Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Freescale BeeStack™

Application Development Guide

Document Number: BSADG
Rev. 1.1
01/2008

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Contents

Chapter 5
Managing BeeStack Resources

Chapter 6
Debugging BeeStack Applications

About This Book

The *BeeStack Application Development Guide* describes how to develop an application for BeeStack, including discussions on major considerations for commercial applications.

Audience

This document is intended for software developers who write applications for BeeStack-based products using Freescale development tools.

It is assumed the reader is a programmer with at least rudimentary skills in the C programming language and that the reader is already familiar with the edit/compile/debug process.

Organization

This document is organized into the following sections.

Chapter 1	Introduction – provides an overview of the BeeStack Application Development Guide, including what's included and what is not in the guide. It also describes the basic development process using both BeeKit and CodeWarrior (only in concept. This guide is not a user guide for either BeeKit or CodeWarrior).
Chapter 2	Building A Custom Application – provides a step-by-step example of creating a custom sample application.
Chapter 3	Designing A Custom Profile – describes designing a new custom-profile application, including selecting a profile, clusters, attributes and endpoints. It also describes ZigBee 2006 security options.
Chapter 4	Selecting Platform Components – describes selecting the appropriate hardware-related platform components, including the use of non-volatile memory, LEDs, the keyboard, serial port, and general hardware selection.
Chapter 5	Managing BeeStack Resources – describes using the non-hardware-related platform components appropriately, including the use of timers, messages, data queues, the task scheduler, Non-volatile-memory and low power library. It also describes how to determine how much RAM and Flash is available to the application and what to do if an application exceeds memory size.
Chapter 6	Debugging BeeStack Applications – describes how to debug an application that may not work, including use of the BDM, LEDs, ZigBee Test Client and Sensor Network Analyzers.

Revision History

The following table summarizes revisions to this document since the previous release (Rev. 1.0).

Revision History

Location	Revision
Entire Document	Minor corrections for SW maintenance release.

Conventions

This *BeeStack Documentation Overview* uses the following formatting conventions when detailing commands, parameters, and sample code:

`Courier mono-space` type indicates commands, command parameters, and code examples.

Bold style indicates the command line elements, which must be entered exactly as written.

Italic type indicates command parameters that the user must type in or replace, as well as emphasizes concepts or foreign phrases and words.

Definitions, Acronyms, and Abbreviations

ACK	Acknowledgement
ADC	Analog to digital converter
AF	Application framework
AIB	Application support sub-layer information base
APDU	Application support sub-layer protocol data unit
API	Application programming interface
APL	Application layer
APS	Application support sub-layer
APSDE	APS data entity
APSDE-SAP	APS data entity - service access point
APSME	APS management entity
APSME-SAP	APS management entity - service access point
ASDU	APS service data unit
Binding	Matching ZigBee devices based on services and needs
BTR	Broadcast transaction record, the local receipt of a broadcast message
BTT	Broadcast transaction table, holds all BTRs
CBC-MAC	Cipher block chaining message authentication code
CCA	Clear channel assessment
Cluster	A collection of attributes associated with a specific cluster-identifier
Cluster identifier	An enumeration that uniquely identifies a cluster within an application profile

CSMA-CA	Carrier sense multiple access with collision avoidance
CTR	Counter
Data Transaction	Process of data transmission from the endpoint of a sending device to the endpoint of the receiving device
Device/Node	ZigBee network component containing a single IEEE 802.15.4 radio
Direct addressing	Direct data transmission including both destination and source endpoint fields
Endpoint	Component within a unit; a single IEEE 802.15.4 radio may support up to 240 independent endpoints
IB	Information base, the collection of variables configuring certain behaviors in a layer
IEEE	Institute of Electrical and Electronics Engineers, a standards body
Indirect addressing	Transmission including only the source endpoint addressing field along with the indirect addressing bit
ISO	International Standards Organization
LCD	Liquid crystal display
LED	Light-emitting diode
LQI	Link quality indicator or indication
MAC	Medium access control sub-layer
MCPS-SAP	MAC common part sub-layer - service access point
MIC	Message integrity code
MLME	MAC layer management entity
MLME-SAP	MAC sub-layer management entity service access point
NIB	Network layer information base
NLDE	Network layer data entity
NLDE-SAP	Network layer data entity - service access point
NLME	Network layer management entity
NLME-SAP	Network layer management entity - service access point
NPDU	Network protocol data unit
NSDU	Network service data unit
NVM	Non-volatile memory
NWK	Network layer
Octet	Eight bits of data, or one byte
OSI	Open System Interconnect
PAN	Personal area network
PD-SAP	Physical layer data - service access point
PDU	Protocol data unit (packet)

PHY	Physical layer
PIB	Personal area network information base
PLME-SAP	Physical layer management entity - service access point
Profile	Set of options in a stack or an application
RF	Radio frequency
SAP	Service access point
SKG	Secret key generation
SKKE	Symmetric-key key establishment protocol
SSP	Security service provider, a ZigBee stack component
Stack	ZigBee protocol stack
UART	Universal asynchronous receiver transmitter
WDA	wireless demo application
WPAN	wireless personal area network
ZDO	ZigBee device object(s)
ZDP	ZigBee device profile
802.15.4	An IEEE standard radio specification that underlies the ZigBee Specification

Reference Materials

This following served as references for this manual:

1. Document 053474r13, *ZigBee Specification*, ZigBee Alliance, December 2006
2. Document 075123r00, *ZigBee Cluster Library Specification*, ZigBee Alliance, July 2007
3. Document 053520r24, *Home Automation Profile Specification*, ZigBee Alliance, September 2007

Chapter 1
Introduction

Freescale's BeeStack is a complete, robust implementation of the ZigBee 2006 networking specification. BeeStack applications typically are used in wireless sensor and control networks.

