ECHELON

Neuron Tools Errors Guide

Warning and error message descriptions for Echelon development tools.

078-0402-01D

Echelon Corporation
www.echelon.com

Welcome

This document describes various warning and error messages that can occur when using Echelon® development tools, such as the IzoT™ NodeBuilder® FX Development Tool, , ShortStack® Developer's Kit, FTXL™ Developer's Kit, and LonTalk® Interface Developer (LID) utility.

Audience

This guide is intended for users of Echelon development tools.

Content

This guide describes the error messages that you can encounter while using Echelon development tools. This guide organizes the errors into separate chapters with each chapter containing errors applicable to a single software component. For example, compiler errors are in one chapter and IzoT NodeBuilder interface errors are in another chapter.

The error messages described in this document include hints, warnings, errors, and fatal errors:

- *Hints* are informative messages that suggest improvements for your implementation.
- *Warnings* are not severe enough to prevent successful compilation, but they should each be examined, and corrected as appropriate. Any code that is flagged by the compiler with a warning message should be viewed as a potential programming error, or at least as a poor programming practice.
- *Errors* result from code constructs that cannot be interpreted by the compiler, or indicate situations that are expressly prohibited by either the ANSI C language standard, or by the Neuron® C language. A compilation with one or more error messages results in build failure.

  If you need help resolving an error, contact LonSupport; see *For More Information and Technical Support* on page v.
- *Fatal Errors* prevent the compiler from performing any further translation. These messages result from resource problems (out of memory, disk full, and so on) or from internal checking on the compiler itself. Fatal errors might also appear in the form *****TRAP *n******, where *n* is a decimal number.

  Traps report unexpected internal errors, and should be reported to LonSupport; see *For More Information and Technical Support* on page v.

Each chapter lists the error messages in numerical order by message code. This ordering allows for possible future minor wording changes without interfering with your ability to locate the documentation for a particular message code. Each error message has a category acronym and a message code. The following message is typical:

The directive '#pragma num_alias_table_entries' is required [NCC#456]

For this message, **NCC** is the category acronym, **456** is the error code, and the text summarizes the error condition. **Table 1** lists the acronyms, what tool categories they represent, and the chapter of this manual that describes the error and warning messages for that category.

Table 1. Error Categories

Acronym	Category	Chapter
DBG	NodeBuilder Debugger	Chapter 1
DEP	Dependency Utility	Chapter 2
LCL	Communication Parameter Calculator	Chapter 3
LID	LonTalk Interface Developer Utility	Chapter 4
LWX	LONWORKS® XML module	Chapter 5
NAS	Neuron Assembler	Chapter 6
NCC	Neuron Compiler	Chapter 7
NEX	Neuron Exporter	Chapter 8
NLD	Neuron Linker	Chapter 9
NLB	Neuron Librarian	
PMK	NodeBuilder Project Make	Chapter 10
UCL	Common command line	Chapter 11
—	Neuron Firmware	Chapter 12

Related Documentation

The following manuals are available from the Echelon Web site (www.echelon.com/docs) and provide additional information about the tools mentioned in this manual:

- *FTXL User's Guide* (078-0363-01A). This manual describes how to develop an application for a LONWORKS device using Echelon's FTXL Transceiver. It describes the architecture of an FTXL device and how to develop the software for an FTXL device.
- *I/O Model Reference for Smart Transceivers and Neuron Chips* (078-0392-01C). This manual provides information about the I/O models used by Echelon's Neuron Chips and Smart Transceivers. It includes hardware and software considerations for each of the I/O models.
- *SmartServer Programming Tools User's Guide* (078-0349-01C). This manual describes how to write custom embedded applications called

Freely Programmable Modules (FPMs) and deploy them on the SmartServer. FPMs let you implement custom functionality and tailor the SmartServer to meet your needs.

