UG235.09: Developing NCP Applications with Silicon Labs Connect v2.x

This chapter of the *Connect v2.x User's Guide* describes how to develop Network Co-processor (NCP) applications with Silicon Labs Connect. The Connect stack is delivered as part of the Silicon Labs Proprietary Flex SDK. The *Connect v2.x User's Guide* assumes that you have already installed the Simplicity Studio development environment and the Flex SDK, and that you are familiar with the basics of configuring, compiling, and flashing Connect-based applications. Refer to *UG235.01: Developing Code with Silicon Labs Connect v2.x* for an overview of the chapters in the *Connect v2.x User's Guide*.

KEY POINTS

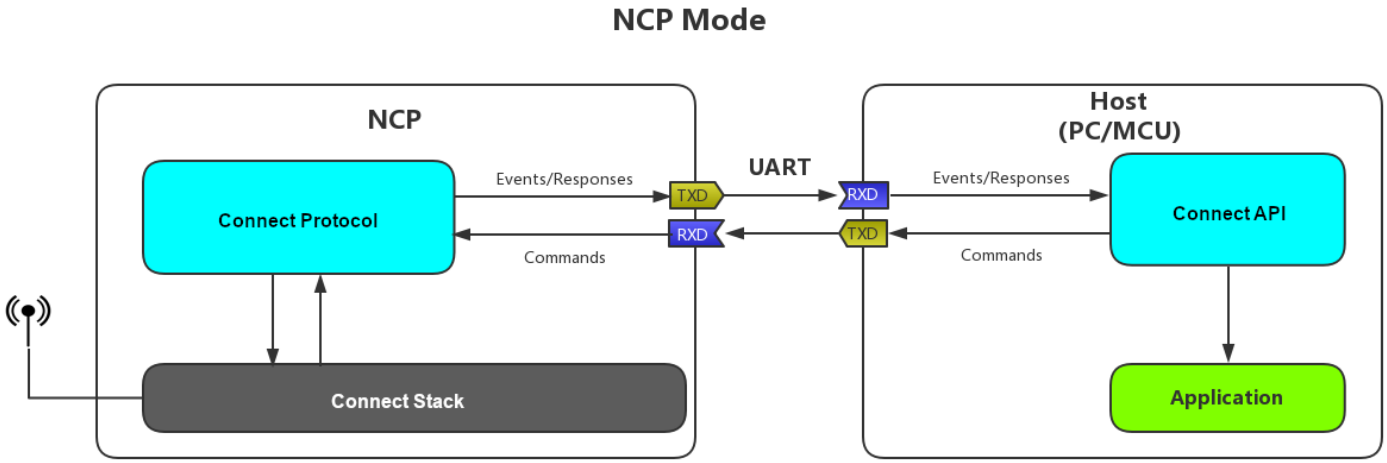
- Introduces the NCP.
- Discusses NCP and Host applications.
- Describes how to compile the two different types of applications.
- Explains the Connect Serial Protocol.

The *Connect v2.x User's Guide* is a series of documents that provides in-depth information for developers who are using the Silicon Labs Connect Stack for their application development. If you are new to Connect and the Proprietary Flex SDK, see *QSG138: Proprietary Flex SDK v2.x Quick Start Guide*.

Proprietary is supported on all EFR32FG devices. For others, check the device's data sheet under Ordering Information > Protocol Stack to see if Proprietary is supported. In Proprietary SDK version 2.7.n, Connect is not supported on EFR32xG22.

1. Introduction

NCP stands for Network Co-processor. By adding a Wireless Gecko (EFR32™) System on Chip (SoC) in NCP mode to their system, customers can implement a Connect-based wireless application that leverages the EFR32 Radio feature set. The original customer application (running on a Host device—PC, MCU) interfaces to and controls the NCP through the Universal Asynchronous Receiver-Transmitter (UART) interface as shown in the following figure.



Note: NCP should not be confused with virtual Network Co-processor (vNCP). For more information on vNCP, see *AN1153: Developing Connect vNCP Applications with Micrium OS*.

Messages sent from the Host to the NCP are known as **commands**. Messages sent from the NCP to the Host are known as **callbacks**.

To carry commands and responses between a Host processor and an NCP, Connect uses the Asynchronous Serial Host version 3 (ASHv3) protocol. The Connect NCP commands and responses are encapsulated into ASHv3 messages. For details on the ASHv3 protocol, see *UG115: ASHv3 Protocol Reference*.

The NCP platform can also be loaded with a correctly-configured serial bootloader (`bootloader-uart-xmodem`). For additional information regarding this bootloader approach, see. *UG235.06: Bootloading and OTA with Silicon Labs Connect*.

