ECHELON®

Neuron® Field Compiler User's Guide

078-0454-01A

Welcome

Echelon's Neuron® C programming language allows you to develop LONWORKS® applications for Neuron Chips and Smart Transceivers. The Neuron Field Compiler 4.0 software is a Neuron C compiler toolchain that you can use to develop a field programming tool that generates applications for Echelon Series 5000 and Series 3100 Smart Transceivers and Neuron Chips. The Neuron Field Compiler 4.0 software includes an application that accepts a Neuron C source file and generates a downloadable Neuron image. The Neuron image can be used by a network management tool to download the application over a LONWORKS network.

Audience

Echelon expects the Neuron Field Compiler to be used primarily by two different types of audiences:

- Developers of field programming tools that are used to generate applications for devices incorporating a Smart Transceiver or Neuron Chip.
- End users of a field programming tool, which uses the Neuron Field Compiler in a way that is generally transparent to the end user.

This document addresses the first audience (field programming tool developers), and thus assumes that you have an expert understanding of the Neuron C programming language, LONWORKS device development, and one of the Echelon Neuron C development platforms (the NodeBuilder® FX Development Tool or the Mini FX Evaluation Kit). In addition, a good understanding of either the Series 3100 or Series 5000 Smart Transceiver architecture is required.

Related Documentation

The following manuals are available from the Echelon Web site (www.echelon.com) and provide additional information that can help you develop applications for Neuron Chip or Smart Transceiver devices:

- *FT 3120 / FT 3150 Smart Transceiver Data Book* (005-0139-01D). This manual provides detailed technical specifications on the electrical interfaces, mechanical interfaces, and operating environment characteristics for the FT 3120® and FT 3150® Smart Transceivers.
- *Introduction to the LONWORKS Platform* (078-0391-01B). This manual provides an introduction to the ISO/IEC 14908 (ANSI/CEA-709.1 and EN14908) Control Network Protocol, and provides a high-level introduction to LONWORKS networks and the tools and components that are used for developing, installing, operating, and maintaining them.
- *LONMARK® Application Layer Interoperability Guidelines.* This manual describes design guidelines for developing applications for open interoperable LONWORKS devices, and is available from the LONMARK Web site, www.lonmark.org.

- *Mini FX User's Guide* (078-0398-01A). This manual describes how to use the Mini FX Evaluation Kit. You can use the Mini kit to develop a prototype or production control system that requires networking, or to evaluate the development of applications for such control networks using the LONWORKS platform.

- *Neuron C Programmer's Guide* (078-0002-02H). This manual describes the key concepts of programming using the Neuron C programming language and describes how to develop a LONWORKS application.

- *Neuron C Reference Guide* (078-0140-02F). This manual provides reference information for writing programs that use the Neuron C language.

- *Neuron Tools Errors Guide* (078-0402-01C). This manual describes error codes issued by the Neuron C compiler and related development tools.

- *NodeBuilder® FX User's Guide* (078-0405-01A). This manual describes how to develop a LONWORKS device using the NodeBuilder tool.

- *NodeBuilder Resource Editor User's Guide Release 4* (078-0194-01C). This manual describes LONMARK resource files and how to use the NodeBuilder Resource Editor to view, create, and modify them.

- *NodeLoad Utility User's Guide* (078-0286-01F). This manual describes the NodeLoad Utility, which lets you download transceiver parameters or application software into Echelon's Free Topology 3120 and 3150 Smart Transceivers and Power Line 3120, 3150, and 3170™ Smart Transceivers, even after they have been soldered into a device.

- *PL 3120 / PL 3150 / PL 3170 Power Line Smart Transceiver Data Book* (005-0193-01B). This manual provides detailed technical specifications on the electrical interfaces, mechanical interfaces, and operating environment characteristics for the PL 3120, PL 3150, and PL 3170 Smart Transceivers.

- *Series 5000 Chip Data Book* (005-0199-01B). This manual provides detailed specifications on the electrical interfaces, mechanical interfaces, and operating environment characteristics for the FT 5000 Smart Transceiver and Neuron 5000 Processor.

All of the Echelon documentation is available in Adobe® PDF format. To view the PDF files, you must have a current version of the Adobe Reader®, which you can download from Adobe at: get.adobe.com/reader.