- *Neuron C Programmer's Guide* (078-0002-02I). This manual describes how to write programs using the Neuron C Version 2.2 programming language.
- *Neuron C Reference Guide* (078-0140-02G). This manual provides reference information for writing programs using the Neuron C Version 2.2 programming language.
- *IzoT NodeBuilder FX User's Guide* (078-0516-01). This manual describes how to develop a LONWORKS device using the IzoT NodeBuilder tool.
- *ShortStack User's Guide* (078-0365-01B)**.** This manual describes how to develop an application for a LONWORKS device using Echelon's ShortStack FX Micro Server. It describes the architecture of a ShortStack device and how to develop a ShortStack device.

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: www.adobe.com/products/acrobat/readstep2.html.

For More Information and Technical Support

If you need help resolving an error, you observe an internal error, or you have technical questions that are not answered by this manual, you can contact Echelon technical support. To receive technical support from Echelon, you must purchase support services from Echelon or an Echelon support partner. See www.echelon.com/support for more information about Echelon support and training services.

You can also enroll in training classes at Echelon or an Echelon training center to learn more about developing devices. You can find additional information about device development training at www.echelon.com/training.

You can obtain technical support by telephone, fax, or e-mail from your closest Echelon support center, as listed in **Table 2** on page vi.

Table 2. Support Contact Information

Region	Languages Supported	Contact Information
The Americas	English Japanese	Echelon Corporation Attn. Customer Support 550 Meridian Avenue San Jose, CA 95126 Phone: +1-408-938-5200 Toll Free (US): 1-800-258-4LON (258-4566) Fax: +1-408-790-3801 lonsupport@echelon.com
Europe	English German French Italian	Echelon Europe Ltd. Suite 12 Building 6 Croxley Green Business Park Hatters Lane Watford Hertfordshire WD18 8YH United Kingdom Phone: +44 (0)1923 430200 Fax: +44 (0)1923 430300 lonsupport@echelon.co.uk
Japan	Japanese	Echelon Japan Holland Hills Mori Tower, 18F 5-11-2 Toranomon, Minato-ku Tokyo 105-0001 Japan Phone: +81-3-5733-3320 Fax: +81-3-5733-3321 lonsupport@echelon.co.jp
China	Chinese English	Echelon Greater China Rm. 1007-1008, IBM Tower Pacific Century Place 2A Gong Ti Bei Lu Chaoyang District Beijing 100027, China Phone: +86-10-6539-3750 Fax: +86-10-6539-3754 lonsupport@echelon.com.cn
Other Regions	English Japanese	Phone: +1-408-938-5200 Fax: +1-408-328-3801 lonsupport@echelon.com

Table of Contents

1

NodeBuilder Debugger Errors (DBG)

This chapter describes errors that can be reported by the NodeBuilder Debugger.

DBG Errors

Table 3 lists the DBG error codes.

Table 3. DBG Error Codes

DBG#	Description
2	***Memory allocation failed [DBG#2]*** An internal memory allocation operation failed. Save your work and restart the debugger.
3	***Cannot open a needed file [DBG#3]*** This error can occur if a debug information file or a source file could not be opened. Clean, rebuild, load and restart the debugger.
4	***Invalid DBT file format or DBT file is corrupt [DBG#4]*** Debug information file is invalid. Clean, rebuild, load and restart the debugger. If this persists, you may need to re-install IzoT NodeBuilder tool.
6	***Cannot communicate with the device over the network [DBG#6]*** Verify that your device is connected and any intervening routers are configured and online.
7	***Cannot find the requested symbol [DBG#7]*** Re-enter the correct symbol name. The DBT file might be out-of-date, and you may need to clean, rebuild, load and restart the debugger.
8	***Source line translates to less than 2 bytes of object code, so a breakpoint cannot be placed here [DBG#8]*** For Series 3100 chips, every instruction in the application has to be at least 2 bytes long in order to be able to set breakpoints successfully. Single-byte instructions need to be padded using the Expand Statements feature for the device template. Enable the Expand Statements feature and re-build if you need to set a breakpoint at this location. To enable the Expand Statements feature, right click the target, click Settings on the shortcut menu, then select the Compiler tab in the NodeBuilder Device Template Target Properties dialog. In the Debug Kernel Options box, select the Expand Statements check box, and then click OK. **Note**: This diagnostic does not apply to Series 5000 and Series 6000 Chips.