This section provides an overview of *BeeStack Application Development Guide*, describing what is and what is not included in the guide. It also describes the basic development process for BeeStack applications using both BeeKit and CodeWarrior.

This guide is a "how-to" guide that leads a developer through the process of developing BeeStack applications. It also includes advice on building robust networks and managing network resources.

At the time of writing, BeeStack is written for the Freescale HCS08 microcontroller, but the concepts in the document apply to BeeStack ported to any microcontroller.

1.1 What This Document Describes

This guide describes the following:

- How to build and customize BeeStack applications for use in wireless sensor and control applications
- A step-by-step example of modifying a BeeStack application
- How to design a custom ZigBee application profile and the intended use of application profiles, endpoints, clusters and attributes. It includes how to manage bandwidth and channels
- A suggested process for selecting the appropriate hardware platform components
- Suggestions on how best to use the Freescale task scheduler, timers, memory and other platform resources
- How to debug an application

1.2 What This Document Does Not Describe

This guide does not describe the following:

- How to install BeeKit. For the BeeKit installation process, see the *BeeKit Wireless Connectivity Toolkit User's Guide*
- How to install CodeWarrior. For instructions, see the CodeWarrior documentation
- The complete BeeStack API in detail. For the BeeStack API, see the *BeeStack Software Reference Manual*
- How to port applications from any other stack including any implementation of the ZigBee 2004 specification.
- How to port BeeStack to a custom board

- ZigBee networking in general. For an overview of ZigBee, see the *BeeStack Software Reference Manual* and the *ZigBee Specification*
- How to use the BeeStack Sample Applications. For the user interface to the sample applications, see *Freescale ZigBee Application User's Guide*
- The HCS08 microcontroller. See the *MC9S08GB/GT Data Sheet* for more information on this Freescale 8-bit microcontroller

1.3 BeeKit

BeeKit is a desktop PC graphical application that allows developers to configure Freescale networking solutions, including BeeStack, IEEE® 802.15.4 MAC, and the Freescale proprietary Simple MAC (SMAC). Figure 1-1 shows the BeeKit Wireless Connectivity start-up window.