2. NCP and Host Applications

In the NCP approach, a typical Connect-based application is divided into two separate applications that each implement one of the distinct roles: NCP and Host. The NCP runs on the EFR32, while the Host application runs on the Host device.

2.1 NCP Application (SoC)

The NCP application runs on the EFR32 and supports communication with a Host application over a UART interface. The NCP application can be built as configured, or optionally can be augmented with customizations for target hardware, initialization, main loop processing, event definition/handling, and messaging with the Host. Silicon Labs provides two example NCP applications:

- project name: `ncp-uart-hw` uses hardware flow control.
- project name: `ncp-uart-sw` uses software flow control.

These NCP applications can be used out-of-the-box without any modification.

2.2 Host Application

The Host application can be compiled to almost any device with a suitable amount of memory and one free UART. The Host application examples are implemented for the Linux operating system. It is the customer's responsibility to port these to alternative systems as needed. Currently, Simplicity Studio does not support compilation for the Host architecture. Thus, compilation of the executable is only possible from outside of the Simplicity Studio environment (for example, from the command line). All Connect applications which are provided as standalone SoC applications are also available as NCP + Host applications.

Physical layer (PHY) parameters can be set only at compile time and only in the NCP application. Radio parameters are not configurable on the Host side.

3. Compiling NCP and Host Applications

3.1 Compiling the NCP Application

The procedure for compiling the NCP application is identical to that for any other Connect-based EFR32 application. Use Application Builder (AppBuilder) if any configuration of the "stock" examples is necessary (radio parameters, plugins, events, etc.).

3.2 Compiling the Host Application

Similar to the NCP case, use AppBuilder if any configuration of the Host application is necessary (plugins, events, etc.), then generate the project files. From this point, there are two ways to compile the application:

- Cross-compile on the computer where Simplicity Studio is installed, using the target toolchain, and deploy to the target device (requires a cross-compiler toolchain for the Host device).
- Compile on the target device using a native compiler (requires a copy of the SDK on the target device). (See the Note below.)

To compile the application, open a terminal, navigate to the project's root directory, and issue the `make -f <project-name>.mak` command. The project should be compiled cleanly and an executable named `<project-name>-unix-host-app` should be created in the directory `build-<project-name>-unix-host`.

Note: Copy `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<SDK-version>` to the target device. Take care (and you may need to edit the make file) to match the directory levels expected by the make file to those on the target device.

There is a special case: when Simplicity Studio is available for the target device (for example, the target device is an x86 PC with a Linux operating system), no further action is needed. Just compile the project as you would any other target application.

The Connect stack API available to a Host application is nearly identical to the API in an NCP application, with the exception of a few calls that make sense only on one or the other. Functions are invoked with the same arguments and generate the same return values, the same callbacks and events govern stack behavior, etc. Therefore, when developing an NCP mode system, interaction with the Connect stack API by the Host application is largely identical to the development experience for an NCP application.

4. Connect Serial Protocol

The Connect Serial Protocol (CSP) is used by a Host application processor to interact with the Connect stack running on an NCP. CSP messages are sent between the Host and the NCP over a UART interface. CSP is a binary protocol encapsulated within the ASHv3 protocol. For details on the ASHv3 protocol, see *UG115: ASHv3 Protocol Reference*.

Every API call from the application translates into a CSP message. The file `csp-host.c` contains the Host API implementation that converts API calls into serial commands and incoming serial commands into calls to the application callback functions. There is also a table mapping serial command IDs to handlers. All API calls are referenced by an identifier that must match on both the Host and the NCP side. The file `csp-enum.h` contains these identifiers.

The `csp-ncp.c` is the NCP code that corresponds to `csp-host.c`. For every API in `csp-host.c`, a corresponding handler exists in `csp-ncp.c`; for every handler in `csp-host.c`, an API exists in `csp-ncp.c` that sends the callback arguments.

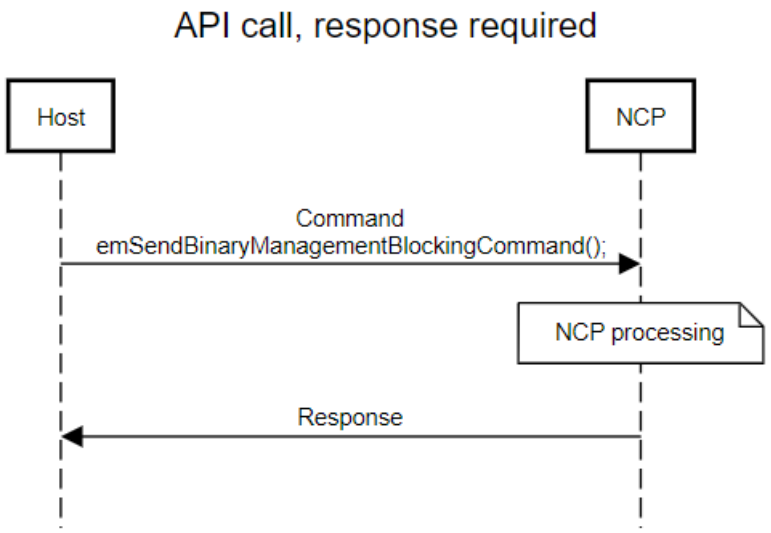
The file `csp-enum-decode.c` contains the code to translate command IDs into strings.