Table of Contents

1

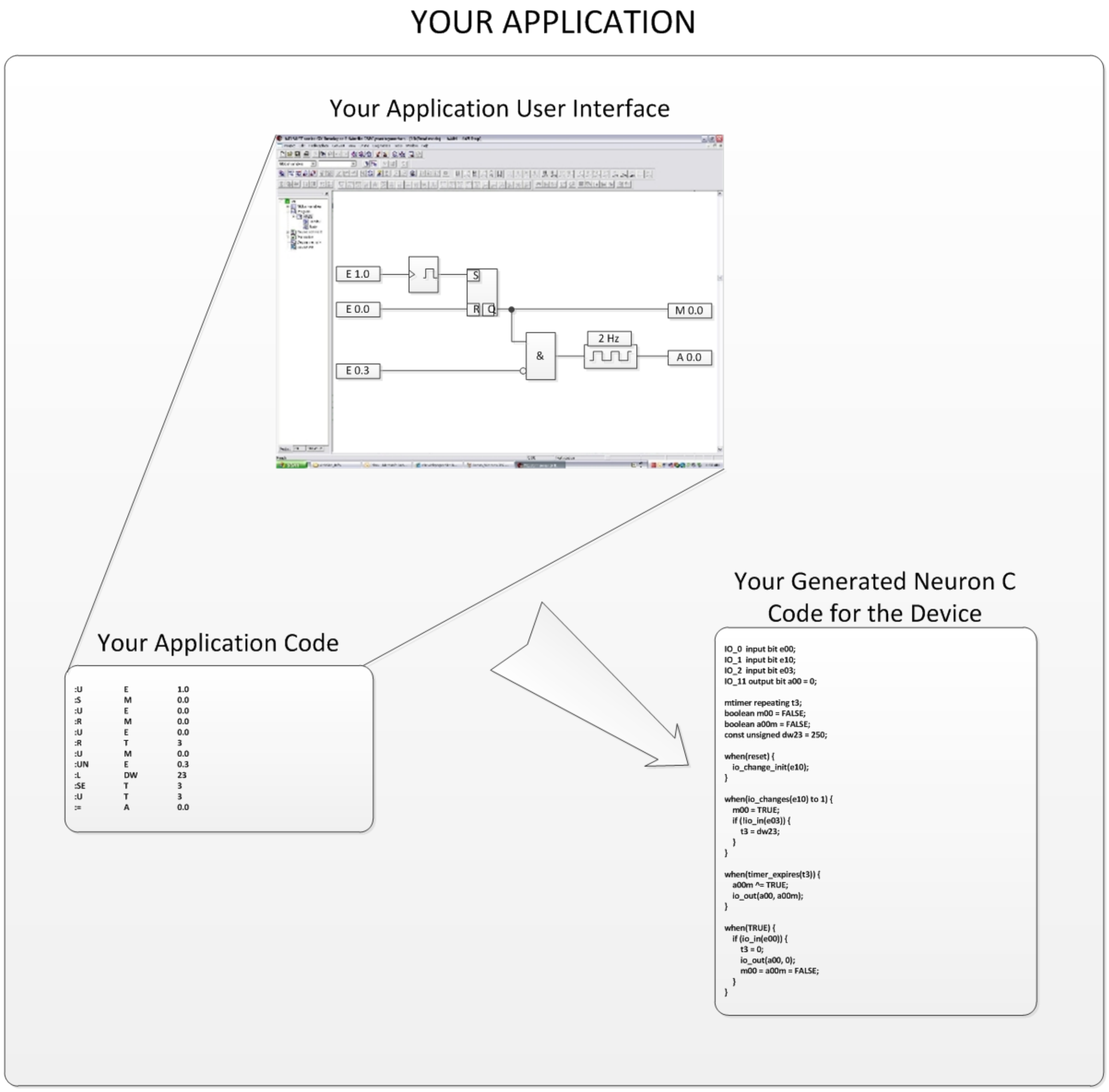
Introduction

This chapter introduces the Neuron Field Compiler, describes how to install it, and describes how to redistribute it.

Introduction

You can use the Neuron Field Compiler to develop a field programming tool for LONWORKS devices based on Echelon Series 5000 or Series 3100 Smart Transceivers or Neuron Chips. A field programming tool typically provides a programming environment that is suitable for use by network integrators who create special-purpose applications for controllers at the time that the controller is installed and commissioned in a network. Your field programming tool would include an application that translates the field application representation created by the user to a Neuron C source file, and then uses the Neuron Field Compiler to generate a downloadable image from that source file.

Figure 1 shows the field programming process.



YOUR APPLICATION
Your Application User Interface
E 1.0
E 0.0
E 0.3
S
R
Q
&
2 Hz
M 0.0
A 0.0
Your Application Code
Your Generated Neuron C Code for the Device

```
:U      E     1.0
:S      M     0.0
:U      E     0.0
:R      M     0.0
:U      E     0.0
:R      T     3
:U      M     0.0
:UN     E     0.3
:L      DW    23
:SE     T     3
:U      T     3
:=      A     0.0
```

```
IO_0  input bit e00;
IO_1  input bit e10;
IO_2  input bit e03;
IO_11 output bit a00 = 0;

mtimer repeating t3;
boolean m00 = FALSE;
boolean a00m = FALSE;
const unsigned dw23 = 250;

when(reset) {
   io_change_init(e10);
}

when(io_changes(e10) to 1) {
   m00 = TRUE;
   if (!io_in(e03)) {
      t3 = dw23;
   }
}

when(timer_expires(t3)) {
   a00m ^= TRUE;
   io_out(a00, a00m);
}

when(TRUE) {
   if (io_in(e00)) {
      t3 = 0;
      io_out(a00, 0);
      m00 = a00m = FALSE;
   }
}
```

Figure 1. Your Application Generates Neuron C Code

After your application converts your programming constructs into Neuron C source code, it can call the Neuron Field Compiler to compile static or dynamically generated Neuron C source code. The Neuron Field Compiler generates a downloadable application image and interface files for the Neuron Chip or Smart Transceiver.

Thus, your application presents its own programming interface for LONWORKS device functionality and generates an internal representation of that functionality in the Neuron C language. However, your application users do not need to know the Neuron C language, or even that Neuron C code is generated. In addition, your application does not need to be able to construct downloadable application image files for Neuron Chips or Smart Transceivers, but instead can rely on the Neuron Field Compiler to generate them from the Neuron C code.

Getting Started with the Neuron Field Compiler

The following sections describe the requirements for using the Neuron Field Compiler software, downloading it, and installing it.