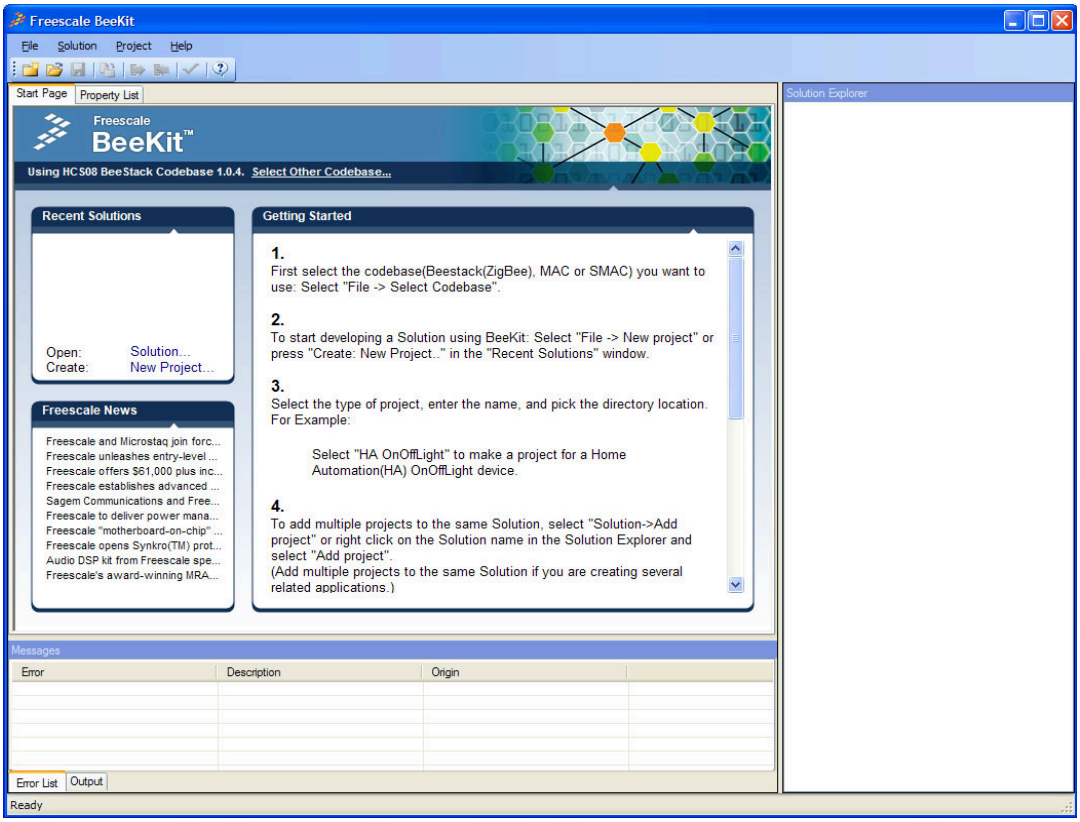

Figure 1-1. BeeKit Starting Window

BeeKit creates a sample application from templates, providing the ability to set properties (also called compile-time options) which configure the application and BeeStack. The resulting project may then be exported as an XML file to a file folder and imported into CodeWarrior for editing, compiling and debugging. In addition, BeeKit provides a quick-start wizard that can prepare and configure sample applications in moments.

BeeKit provides compile-time configuration of BeeStack and applications during the entire life of the project.

See the BeeKit documentation for more information.

1.4 CodeWarrior

CodeWarrior is a desktop PC integrated development environment (IDE) which includes a C compiler for the HCS08 MCU and the other tools to generate a downloadable image as well as a debugger that can download code into the HCS08 MCU's flash memory.

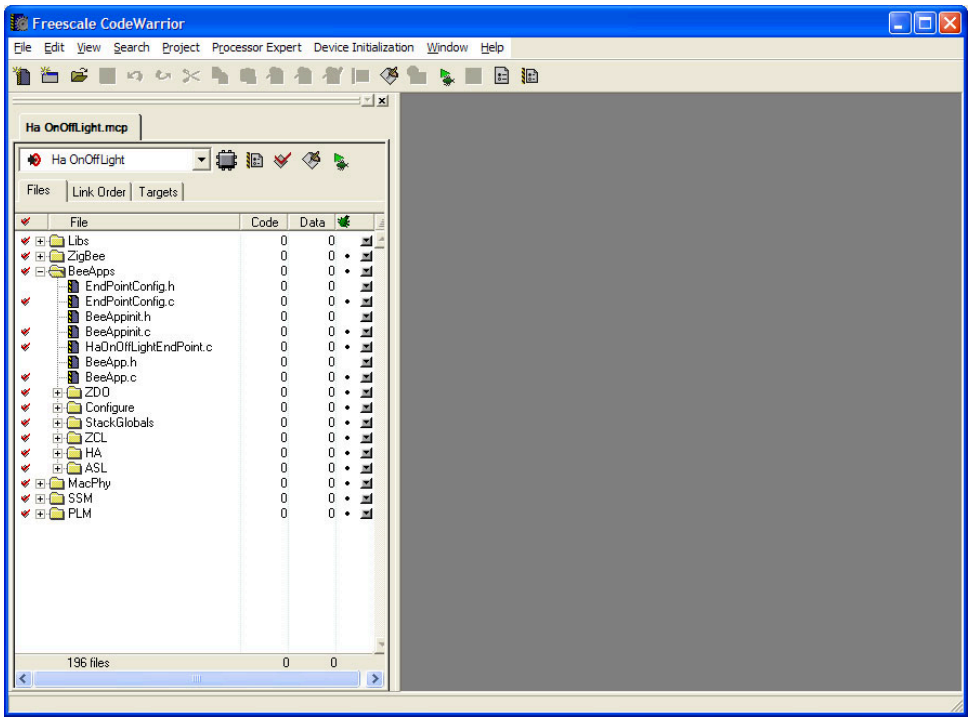

Figure 1-2. Freescale CodeWarrior

CodeWarrior takes the output of BeeKit (an XML file that it can import and convert into a project file and a source directory tree) and compiles and links the C source code and libraries into a binary image that may be downloaded into the Flash memory of an HCS08 MCU using the background debug memory (BDM) port.

See the CodeWarrior documentation for more information.

1.5 BeeStack

BeeStack is the term used to describe all of the software placed into target boards, with the exception of the application. BeeStack is comprised of ZigBee networking components, which provide access to ZigBee networking functionality, and platform components, which provide a framework for the application to operate and access the hardware.

The components of BeeStack are shown in Figure 1-3.