DBG#	Description
9	***Object code at breakpoint is not in writable memory, so a breakpoint cannot be set [DBG#9]*** NodeBuilder debugger modifies program memory when it sets breakpoints. Breakpoints can only be set in writeable memory. To debug code that is designed to execute from ROM (or other, non-writeable memory types), consider executing the same application on suitable development hardware which supports memory suitable for debugging.
10	***No instruction for BP at this location [DBG#10]*** This error can occur if the source file is not in sync with the application in the device. Verify that the build status is up-to-date. Try closing and re-opening the source file.
12	***Command is only legal when debugger is suspended [DBG#12]*** Some commands such as writing a value to an output network variable are available when the debugger suspended. Try the operation when the debugger is halted or suspended.
13	***Failed in file read/write [DBG#13]*** File operation failed on debug information file. Rebuild and load the application and try again.
16	***Debug kernel version not supported by debugger [DBG#16]*** The debug kernel loaded into the device is not supported by this version of the NodeBuilder debugger. Link your application with the correct debug library.
17	***Feature is not included in the version of the debug kernel included in the target device [DBG#17]*** Some features (for example, function execution) were excluded at compile time with various debug kernel options or #pragma debug <option> statements. The operation the debugger attempted will not work with the target's application. Right click the target, click Settings on the shortcut menu, then select the Compiler tab in the NodeBuilder Device Template Target Properties dialog. In the Debug Kernel Options box, select the required options, and then click OK.
19	***Bad parameter passed to method [DBG#19]*** Program error.

DBG#	Description
20	***Debug session start failed or not done yet [DBG#20]*** Exit and restart the program and try to debug the device again. Verify that you can communicate with the device being debugged.
21	***Command is invalid in the current dbgDebugStatus [DBG#21]*** The command being sent is invalid given the current state of the debugger. For example, you cannot halt a device which is currently in the DS_SUSPENDED state.
22	***Command invalid until pending command completes [DBG#22]*** Some commands cannot be sent while the debugger is still processing a previous request. For example, you cannot send a Resume command while a restart is pending. This error could occur because the device could be taking longer than expected to complete its reset processing. Check whether the device is operational.
23	***The target has been built since it was last loaded. Reload it and try again [DBG#23]*** The dependency checker keeps track of all source files, resources, hardware templates, and so on. The debug file (.DBT) must be kept in sync with the loaded application to allow the debugger to function. Clean, rebuild, load and restart the debugger.
24	***Device initialization failed or not done yet. [DBG#24]*** The debug file information could not be found, possibly because initialization did not complete successfully. Clean, rebuild, load and restart the debugger.
25	***The source file specified is not in the debugger's application image [DBG#25]*** The debugger tells the editor to load the top-level .nc file when it starts. In this case the debug file (.DBT) does not agree with the built image. Clean, rebuild, load and restart the debugger.
26	***The source file specified has been modified from the version in the debug application image. [DBG#26]*** A dependency check indicated that you need to re-build, load and start the debugger again. Clean, rebuild, load and restart the debugger.
27	***Maximum number of breakpoints already set [DBG#27]*** The debugger supports up to 100 concurrent breakpoints. Open the breakpoint list and delete some or all of the existing breakpoints.