You use the NodeBuilder FX Development Tool or the Mini FX Evaluation Kit to develop the Neuron C constructs that your field programming tool will use, and you can use the NodeBuilder FX Development Tool or the Mini FX Evaluation Kit to test the generated Neuron C code. In addition, you use the NodeBuilder FX Development Tool or the Mini FX Evaluation Kit to create additional data files, such as descriptions of the target hardware configurations (hardware template files) and definitions of user-defined device resource types.

You use the Neuron Field Compiler with your Neuron C generation tool to compile the generated Neuron C code and produce the application image files.

Hardware and Software Requirements

To install and use the Neuron Field Compiler software, your computer must meet the following minimum requirements, in addition to those imposed by your operating system:

- 512 MB RAM (or the operating system minimum requirement)
- Microsoft Windows® 7 (32-bit or 64-bit), Windows Vista® (32-bit), or Windows XP SP3
- 50 MB of available hard-disk space
- 1024x768 screen resolution

To develop applications that use the Neuron Field Compiler, you must also have the NodeBuilder FX Development Tool or the Mini FX Evaluation Kit installed. These products provide documentation and tools to help you create required additional data files. The NodeBuilder FX Development Tool also allows you to debug your generated Neuron C code.

Your end users do not need either the NodeBuilder FX Development Tool or the Mini FX Evaluation Kit installed to run the Neuron Field Compiler.

Recommendation: Test your application with the Neuron Field Compiler runtime on a clean machine that does not have either the NodeBuilder FX Development Tool or the Mini FX Evaluation Kit installed.

Installing the Neuron Field Compiler

To install and use the Neuron Field Compiler on a development computer, perform the following steps:

1. Download the **ReadMe.htm**, **FieldCompiler400.exe**, and **LonMarkResourceFiles1300.exe** files. After licensing the Neuron Field Compiler product, you will be provided with a download link for these files.
2. After reviewing the ReadMe document, double-click the **FieldCompiler400.exe** file to begin the Neuron Field Compiler installation. The Neuron Field Compiler main installer window opens.
3. Follow the installation dialogs to install the Neuron Field Compiler onto your computer.
4. Double-click the **LonMarkResourceFiles1300.exe** file to begin the LONMARK® resource files installation. The LONMARK Resource Files installer main installer window opens.
5. Follow the installation dialogs to install the LONMARK resource files onto your computer.

The Neuron Field Compiler installer installs the Neuron Field Compiler runtime components, including the command-line interface and the compiler adapter interface. As part of the installation, the Neuron Field Compiler installer creates the LONWORKS installation folder, **\Program Files\LonWorks**.

The LONMARK Resource Files installer installs the LONMARK resource files, which are required for Neuron C application development.

You can incorporate the Neuron Field Compiler installer into your application's installation, either as a standalone component that your end-users will install, or as a component that your overall software installer will install.

Review the Neuron Field Compiler Release 4.0 ReadMe document before running the Neuron Field Compiler installer, or developing your application with the Neuron Field Compiler.

Neuron Field Compiler Restrictions

The Neuron Field Compiler allows the same Neuron C language constructs and features as are supported by the NodeBuilder FX Development Tool and Mini FX Evaluation Kit. However, the Neuron Field Compiler does not support the following features:

- Creation of images for ShortStack® or Microprocessor Interface Program (MIP) devices. The **#pragma micro_interface** and **#pragma set_netvar_count** directives are not available.
- The direct memory file (DMF) access method, which is used by some types of ShortStack devices.
- Debug images.

Redistributing the Neuron Field Compiler

To enable your products to use the Neuron Field Compiler, you can redistribute the Neuron Field Compiler with your products' installation programs (for example, with the Flexera® InstallShield® installer).

You can download the installer for the Neuron Field Compiler, **FieldCompiler400.exe** and its required component, the LONMARK Resource Files installer, **LonMarkResourceFiles1300.exe**. After licensing the Neuron Field Compiler product, you will be provided with a download link for these files.

To embed the Neuron Field Compiler installation into your product installation, your installer must perform the following tasks:

1. Preset the LONWORKS path, if you want to change its default
2. Install the Neuron Field Compiler
3. Install the standard LONMARK resource files and any user-defined LONMARK resource files
4. Install hardware template files that are required for your programmable devices
5. Install other related prerequisites, as necessary

These tasks are described in the following sections.

Preset the LonWorks Path

The Neuron Field Compiler installer creates a LONWORKS installation folder, if it does not already exist. If you want to specify a different path than the default, your installer can define a **LONWORKS Path** entry in the Windows Registry. The default path setting is **[WindowsVolume]\Program Files\LonWorks**, where [WindowsVolume] is the drive where the Windows operating system resides.

The Neuron Field Compiler components (and other Echelon products) read the LONWORKS path information from the following Windows Registry string value:

```
HKEY_LOCAL_MACHINE\SOFTWARE\LonWorks\LonWorks Path
```

This string value determines the location of the main LONWORKS installation folder. The Neuron Field Compiler component files are installed in subfolders of this folder.

The LONWORKS path cannot be modified after any application is installed that uses the LONWORKS path. If the LONWORKS path key is changed after it is initially set, some or all of the Echelon software installed on your computer could malfunction.

For compatibility with all releases of Echelon products, the value of the LONWORKS Path entry should be a full path, including drive designation, and never end in a slash, "\".

Install the Neuron Field Compiler

The Neuron Field Compiler runtime installation can take a noticeable amount of time. To provide your end users with progress cues during installation, Echelon recommends that you install the Neuron Field Compiler runtime as a visible installer.