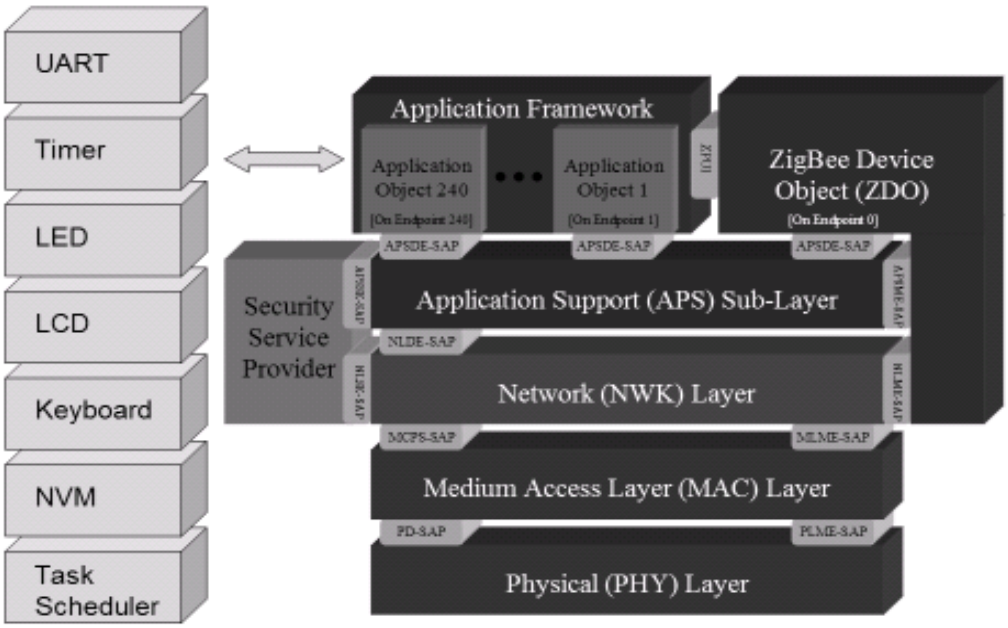

Figure 1-3. BeeStack Components

The networking (NWK) task in BeeStack is responsible for routing packets, including broadcasting, route discovery, unicasting and rejecting packets not for this node or network.

The Application Support Sub-layer (APS) task is responsible for delivering and receiving application data, including binding endpoints, and end-to-end acknowledgements. APS also contains the authentication process for secure networks, including the trust center on ZigBee Coordinator (ZC) nodes.

The Application Framework (AF) task is responsible for delivery of data indications and confirms to the application endpoints.

The ZigBee Device Object (ZDO) task is responsible for the state of the network, and it includes functions to join and leave the network.

The ZigBee Device Profile (ZDP) task handles requests and responses for a set of common over-the-air ZigBee commands for managing nodes within the network. For example, any node may ask for the IEEE (or MAC) address of any other node in the network using a ZDP command.

The various platform management (PLM) components are responsible for interacting with the hardware such as switches, LEDs, the LCD or timers. All of the PLM components may be customized for a particular application.

1.6 The Development Process

Developing applications for BeeStack is similar to any embedded development. The addition of BeeKit makes starting and configuring a new application very easy. The steps are as follows.

1. Design the application.
2. Use BeeKit to create the application framework from a template and to configure the application to include the appropriate components, property settings and endpoint settings.
3. Export the application solution from BeeKit and import it into CodeWarrior.
4. Edit the application as necessary, adding custom code.
5. Compile the application.
6. Download the application into a target board.
7. Debug the application (see Chapter 6, “Debugging BeeStack Applications”).
8. Repeat steps 4-7 as necessary.
9. If creating more than one application, use BeeKit to add another application to the BeeKit solution and repeat steps 2-8 as necessary.

Chapter 2
Building A Custom Application

This chapter provides a step-by-step example of how to create a sample custom application. Chapter 3, "Designing A Custom Profile" and Chapter 4, "Selecting Platform Components" explain in more detail about how to build applications for BeeStack.

The general process for creating this custom application is as follows:

- Create the project in BeeKit from an existing application template, making custom property settings and endpoint settings
- Export the project from BeeKit
- Import the project into CodeWarrior
- Edit the application in CodeWarrior to remove unneeded functionality from the template code and add the new functionality of the application
- Compile the custom application with CodeWarrior
- Download the custom application in the target board with CodeWarrior
- Debug the custom application

When building a custom application, always start with a template application in BeeKit. In this case the example will start with the Generic Application Template and transform it into a custom application.

The Generic Application Template by default uses the accelerometer hardware available in the Freescale SRB and SARD boards to determine tilt of the board and transmit this data to a remote node for display on that remote node. The same code is used for both the accelerometer and display nodes (that is, a node can assume either role).

The custom application described in this chapter will ignore the accelerometer; it will simply flash a light (LED2) on the remote display for a one-second period.

NOTE

Both the Generic Application and this Custom Application use what is called a private profile. Private profiles are useful for those application that do not need to interoperate at an ZigBee application level with other vendors' applications. For Public Profiles which do interoperate (such as a Home Automation On/Off Light and Switch), see the *ZigBee Cluster Library Reference Manual*.

The keys for the Custom Application will be as follows:

Table 2-1. Custom Application Keys

Switch	Description
SW1	Form (ZC) or join (ZR, ZED) the network
SW2	Flash remote light (LED2)
Long SW2	No action
SW3	Find a remote node with a light for sending light commands to
SW4	No action

2.1 Creating a Custom Application In BeeKit

This section describes the steps required in BeeKit to create the custom application. This is not a BeeKit tutorial and assumes users have some familiarity with BeeKit.

1. Create a new project starting from GenericApp named ZcNcbCustomApp. The solution should be named CustomApp. This example uses the Freescale NCB board. If using another board for the ZigBee Coordinator besides the NCB (such as the QE128-EVB), use the name of that board in the name of the project. Using the three-part naming convention for projects, with the ZigBee node type (ZC, ZR or ZED), the Freescale board type (NCB, SRB, QE128-EVB), and the application name (CustomApp) allows any project to be easily recognized by name. The location of this project should be the folder: `C:\BeeStack`. Also users may select another path, but this path should be known.