DBG#	Description
28	***Communication with device timed out before a response was received*** A call made using function execution did not return or did not return valid data. Verify that the device is operational and communicating with the network. Rebuild and load the device and try again.
29	***Bad format found when reading [debug] NXE [DBG#29]*** The .NXE file might be corrupt; possibly it was hand-edited. Clean, rebuild, load and restart the debugger.
31	***Operation requested on symbol that is out of context, e.g. a local variable in a routine that is not in the current calling sequence [DBG#31]*** This error can happen if the operation attempted on a symbol is not permitted because the symbol is out of context. The symbol will be evaluated correctly when available. Using a local variable for example, the current value will be displayed when the debugger is suspended within the scope where the local variable is in context.
32	***Cannot read/write timers while running [DBG#32]*** The value of a timer variable cannot be evaluated while the debugger is running. You can see the correct value of a timer variable when the debugger is in suspended state.
33	***Cannot write this variable[DBG#33]*** An attempt was made to update a variable that is read-only or resides in read-only memory.
34	***No remote procedure calls while running [DBG#34]*** The debugger uses remote procedure calls or function execution to accomplish certain tasks while it is not running. Invoking function execution while running is not allowed.
35 **36**	***Not enough memory in debug kernel to pass RPC parameters [DBG#35]*** ***Not enough Neuron stack space left for RPC [DBG#36]*** A remote procedure call failed because of a lack of memory on the device. Rebuild and load the device and try again. You may need to reduce your application size in order to debug this device.

DBG#	Description
37	***Cannot add to watchlist, limit reached [DBG#37]*** You can concurrently view a maximum of 100 objects in the Watch List pane. Remove some or all of the entries in the Watch List pane and try again.
38	***Can only watch network variables, variables and timer object types [DBG#38]***
39	***Cannot debug a read/write protected device [DBG#39]*** The NodeBuilder debugger modifies program memory when it sets breakpoints. You cannot debug a device that has its read/write protect bit set. Clear the read/write protect bit in your application, re-build and load the device, and try again.
40	***Cannot refer to a memory area that is not in the Neuron memory map [DBG#40]*** Pointer type variables that reside in unmapped areas cannot be watched by the debugger.
41	***The debugger uses memory blocks to enable peek and poke operations. These blocks must completely reside within one memory area. Areas include: ROM, onchip EEPROM, offchip EEPROM, onchip RAM, offchip RAM and memory mapped I/O. Different memory areas have different read/write capabilities, etc. [DBG#41]***
42	***Peek/poke tried to access a code area that is not readable. [DBG#42]***
43	***There is no debug kernel on the device The debug kernel must be present in order to perform network debugging. Verify that the Use debug kernel option is set for the build target [DBG#43]***
44	***Cannot activate a breakpoint or a steppoint at the current location because it would overlap with a currently-active breakpoint [DBG#44]*** The NodeBuilder debugger executes single-step operations by setting implicit breakpoints at the next statement in each possible code path. If the NodeBuilder debugger finds user breakpoints at one of these statements that would overlap with an implicit breakpoint, this error will be encountered. Remove the user breakpoint and try the step operation again.
45	***Peek/poke operation attempted in a system area that is not readable. [DBG#45]***

DBG#	Description
47	***Variant data too large for provided buffer [DBG#47]*** Internal error. The size of a variable encountered was greater than the buffer provided for it.
49	***Failed to get IDispatch pointer [DBG#49]*** This is internal error in the COM subsystem.
50	***Device did not send a response message [DBG#50]*** The debugger sent a request that was not honored. Did the network buffers overflow? Was the device detached from the network?
51	***Device sent an unexpected response message [DBG#51]*** Are there two devices with the same subnet/device addresses?
52	***Response datapoint not returned [DBG#52]*** This error can occur because of communication problems with the device or problems with OpenLNS. Check that your device is functioning properly and that the network is not flooded. Verify that you have the latest versions of OpenLNS and the IzoT Commissioning tool installed on your computer.
53 **55**	***Request datapoint not returned [DBG#53]*** ***Output datapoint not returned [DBG#55]*** This error can occur because of problems with OpenLNS. Verify that you have the latest versions of OpenLNS and the IzoT Commissioning tool installed on your computer
56	***Debugger feature not yet implemented [DBG#56]***
57	***Debugger feature not implemented [DBG#57]***
58	***Failed to remove breakpoints while stopping the debugger [DBG#58]*** This error can occur if you modified the source file during a debug session. If you do not clear all breakpoints at the end of a debugging session, it can cause the device to not function properly. Clean, re-build, load and restart the debugger.
59	***Debug file is not available [DBG#59]*** The Debug file information was not loaded successfully during the debugger initialization. Verify that you have the correct debug settings selected for your device template. Clean, re-build, load and restart the debugger.