When you run it standalone, the Echelon Neuron Field Compiler installer also installs the LONMARK Resource Files installer as an embedded installation. When the Neuron Field Compiler installer runs as an embedded installation, it does not install the LONMARK resource files. Because the LONMARK resource files are required by the Neuron Field Compiler, your product installer must embed the LONMARK Resource Files installer, as described in the next section, *Install the LonMark Resource Files*.

The Neuron Field Compiler software product uses the Microsoft Windows Installer platform. It has an EXE wrapper that ensures that it installs with the Administrative user privileges that it requires. Use the following command to launch it with a basic user interface (shows only progress messages) from your installer:

```
FieldCompiler400.exe /s /v" /qb REBOOT=R"
```

This command suppresses user interface dialogs from the wrapper EXE (**/s**), but passes the command-line parameters between the quotes to the Windows Installer **msiexec.exe** through the **/v** parameter. These parameters tell the Windows Installer to run a basic user interface (**/qb**), and to set the **REBOOT** property to the value "R", which suppresses any reboots that might be needed by this installation, so that a forced reboot will not disturb the main installation. If you specify no user interface (**/qn**) instead of the basic user interface, the installation is completely silent.

Install the LonMark Resource Files

Neuron C code generated by your application, and compiled into application images using the Neuron Field Compiler, requires that the standard LONMARK resource files be installed on the computer that will deploy the built images.

You can find the LONMARK Resource Files installation on the LONMARK International Web site, www.lonmark.org/technical_resources/resource_files/. If you are licensed to redistribute the Neuron Field Compiler runtime with your application, you are also licensed to redistribute the LONMARK resource files.

You can install the LONMARK resource files with a visible installer to provide your end users with progress cues during installation.

The LONMARK Resource Files software product uses the Microsoft Windows Installer platform. It has an EXE wrapper that ensures that it installs with the Administrative user privileges that it requires. Use the following command to launch it with a basic user interface (shows only progress messages) from your installer:

```
LonMarkResourceFiles1300.exe /s /v" /qb REBOOT=R
DEVELOPER=1"
```

This command is similar to the Neuron Field Compiler installation command line, but additionally sets the **DEVELOPER** property to 1. This property ensures that the additional files that are required for a development environment, including the LONMARK resource include files, are installed as an optional feature.

Your application might also require one or more user-defined sets of resource files. Your installer must install these resource files to a suitable location, and register them with the resource file catalog. To register a resource file from a build script, change the current directory to the LONWORKS **\Types** folder and enter the following command:

```
mkcat –a<ResourceFolderPath>
```

See the *NodeBuilder Resource Editor User's Guide Release 4* (or later) for more information about working with resource files and the resource file catalog.

Install the Hardware Template Files

One of the necessary components for the Neuron Field Compiler to be able to compile your generated Neuron C code is the hardware template files (***.NbHwt**) for the target devices. A hardware template file describes the target hardware, including chip type, memory map, clock rate, and so on. You must provide one hardware template file for each type of device hardware that your application supports. Your installer must include these hardware template files. You can install them to any suitable folder; ensure that when your application calls the Neuron Field Compiler, it passes the installed location of these hardware template files with the --**hardware** command switch, as described in *Command Usage* on page 12.

Install Other Prerequisites

Your application might require the installation of additional Echelon components, such as the OpenLDV™ runtime or the NodeLoad utility. See the appropriate product documentation for redistribution licenses and redistribution instructions.

2

Using the Neuron Field Compiler

This chapter describes how to use the Neuron Field Compiler.

Compiling a Neuron C Program

To compile Neuron C application code, the Neuron Field Compiler includes several components. **Figure 2** shows the basic compilation flow for how your Neuron C generator tool works with the Neuron Field Compiler.

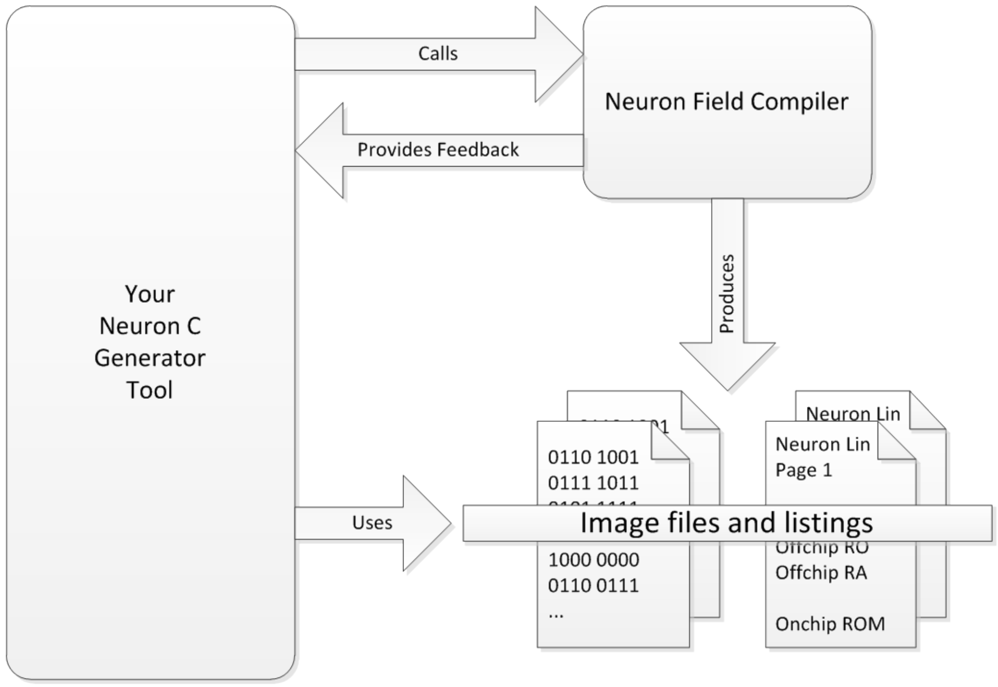


Figure 2. Basic Compilation Flow

As shown in the figure:

1. Your Neuron C generator tool calls the Neuron Field Compiler, passing in generated Neuron C code and the hardware template file for the target device.
2. The Neuron Field Compiler compiles the Neuron C code and provides compilation feedback, such as a success or error code.
3. The Neuron Field Compiler generates listing files and compiled image files.
4. Your Neuron C generator tool uses the generated listing and image files (such as an .**NME** or .**NDL** file) to load into the Smart Transceiver or Neuron Chip for a LONWORKS device.

The Neuron Field Compiler builds application image files as release images with default optimization preferences (you can control compilation optimization with compiler directives within the Neuron C source). Images are built as unconfigured and not authenticated. If your devices require shipment with configured and authenticated images, you must specify the related keys and settings during device production.

Figure 3 on page 11 shows the compilation flow within the Neuron Field Compiler to compile, assemble, and link a Neuron C source file.

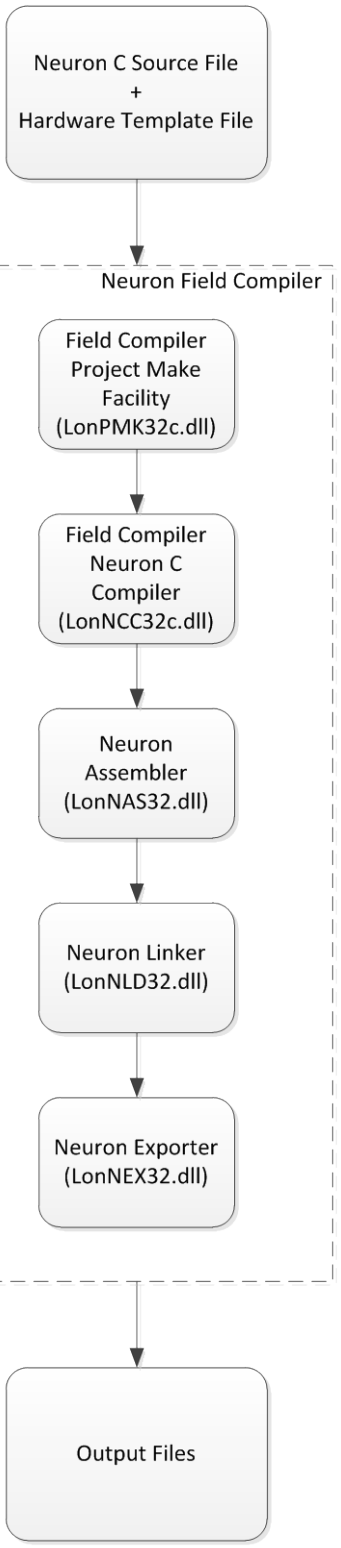

Neuron C Source File
+
Hardware Template File
Neuron Field Compiler
Field Compiler Project Make Facility (LonPMK32c.dll)
Field Compiler Neuron C Compiler (LonNCC32c.dll)
Neuron Assembler (LonNAS32.dll)
Neuron Linker (LonNLD32.dll)
Neuron Exporter (LonNEX32.dll)
Output Files

Figure 3. Compiling a Neuron C Source File

The components shown within the dashed line in **Figure 3** are part of the Neuron Field Compiler. Your Neuron C generator tool generally calls the Neuron Field Compiler, **LonNCA32**, rather than calling any of the components directly.

However, any of the Neuron Field Compiler components could potentially produce an error message that your Neuron C generator tool must be able to handle. See the *Neuron Tools Errors Guide* for information about the various error codes from the Neuron Field Compiler components. See *NCA Error Messages* on page 18 for a description of error codes issued by the **LonNCA32** command.

Note: The Neuron Field Compiler Project Make Facility (**LonPMK32c.dll**) issues the same error codes as the NodeBuilder Project Make Facility (**LonPMK32.dll**); likewise, the Neuron Field Compiler Neuron C Compiler (**LonNCC32c.dll**) issues the same error codes as the NodeBuilder Neuron C Compiler (**LonNCC32.dll**).

See the *Neuron C Programmer's Guide* for more information about the process of compiling a Neuron C program.

Command Usage

The command for running the Neuron Field Compiler is **LonNCA32**. You can issue this command from a Windows command prompt or you can call it from your Neuron C generator tool. The command accepts one parameter, the name of the Neuron C source file, and requires two command switches to specify the hardware template file and the program ID.

The following command usage notes apply to running the **LonNCA32** command:

- If no command switches or arguments follow the command name, the tool responds with usage hints and a list of available command switches.
- Most command switches come in two forms: A short form and a long form.

  The short form consists of a single, case-sensitive, character that identifies the command, and must be prefixed with a single forward slash '/' or a single dash '-'. Short command switches can be separated from their respective values with a single space or an equal sign. Short command switches do not require a separator; the value can follow the command identifier immediately.

