2 porting.png *Source:* http://processors.wiki.ti.com/index.php?title=File:Step2_porting.png *License:* unknown *Contributors:* T Camise

File:Step3 porting.png *Source:* http://processors.wiki.ti.com/index.php?title=File:Step3_porting.png *License:* unknown *Contributors:* T Camise

File:Step4 porting.png *Source:* http://processors.wiki.ti.com/index.php?title=File:Step4_porting.png *License:* unknown *Contributors:* T Camise

File:Step5 porting.png *Source:* http://processors.wiki.ti.com/index.php?title=File:Step5_porting.png *License:* unknown *Contributors:* T Camise

File:Step6 porting.png *Source:* http://processors.wiki.ti.com/index.php?title=File:Step6_porting.png *License:* unknown *Contributors:* T Camise