DBG#	Description
60	***Cannot debug this device because it is not responding. Please make sure that it is attached to the network and powered on [DBG#60]***
61	***Cannot debug this device because it has not been commissioned. Use LonMaker to commission it and try again [DBG#61]***
62	***Network interface must be configured to run VNI [DBG#62]*** Use the control panel to select a VNI network image. The name depends on which network interface you use. For the PCLTA-20 use: PCL10VNI. Do not use NSIPCLTA or PCC10L7.
63	***Device not found in network database [DBG#63]*** Remove the device from the NodeBuilder project tree view. You can then right-click the Devices folder, click Insert on the shortcut menu, and then select the device you want to debug. You may have to re-drop the target device shape with the IzoT Commissioning Tool.
64	***Cannot debug this device because it is unconfigured. Use the LonMaker Commission command to commission it and try again [DBG#64]***
65	***Cannot debug this device because it is applicationless. Use the LonMaker Load command to load the application image and try again [DBG#65]***
66	***Cannot debug this device because it is hard-offline. Use the LonMaker Device Manage command to put it online and try again [DBG#66]***
67	***Cannot debug this device because it is soft-offline Either reset it or use the LonMaker Device Manage command to put it online [DBG#67]***
68	***Failed to write device memory. Please try again [DBG#68]***
69	***Failed to read device memory. Please try again [DBG#69]***
70	***A device being debugged either stopped communicating or was deleted. The debug session will stop [DBG#70]***
71	***Device did not respond to Query Status command [DBG#71]*** Probably not a serious problem. The device may not have responded because of a lost packet.

2

Dependency Utility Errors (DEP)

This chapter lists and describes the errors that can be reported by the Dependency Utility component. The Dependency Utility is used by several build tools, including the compiler, assembler, linker, exporter, and project make; therefore, these errors could appear when using any of these tools.

DEP Errors

Table 4 lists the DEP error codes.

Table 4. DEP Error Codes

DEP#	Description
1	***An error occurred accessing file <file>: <reason> [DEP#1]*** A system error occurred when accessing a dependency file, see the error message for details provided as <reason>.
2	***An error occurred when processing dependency information: <reason> [DEP#2]*** A system error not related to file I/O occurred, see the error message for details provided in <reason>.
3	***An error occurred, but no details are available*** An unknown error occurred. You should try to clean and rebuild.
4	***malformed record '<tag>' in dependency file <file> [DEP#4]*** This error denotes a malformed dependency file. You should try to clean and rebuild.
5	***file <file> can't be referenced in dependency file (might cause build status calculation to become incorrect). (<reason>) [DEP#5]*** A file cannot be referenced when being added to a dependency file, or a non-recoverable problem occurs when investigating the file described by an existing dependency file record. This might be caused by the file being present but corrupt, or being locked by some other process. See <reason> provided in the message for failure details. You can ignore this message unless it persists; however, you should try to perform an unconditional rebuild because the missing data could cause the Project Make Facility to incorrectly determine the device's build status.
6	***missing separator in clause '<tag>' [DEP#6]*** The dependency file is missing a separator in a key/value pair. You should try to clean and rebuild.
7	***index <idx> is unsuitable for section <section> [DEP#7]*** A bad index value has been detected within the dependency file (see error message for section details). You should try to clean and rebuild.

3

Communication Parameter Calculator Errors (LCL)

This chapter lists and describes errors that can be reported by the communication parameter calculator.

LCL Errors

Table 5 lists the LCL error codes.