  The long form consists of the verbose, case-sensitive, name of the command, and must be prefixed with a double dash '- -'. Long command switches require a separator, which can consist of a single space or an equal sign.

  Examples:
  Short form: `lonnca32 -dmymacro …`

  Long form: `lonnca32 --define mymacro …`
- Multiple command switches can be separated by a single space.
- Commands of a Boolean type need not be followed by a value. In this case, the value **yes** is assumed. Possible values for Boolean commands are **yes**, **on**, **1**, **+**, **no**, **off**, **0**, **-** (a minus sign or dash).

Examples:

```
lonnca32 --keep yes
lonnca32 --keep
```

- Commands can be read from the command line or from a command file (script file). A command file contains empty lines, lines starting with a semicolon (comment lines), or lines containing one command switch on each line (with value as applicable). The file extension can be any characters, but it is recommended that you use the "**.nca**" extension. For the command line, you must use quotation marks for strings that include spaces. However, do not include the quotation marks in a command file (spaces in strings are supported for command files).

  Example command file:

  ```
  ; Example command file for the Neuron Field Compiler

  --include d:\lm\Source\Demo\Example\Development
  --library d:\lm\Source\common\lib\myLibrary.lib
  --hardware d:\lm\Source\Demo\Example\myBoard.nbHwt
  --pid 9515310100000400
  ```

- Command switches can appear at any location within the command line or in any order (on separate lines) within a script.

Table 1 lists the available command switches for the **LonNCA32** command. Only the following switches are required for the command:

- --hardware (-h)
- --pid (-p)

Other command switches are optional.

Table 1. Command Switches for the **LonNCA32** Command

Command Switch		
Long Form	**Short Form**	**Description**
--define	-d	Define a valueless preprocessor symbol (macro).
--defloc		Location of an optional default command file.
--file	-@	Accept additional commands from a command file.
--hardware	-h	Specify the hardware template (**.NbHwt** file) for the device.
--help	-?	Display usage hints.
--include	-i	Add the specified folder to the include search path for **#include** statements.
--keep	-k	Keep generated intermediate files.

Command Switch		
Long Form	**Short Form**	**Description**
--library	-l	Specify a library (**.lib** file) to link with the application.
--mkscript		Generate a command script in a specified location.
--nodefaults		Disable processing of default command files.
--pid	-p	Use the specified program ID (in compact ASCII format or in colon-separated format).
--silent		Suppress banner message display.
--target	-t	Specify the name of the folder that contains the generated image files. This command switch also defines the target name within the device template (**.NbDt** file).
--warning		Display the specified message as a warning (used with --**mkscript**).
--which		Display path and version information about the specified service.

Notes:

- For the --**define** switch, the preprocessor symbols **_FIELD_COMPILER** and **_NEURONC** are already defined automatically, and do not need to be defined using this command switch.
- For the --**include** switch, specify a path for application-specific include files (specified as **#include "myFile.h"**). The default search-path for application-specific include files automatically includes the location of the main Neuron C source file. This command switch has no effect on system-specific include files (specified as **#include <stdlib.h>**).
- For the --**library** switch, you can use the **#pragma library** directive within your Neuron C source code, rather than this command switch, to specify additional libraries. The **#pragma library** directive provides support for location-independent library specifications and promotes self-documenting Neuron C source code.
- For the --**target** switch, you can specify the target ("Field Compiler" by default). The target defines the name of the folder that contains the generated image files (relative to the location of the main Neuron C source file), and is used within the device template (**.NbDt**) file.

When the Neuron Field Compiler prepares the build, it automatically creates NodeBuilder device templates, if one does not already exist. If it finds an existing device template, the Neuron Field Compiler modifies the existing template, as necessary.

To avoid target name conflicts with other Neuron C development tools, target names used with the Neuron Field Compiler must meet all of the following conditions:

- The name must have at least one character, and no more than 26.
- The first character must be in the ASCII range A-Z or a-z. Special characters are not supported.
- All other characters must be in the A-Z, a-z, or 0-9 range, be a space, or be one of the supported punctuation characters: - _ $ # ~
- At least one space or punctuation character must be included.

Recommendation: Set the target name to the Neuron Field Compiler's client application name or project name.

Calling the Neuron Field Compiler

Your Neuron C generator tool calls the Neuron Field Compiler (**LonNCA32**), parses the output from the compilation (see *Example Console Output* on page 17 for an example of the output generated by the Neuron Field Compiler), and then uses the generated image files to download and program a Smart Transceiver or Neuron Chip.

This section provides a simple example for calling the Neuron Field Compiler. This section does not describe how to parse the compilation output, but you could use a standard utility (such as **grep**) or string functions (such as the C **strcmp()** function). This section also does not describe how to download and program a Smart Transceiver or Neuron Chip, but you could use similar techniques to those shown below with the Echelon NodeLoad Utility; see the *NodeLoad Utility User's Guide*.