In the BeeStack configuration wizard, make sure to select the MC1321x-NCB board as the hardware target (or the board already chosen), with LCD Display module enabled in the Platform Modules Page, ZTC disabled, ZigBee Coordinator device type, no security without mesh routing network type, default extended address and PAN ID and channel 25 as the default channel.

2. Modify the endpoint and simple descriptor to contain the information depicted in figure 2.2 below (endpoint number 1, profile 0xc021, application id 0x1234, an input and output OnOff cluster 0x0100).

To edit the endpoint, click on the button in the BeeKit window as circled in Figure 2-1. This button will only be available if the “Generic Endpoint” is selected in the Solution Explorer window.

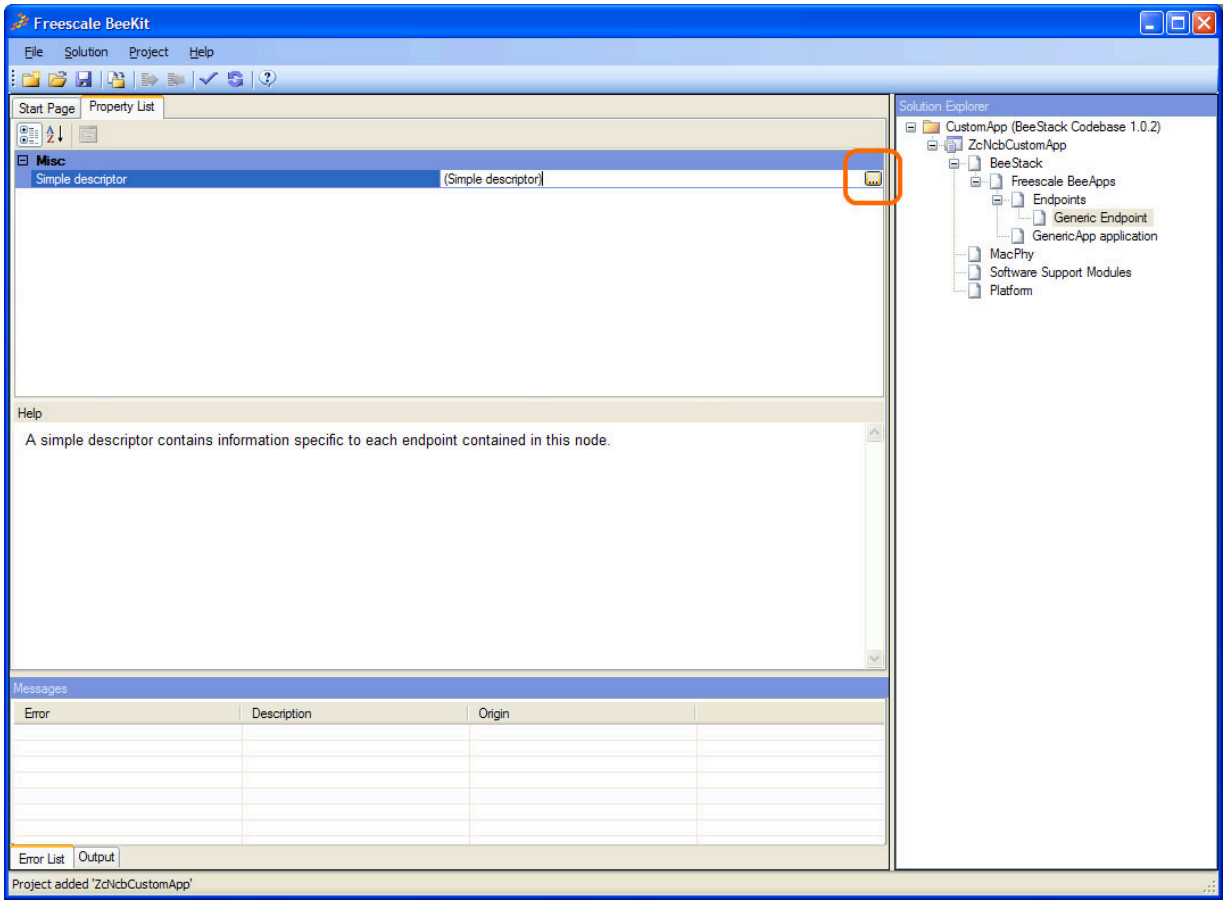

Figure 2-1. Modifying Endpoints in BeeKit

After clicking this button, the Simple Descriptor Editor window appears as shown in Figure 2-2. Modify the endpoint's simple descriptor to contain the same information as shown in Figure 2-2.