Note: Bear in mind that the files `csp-host.c`, `csp-ncp.c`, `csp-enum-decode.c`, and `csp-enum.h` are automatically generated so they are subject to change without notice.

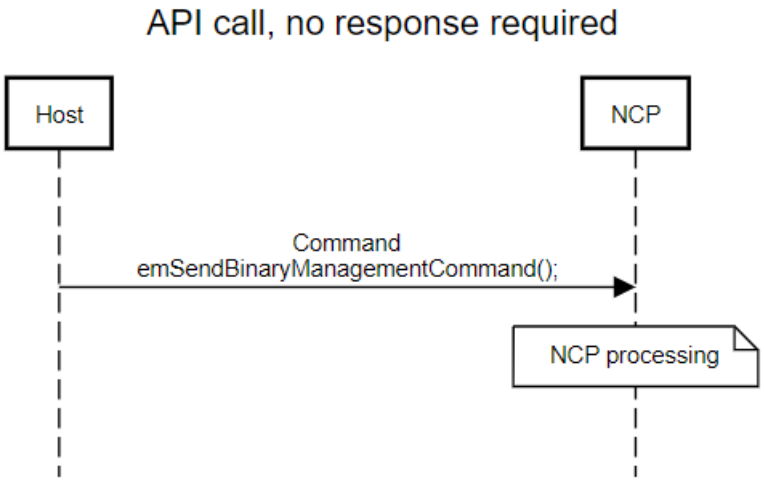
In general, the CSP works as follows:

1. An API call at the Host generates a serial message that gets sent to the NCP.
2. The Host spins waiting for a response from the NCP.
3. The NCP, upon receiving a message from the Host, decodes the message and invokes the actual stack API.
4. The NCP gets a return status from the API, packages it into a response, and sends it to the Host.
5. The Host, which was waiting for the response, gets the response, which is the return value of the API call.
6. Finally, the stack API call at the Host returns the actual return value.

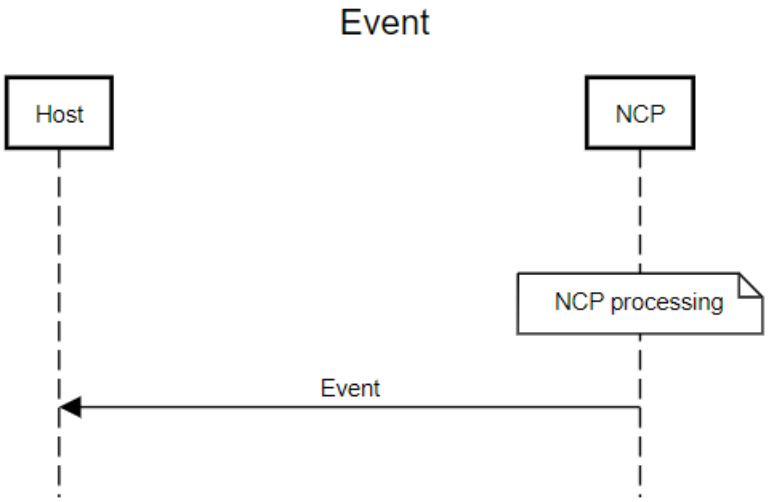
The following figure illustrates the typical CSP data flow.



There are also API calls that do not require a response. In these cases, the Host sends a command and does not wait for a response as shown in the following figure.



The third case occurs when an event is generated on the NCP side. In this case, the NCP sends the message and the corresponding callback function is invoked on the Host side as shown in the following figure.



4.1 CSP Frame Format

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	(Bytes > 5)
Start of frame	UART link type	Length		Management type	payload	

- Start of frame: 0x5A / '['
- UART link type (management / non-management—currently only management (0x01) is in use)
- Message length stored in 16 bits
- Management type (see `ncp-link.h`)
- Payload (for details, see section 4.2 CSP Payload Format)

4.2 CSP Payload Format

Byte 0	Byte 1	Byte 2	(Bytes > 2)
Command identifier		Arguments	

The payload of the CSP message consists of a 16-bit command identifier and the packed arguments (see `binary-management.h`).