The following simple example is written in the C# language, and it allows you to call the Neuron Field Compiler and retrieve the compilation output without reading output files from the disk.

```
using System;
using System.Diagnostics;
using System.IO;

class MyProgram {
  static void Main() {
    //
    // Initialize the ProcessStartInfo class
    //
    string fieldCompiler = "LonNCA32.exe";
    string ncfile = " myFile.nc";
    string hwtemplate = " --hardware=myTemplate.NbHwt";
    string pid = " --pid=9515310100000400";

    ProcessStartInfo nca = new ProcessStartInfo();
    nca.FileName = fieldCompiler;
    nca.Arguments = hwtemplate + pid + ncfile;
    nca.UseShellExecute = false;
    nca.RedirectStandardOutput = true;
```

```
        //
        // Start the process
        //
        Process p = Process.Start(nca) {
          //
          // Read text from the process with StreamReader
          //
          StreamReader sr = p.StandardOutput {
            string compilerOutput = sr.ReadToEnd();
            // parse the compiler output here
            // or display it:
            Console.Write(compilerOutput);
          }
        }
      }
    }
```

Neuron Field Compiler Output

When you build your application, the Neuron Field Compiler creates *application image files* and *device interface files*. The downloadable application image files are used by network management tools to download the compiled application image to a device. Other application image files are used for in-circuit or ex-circuit programming of Neuron Chips or Smart Transceivers and non-volatile memory devices. The device interface file describes the external interface for your device. It is used by network tools such as the LonMaker® Integration tool to determine how to bind and configure your device.

The location of the output files and folders is relative to the location of the Neuron C source file. The Neuron Field Compiler creates a folder with the name of the target (see the description of the --**target** switch in *Command Usage* on page 12) in the folder that contains the Neuron C source file. This created folder contains the log file (**.log** extension), the device interface files (**.xif** and **.xfb** extensions), the linker map file (**.map** extension), and the application image files (various extensions depending on the target hardware, including **.apb**, **.ndl**, **.nei**, **.nfi**, **.nme**, **.nmf**, **.nxe**).

Important: The folder that contains the Neuron C source file must be writeable. That is, the Neuron Field Compiler process must have permission to create, modify, and delete files and folders in the location of the Neuron C source file.

The Neuron Field Compiler creates a number of hidden intermediate files that are generally not needed after the build is complete, but can be useful for program development and debugging. These files include the NodeBuilder project file (**.NbPrj**), device template files (**.NbDt**), and other intermediate files. These intermediate files are created within an **IM** subfolder that is deleted after compilation successfully completes. You can use the --**keep** switch to ensure that the **IM** subfolder and its contents are not deleted. To view these files within Windows Explorer, you must show hidden files: open **Folder Options** in the Windows Control Panel, select the View tab, and select the **Show hidden files, folders, and drives** radio button.

Recommendation: Keep the files created in the **IM** subfolder when you need to submit project files to Echelon Support for analysis. Your application can support the keeping of these files by providing an optional method for specifying the --**keep** switch when calling the Neuron Field Compiler.

Processing Error Messages

Any warning or error messages issued by the Neuron Field Compiler or its components have the following format:

Message-type: Model_file_name Line_number(Column_number): Message

Example: A model file named **tester.nc** includes the following single network variable declaration:

```
network input SNVT_volt nviVolt
```

Note the missing semicolon at the end of the line. When you use this file to build a project, the compiler issues the following message:

```
Error: TESTER.NC 1( 32):
Unexpected END-OF-FILE in source file [NCC#21]
```

The message type is Error, the line number is 1, the column number is 32 (which corresponds to the position of the error, in this case, the missing semicolon), and the compiler message number is NCC#21. To fix this error, add a semicolon to the end of the line.

See the *Neuron Tools Errors Guide* for information about the various error codes from the Neuron Field Compiler components. See *NCA Error Messages* on page 18 for a description of error codes issued by the **LonNCA32** command.

Example Console Output

The following example output from the **LonNCA32** command shows compilation of a simple Neuron C file that is linked for an FT 5000 Smart Transceiver.

```
Build log c:\src\test\mytarget-01\a.log
Created Thursday, January 13, 2011, at 15:12:00

Compiling...
Echelon Neuron C Compiler, version 5.01.7, build 0
Copyright (c) Echelon Corporation 1989-2010

Assembling...
Neuron Assembler, version 5.01.8, build 0
Copyright (c) Echelon Corporation 1992-2010

Linking...
Neuron Linker, version 5.01.8, build 0
Copyright (c) Echelon Corporation 1992-2010

                       LIST OF MEMORY AREAS
  * AREA *                    * ALLOCATED *          * UNUSED *
                             From  To   Len       From  To   Len
SYSTEM IMAGE ROM             0000 3FFF 4000        - RESERVED -
OffChip NVRAM  (EECODE)      4000 40FF 0100       402A 40FF 00D6
OffChip NVRAM  (Pool)         - AVAILABLE -       4100 DFFF 9F00
OffChip RAM    (Pool)         - AVAILABLE -       E000 E7FF 0800
OnChip  RAM    (System)      E800 ECC3 04C4          - NONE -
OnChip  RAM    (Pool)         - AVAILABLE -       ECC4 EFFB 0338
OnChip  RAMNEAR              EFFC EFFF 0004          - NONE -
OnChip  EEPROM (Prolog)      F000 F02D 002E          - NONE -
OnChip  EEPROM (Config)      F02E F0B6 0089          - NONE -
OnChip  EECODE               F0B7 F0B8 0002       F0B7 F0B8 0002
OnChip  EEPROM (MemCtl)      F0B9 F0C6 000E          - NONE -
OnChip  EEPROM (Pool)         - AVAILABLE -       F0C7 F7FE 0738
OnChip  EEPROM (Epilog)      F7FF F7FF 0001          - NONE -
End of Memory Area List
Link Memory Usage Statistics:
EEPROM Usage:  (not necessarily in order of physical layout)
               (includes application use of external NVRAM)
        System Data and Parameters          89 bytes
        Domain and Address Tables          105 bytes
        Network Variable Config Tables       6 bytes
        Application EEPROM Variables         0 bytes
        Library EEPROM Variables             0 bytes
        Application Code and Const Data     29 bytes
        Library Code and Const Data          0 bytes
        Self-Identification Data            11 bytes
                                         -----
        Total EEPROM Requirement           240 bytes
        Remaining EEPROM                 42768 bytes
RAM Usage:      (not necessarily in order of physical layout)
        System Data and Parameters         701 bytes
        Transaction Control Blocks          95 bytes
        Appl Timers and I/O Change Events    0 bytes
        Network and Application Buffers    424 bytes
        Application RAM Variables            4 bytes
        Library RAM Variables                0 bytes
                                         -----
        Total RAM Requirement             1224 bytes
        Remaining RAM                     2872 bytes
Successfully linked for FT 5000.
        Memory Map (9 bytes of System EEPROM) supports extended memory.
                This amount varies by the firmware version.
        Extended Version Number (4 bytes) varies by firmware version.
        System Upper EEPROM Area (1 byte) varies by firmware version.
        Priority outgoing message transaction control block uses 28 bytes of
RAM.
                Of this amount, 10 bytes varies by firmware version.
        Some system RAM usage, for System Data and Transaction Blocks
                (totaling 270 bytes) varies by the Neuron model, the
                firmware version, and the number of receive
                transaction control blocks.
        Default buffer counts vary by Neuron model
                (resulting in an additional 124 bytes of RAM).
        NVRAM Signature  = 0000 @ 4000 .. 4001
End of Link Statistics
Exporting...
Neuron Exporter, version 5.01.8, build 0
Copyright (c) Echelon Corporation 1992-2010

Creating 'c:\src\test\mytarget-01\a.NXE'... Done.
Creating 'c:\src\test\mytarget-01\a.XIF'... Done.
Creating 'c:\src\test\mytarget-01\a.XFB'... Done.
Creating 'c:\src\test\mytarget-01\a.APB'... Done.
Creating 'c:\src\test\mytarget-01\a.NDL'... Done.
Creating 'c:\src\test\mytarget-01\a.NME'... Done.
```