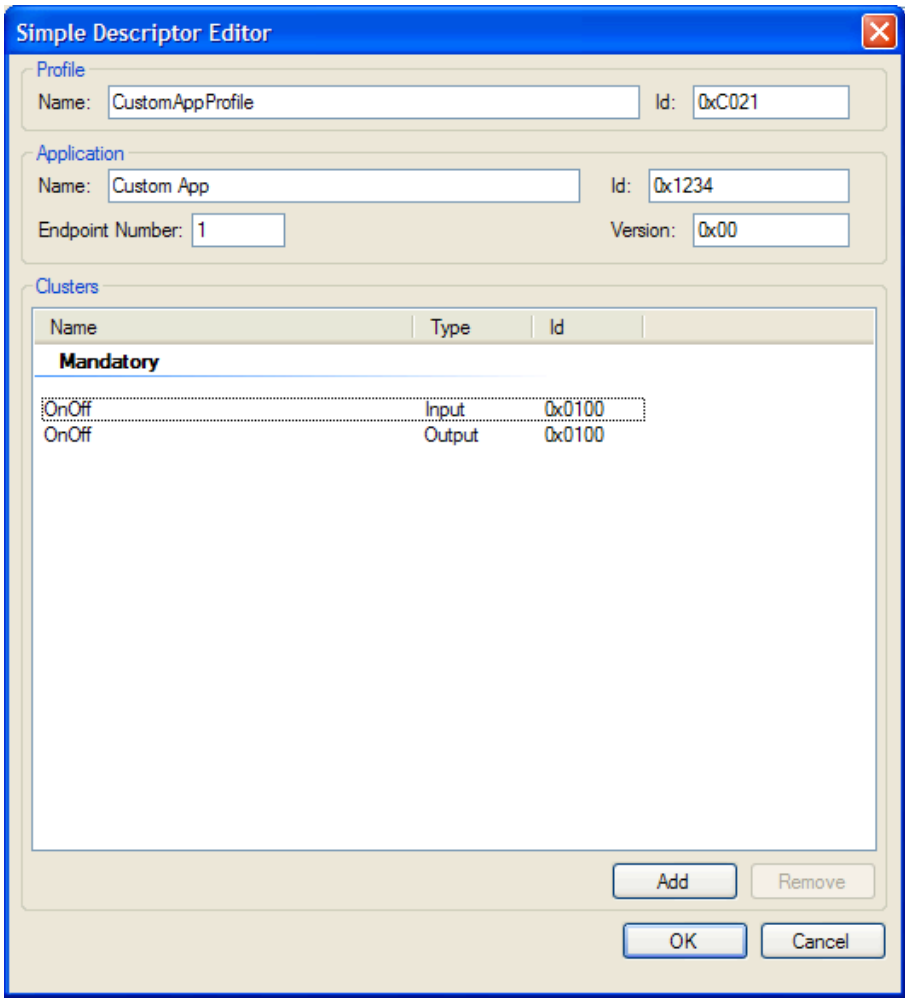

Figure 2-2. Custom Application Simple Descriptor

3. Add another project in the same solution starting from GenericApp named ZedSrbCustomApp (from the menu, use Solution ->Add New Project…). This project will be used with a Freescale SRB board. Again, if selecting a different Freescale board, change the project name accordingly. This time, make sure to select ZigBee node type of ZigBee End Device in the BeeStack configuration wizard on the "Select ZigBee Device Type" page. All other settings are the same as in Step 1.
4. Change the endpoint's simple descriptor as described in Step 2.
5. Export the solution to the folder `C:\BeeStack\CustomApp` (from the menu, use Solution ->Export Solution…). Users can employ another path; BeeKit and CodeWarrior place no restriction on using another path.

Once the solution (containing two projects) is exported, the following two directories will exist:

```
C:\BeeStack\CustomApp\ZcNcbCustomApp
C:\BeeStack\CustomApp\ZedSrbCustomApp
```

At this point, the application is ready for importing into CodeWarrior and editing to contain custom code.

2.2 Editing the Custom Application in CodeWarrior

This section describes the steps involved to transform the source code from the Generic Application Template to the custom application.

1. Start CodeWarrior 6.1. If CodeWarrior 6.1 (or later) is not installed on the system, obtain a copy from Freescale. The Standard Edition or better is required (the Special Edition cannot accept a project with so many files).
2. Import the project that was exported by BeeKit into CodeWarrior. From the menu, choose File ->Import Project… and navigate to `BeeStack\CustomApp\ZcNcbCustomApp`. A file called `ZcNcbCustomApp.xml` is in that directory. Import the `.xml` file and call the resulting project `ZcNcbCustomApp.mcp`.
3. Compile the project to make sure everything in BeeKit and CodeWarrior worked. The project should compile without warnings or errors. To compile, click on the compile icon as shown in Figure 2-3.

Figure 2-3. Compile Icon

4. Edit the `BeeApp.c` file to contain the custom code. This is the longest step and requires a number of edits as outlined below. `BeeApp.c` can be found in the BeeApps folder.

In `BeeApp.c`, find the BeeAppInit() function (line 205). The key combination Ctrl-G in the CodeWarrior editor will go to a specified line. Ctrl-F will find text.