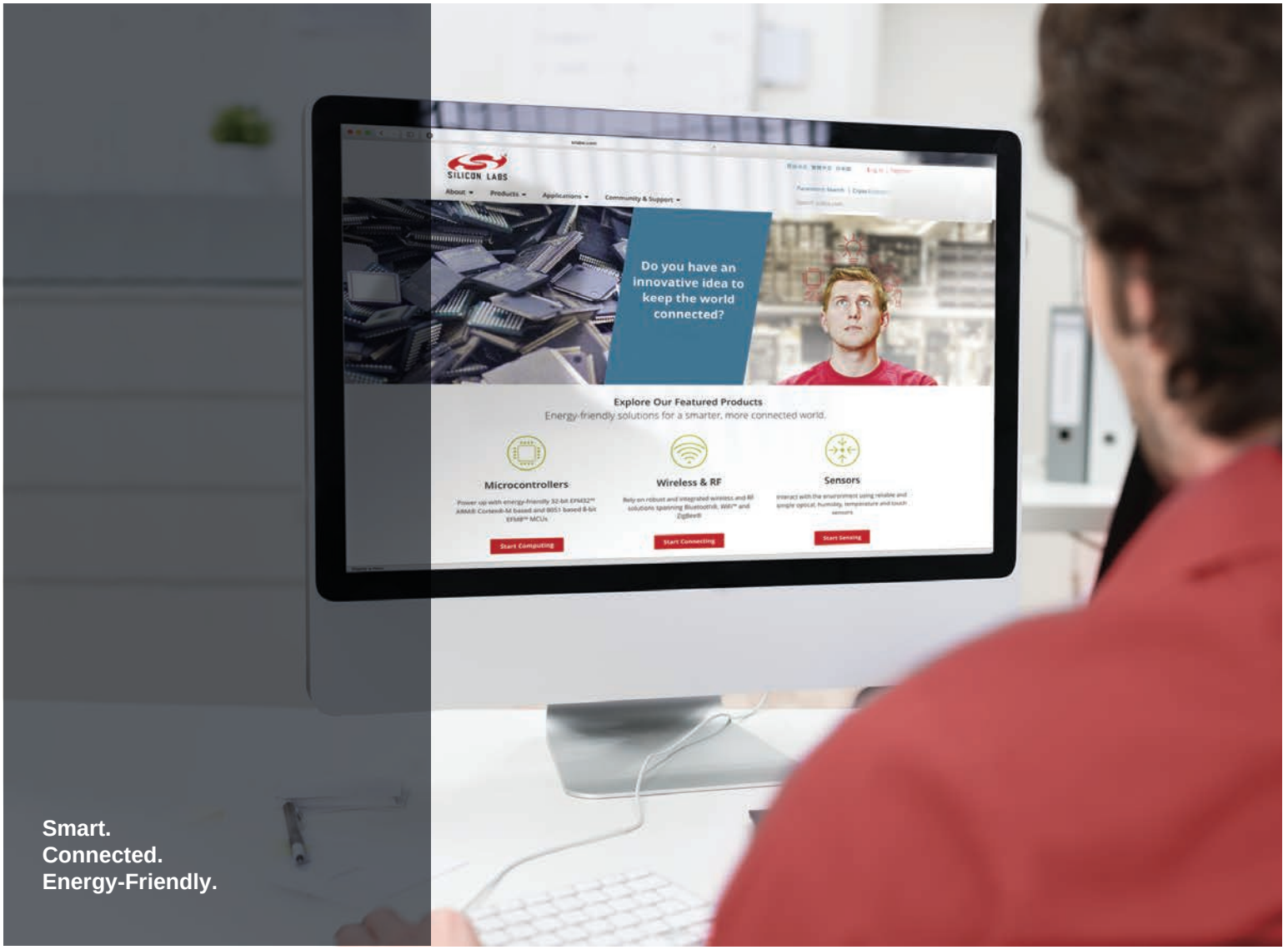
The `formatBinaryManagementCommand()` function packs the arguments into the CSP buffer and returns with the amount of space required by the payload.

5. Files to Port to Use NCP on a Specific Architecture / Platform

Most of the files in a Host application can be used as is. Some functions, however, need to be ported to the target system:

- `halCommonGetInt32uMillisecondTick()` (in `system-timer.c`)
- `emberSerialReadByte()` (in `simple-linux-serial.c`)
- `putchar()`

Porting the above functions should result in a project that can be compiled for the target Host.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com