NCA Error Messages

Table 2 on page 19 lists the NCA error codes issued by the Neuron Field Compiler. See the *Neuron Tools Errors Guide* for information about the error codes from other Neuron Field Compiler components.

Table 2. NCA Error Codes

NCA#	Description
1	***The temporary folder <path> cannot be removed: <reason> [NCA#1]*** This error could occur if the temporary folder had previously existed but is write-protected. Other errors are also likely to occur in this situation. To analyze this error further, check the access permissions granted to the temporary folder and its parent folder.
2	***Cannot attach to the project make facility service: <detail> [NCA#2]*** The LonNCA32c Neuron Field Compiler driver cannot find or load the project make facility, **LonPMK32c.dll**. This error could result from an incorrect installation or a corrupt runtime environment. To analyze this error further, check the system search path. Ensure that the **LonPMK32c.dll** is included in this search path. This DLL and its companion DLLs should (by default) be installed in the LONWORKS **\Bin** folder. From the console, you can also try launching the project make facility stand-alone, using the **UCL32 lonPMK32c** command.
3	***Cannot initialize project make facility: <detail> [NCA#3]*** The project make facility can be loaded, but fails initialization. This error could result from an incorrect installation or from a corrupt runtime environment; see the discussion of the NCA#2 error for suggestions for further analysis.
4	***Cannot attach to <service>: Client already engaged with different service [NCA#4]*** Under normal conditions, this is an internal error, which should be reported to Echelon Support. During development and debugging of the Neuron Field Compiler client application, this error can occur as result of aborted and re-launched debug sessions. Closing and restarting the tool used for development and debugging of the Neuron Field Compiler client application should solve this problem.
5	***Cannot load service <name>: <detail> [NCA#5]*** This error is a generalization of the more specific errors NCA#2 and NCA#3.

NCA#	Description
6	***Cannot detach from service <name>: <detail> [NCA#6]*** Under normal conditions, this is an internal error, which should be reported to Echelon Support. During development and debugging of the Neuron Field Compiler client application, this error can occur as result of aborted debug sessions, especially when source code is edited and re-loaded during an active debugging session. Close and restart the tool used for development and debugging of the Neuron Field Compiler client application to solve this problem.
7	***Bad context: primary service is not attached [NCA#7]*** This is an internal error. Contact Echelon Support.
8	***Bad context: primary service is not initialized [NCA#8]*** This is an internal error. Contact Echelon Support.
9	***The temporary file <name> cannot be removed: <detail> [NCA#9]*** See NCA#1 for suggestions.
10	***Bad number of arguments. NCA requires exactly one argument, the path to the Neuron C source [NCA#10]*** The tool distinguishes between parameters (commands) and their values, and the single supported argument. This argument specifies the main Neuron C source file, and only one such argument is supported. This error might also arise if the main Neuron C source file is specified on the console, using a path containing one or more whitespace characters, but without using the quoted form.
11	***Cannot update file <name> (file is write-protected) [NCA#11]*** Check the permissions for the named file.
12	***Cannot update file <name> (system error) [NCA#12]*** Check the permissions for the named file.
13	***The Neuron C source file is not accessible <filename> [NCA#13]*** The Neuron C source file that is named cannot be found. Check the spelling of the filename and check the Application Directories' and Include Directories' search paths in the development tool's settings or the command line being used to execute the compiler.

ECHELON®

www.echelon.com