In BeeAppInit(), change the name of the application. Change the following lines from:

```
/* indicate the app on the LCD */
LCD_WriteString(2, "Accelerometer");
```

To:

```
/* indicate the app on the LCD */
LCD_WriteString(2, "CustomApp");
```

In BeeAppInit(), remove the accelerometer timer and initialization code. Change the following lines from:

```
/* allocate timers for use by this application */
appTimerId = TMR_AllocateTimer();
accelModeTimerId = TMR_AllocateTimer();

/* initialize accelerometer */
AccelerometerInit();
```

To:

```
/* allocate timers for use by this application */
appTimerId = TMR_AllocateTimer();
```

Next, modify the defines for the Accelerometer to be those for the custom app. Find these defines (line 110):

```
/* BeeAppTask events */
#define accelEventReport_c  (1 << 0)  /* send a report */
#define accelEventState_c   (1 << 1)  /* move on to next state */
#define accelEventDisplay_c (1 << 2)  /* display data */
```

Add an event, so the lines read:

```
/* BeeAppTask events */
#define accelEventReport_c  (1 << 0)  /* send a report */
#define accelEventState_c   (1 << 1)  /* move on to next state */
#define accelEventDisplay_c (1 << 2)  /* display data */
#define customAppTurnOffLed2_c  (1 << 3)  /* turn off light */
```

Find the function, BeeAppTask() (line 257). Find the lines that read:

```
/* display the accelerometer data */
if(events & accelEventDisplay_c) {
AccelerometerDisplayData();
}

/* report accelerometer data */
if(events & accelEventReport_c) {

/* report data over-the-air (assumes SW3 has been pressed to find display) */
AccelerometerReportData();

/* start up state machine again */
giAccelDemoState = accelStateStart_c;
TS_SendEvent(gAppTaskID, accelEventState_c);
}

/* handle accelerometer events */
if(events & accelEventState_c) {
AccelerometerStateMachine(giAccelDemoState);
}
```

And replace these lines in BeeAppTask() with the following lines of code:

```
if(events & customAppTurnOffLed2_c)
LED_SetLed(LED2, gLedOff_c);
```

Next, find the function BeeAppHandleKeys() (line 290). Find the lines that read as follows, and remove them:

```
uint8_t led;
uint8_t accelData;
```

Next find the lines in BeeAppHandleKeys() that read:

```
case gKBD_EventSW2_c:
      /* walk through value of accelerometer */
      accelData = gaAccelDemoXYZ[giAccelIndex];
      if(accelData < accelDemo1Led_c)
        accelData = accelDemo1Led_c;
      else if(accelData < accelDemo2Leds_c)
        accelData = accelDemo2Leds_c;
      else if(accelData < accelDemo3Leds_c)
        accelData = accelDemo3Leds_c;
      else if(accelData < accelDemo4Leds_c)
        accelData = accelDemo4Leds_c;
      else
        accelData = 0;
      gaAccelDemoXYZ[giAccelIndex] = accelData;

      /* display on LEDs, LCD */
```

```
      TS_SendEvent(gAppTaskID, accelEventDisplay_c);
      break;
```

And replace them with:

```
 case gKBD_EventSW2_c:
      CustomAppToggleLED2();
      break;
```

Remove the code from the case statements for gKBD_EventSW4_c and gKBD_EventLongSW2_c so they have no action, as shown below:

```
 case gKBD_EventSW4_c:
      break;
```

Find the `BeeAppDataIndication()` function (line 360). Find the text that reads:

```
 if(pIndication->aClusterId[0] == appDataCluster[0]) {

      /* indicate we're the display */
      gfAccelIsDisplay = TRUE;

      /* get the new accelerometer readings */
      FLib_MemCpy(gaAccelDemoXYZ, pIndication->pAsdu, sizeof(gaAccelDemoXYZ));

      /* update display with new data */
      TS_SendEvent(gAppTaskID, accelEventDisplay_c);
    }
```

And change that text to read:

```
if(pIndication->aClusterId[0] == appDataCluster[0]) {
         LED_SetLed(LED2, gLedOn_c);
         TMR_StartSingleShotTimer(appTimerId, 1000, CustomAppTimerCallBack);

         /* update display with new data */
         TS_SendEvent(gAppTaskID, accelEventDisplay_c);
    }
```

Add the following two functions at the end of the file:

```
void CustomAppTimerCallBack
  (
  tmrTimerID_t timerId  /* IN: */
  )
{
  (void)timerId;  /* to prevent compiler warnings */

  TS_SendEvent(gAppTaskID, customAppTurnOffLed2_c);
}

void CustomAppToggleLED2
  (
  void
  )
{
  afAddrInfo_t  addrInfo;

  /* don't have a place to send data to, give up */
  if(!gfAccelFoundDst)
    return;
```

```
  /* set up address information */
        addrInfo.dstAddrMode = gZbAddrMode16Bit_c;
  Copy2Bytes(addrInfo.dstAddr.aNwkAddr, gaAccelDstAddr);
  addrInfo.dstEndPoint = gAccelDstEndPoint;
  addrInfo.srcEndPoint = appEndPoint;
  addrInfo.txOptions = gApsTxOptionNone_c;
  addrInfo.radiusCounter = afDefaultRadius_c;

  /* set up cluster */
  Copy2Bytes(addrInfo.aClusterId, appDataCluster);

  /* send the data request */
  (void)AF_DataRequest(&addrInfo, 10, "ToggleLed2", NULL);
}
```

Finally, add prototypes for those functions in the "Private Prototypes" section of the file, near line 140.

```
void CustomAppTimerCallBack ( tmrTimerID_t timerId );
void CustomAppToggleLED2(void);
```

At this point, all changes are made to the code. Press Ctrl-S to save the file.

5. Make sure the code compiles without errors or warnings (if users cut and paste from this document, it should). Resolve any compiler warnings or errors before running the custom application. (See Section 2.3, "Installing and Running The Custom Application")
6. Copy the `BeeApp.c` file created in the `BeeStack\CustomApp\ZcNcbCustomApp\BeeApps` directory to the `BeeStack\CustomApp\ZedSrbCustomApp\BeeApps` directory. This overwrites the previous `BeeApp.c` in that directory.
7. Import and compile the ZedSrbCustomApp application. CodeWarrior allows multiple projects to be open at the same time.

2.3 Installing and Running The Custom Application

This section describes how to download the custom application created in the previous section.

1. Connect the P&E USB Multilink pod to the BDM port on the NCB board. Click the green debug icon in the ZcNcbCustomApp project to download the code to the NCB board. Note: the red portion of the ribbon cable should be toward the edge of the board. The 6-pin connector is labelled BDM.
2. Connect the P&E USB Multilink pod to the BDM port on the SRB board. Click the green debug icon in the ZedSrbCustomApp project to download the code to the SRB board. Note: the red portion of the ribbon cable should be toward the edge of the board. The 6-pin connector is labelled BDM. Disconnect the BDM pod.
3. Reset each board. Press SW1 on each board. The LEDs should chase each other for a few seconds while BeeStack forms a ZigBee network.
4. Tell each board to find the other in the network by pressing SW3 on each board. LED3 should light indicating a remote custom application was found.
5. Press SW2 on the device that has LED3 on to toggle the remote LED2 on for 1 second.

2.4 Examining the Custom Application

In addition to using the tools, this example demonstrates a number of concepts.

- All application initialization takes place in BeeAppInit().
- Events for the application task come into the function BeeAppTask().
- Incoming ZigBee messages come into the function BeeAppDataIndication().
- Keyboard events come into the function BeeAppHandleKeys().

Examine the BeeAppDataIndication() function. Notice the newly added code both starts a timer and sends an event to the application task. The timer is used to turn off the LED that was turned on in the data indication handler.

Notice also the data indication handler didn't need to worry about the application profile or endpoints, because these are taken care of when the application registered the endpoint in BeeAppInit(). The lower layers will filter any incoming data not for that registered endpoint or on the wrong application profile ID. The application needs only to concern itself with clusters.

The cluster ID itself was retrieved from the endpoint's simple descriptor, as found in BeeAppDataIndication().

Examine the function BeeAppHandleKeys(). Note how SW1 starts the network with a single call to ZDO. Note how SW3 finds the other node using the ASL_MatchDescriptor_req() function. The results of that function come back to the callback registered in BeeAppInit(), in the line that reads:

```
Zdp_AppRegisterCallBack(BeeAppZdpCallBack);
```

The function BeeAppZdpCallBack() stores the results of a successful match descriptor so that LED3 can be lit and the application can now know which node to send its commands to. Note that match descriptor will return ALL nodes that match, so it's only useful if the application knows there will be no or only a few nodes in the network with the same profile ID and cluster list as described by their endpoint's simple descriptor.

This same set of nodes will work in any sized ZigBee network, filled with many devices on many application profiles, and they can still communicate with each other.

Chapter 3
Designing A Custom Profile

This chapter describes some issues to consider when designing a new custom-profile application, including selecting a profile ID, clusters, attributes and endpoints. It also describes ZigBee 2006 security options and includes a discussion on channel and bandwidth use.

3.1 Application Profiles

Application profiles are a collection of related services designed to be interoperable. In the case of public application profiles, the ZigBee Alliance specifies these services to allow for interoperability between OEM vendors' products. An Application Profile ID is a 16-bit number assigned by the ZigBee Alliance. Private Application Profile IDs must also be obtained from the ZigBee Alliance. The Freescale Private Profile ID 0xc021 is used for the example code provided by Freescale.

ZigBee Alliance public profiles include

- Home Automation
- Commercial Building Automation
- Industrial Plant Monitoring

Most profiles require the use of the ZigBee Cluster Library, a common library of services shared among profiles.

Every ZigBee node contains one or more application profiles. As an OEM, the only decision to make is whether to use a public ZigBee Alliance profile or a private profile. Private profiles have the advantage of being simple to implement and flexible for the project. Public profiles have the advantage of being interoperable among vendors, but at the expense of extra code size and complexity.

Public profiles are given an ID by the ZigBee alliance, for example 0x0104 for Home Automation. Contact the ZigBee Alliance (http://www.zigbee.org) for a private profile ID.

One public profile is available in every ZigBee node: the ZigBee Device Profile (profile ID 0x0000). This profile provides common services to all ZigBee nodes.

As an OEM, users should choose a public profile ID that matches their particular application or request a private profile from the ZigBee Alliance.