Table 5. LCL Error Codes

LCL#	Description
1	***Can not compute communication port control byte. [LCL#1]*** The tool failed to compute the communication port (CP) control byte. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. Re-install the product and choose the **Repair** option when prompted.
2	***Unrecognized encoded clock rate value <id> [LCL#2]*** An unrecognized encoded value was for clock speed (5 for 10MHz, 6 for 20MHz, and so on). Re-install the product and choose the **Repair** option when prompted.
3	***The transceiver's general purpose data record seems malformed: <gp data> [LCL#3]*** The xcvr_gp_data in STDXCVR.XML seems malformed. Verify that your standard transceiver database (stdxcvr.xml) has not been corrupted. Re-install the product and choose the **Repair** option when prompted.
4	***The device's clock rate (encoded value <id>) and the transceiver's communication rate (encoded value <id>) result in an invalid communication clock divider value. One of the two input rates might be invalid [LCL#4]*** The device's clock rate and the transceiver's communication rate result in an invalid communication clock divider value. One of the two input rates might be invalid. See the error message for failure details. Try increasing or lowering the Neuron clock speed.
5	***Unable to determine <attribute> using transceiver <xcvr name> [LCL#5]*** The aspect described as <attribute> cannot be computed as part of the communication parameter calculations. Verify that your standard transceiver database (stdxcvr.xml) has not been corrupted. Re-install the product and choose the **Repair** option when prompted.
6	***The transceiver requires a minimum clockrate which is higher than the device's configured input clock speed. [LCL#6]*** The target chip's system clock speed for the MAC context is lower than the required minimum for this transceiver. Choose a higher clock rate where possible, or choose a different transceiver type.

LCL#	Description
7	***The encoded value for the device's clock input of <id> is not within the supported range of <min> to <max> [LCL#7]*** An invalid encoded clock value has been used to describe the hardware clock speed. See LCL#2.
8	***The encoded value for the channel's minimum clock rate of <id> is not within the supported range of <min> to <max> [LCL#8]*** An invalid encoded clock value has been specified for the channel's minimum clock speed. This value originates from the standard transceiver database, and the presence of this problem indicates a corrupted or incorrect database record. Verify that your standard transceiver database (stdxcvr.xml) has not been corrupted. Re-install the product and choose the **Repair** option when prompted.
9	***Unable to compute media access control values (raw transceiver data): the specified Neuron clock frequency is too high for the transceiver [LCL#9]*** This error condition is unavoidable for some combinations of (typically) high Neuron clock rates and (typically) slow channels. A different operating mode of the same transceiver or a lower Neuron clock rate might solve the problem.

4

LonTalk Interface Developer Errors (LID)

This chapter lists the LonTalk Interface Developer utility error and warning messages that are applicable to the ShortStack Developer's Kit, FTXL Developer's Kit, and *i*.LON SmartServer LonTalk Interface Developer tool. This chapter offers suggestions for how to correct the indicated problems.

Overview

The LID error messages described in this chapter do not necessarily include the same wording that is shown at runtime. Instead, this chapter provides a summary of the message's meaning for each message, followed by a brief discussion of possible reasons and remedies. In all cases, be sure to consult the actual message reported by the utility at runtime, because the actual message is likely to contain additional details (for example, the name of the specific file for which the message is issued, or more details about the precise failure reason).

A numbering convention is used to identify the LID error messages as errors (1-3999), warnings (4000-7999), or hint codes (8000-9999). Not all messages are displayed in all contexts, but the numerical message identifier is always unique and unambiguous.

LID Errors

Table 6 lists the LID error codes.

Table 6. LID Error Codes

LID#	Description
1	***An NV, CP, or MT item was expected but not present – internal error*** Remove the device interface files (.xif and .xfb extension), and re-run the LonTalk Interface Developer utility to see if the problem persists. Use the **Trace** verbosity level to help track down the problem.
2	***A file cannot be opened for read access*** See the error message received for details of the offending file. Verify that the file is available and readable and the path is accessible.
3	***A file cannot be opened for write access*** See the error message received for details of the offending file. Verify that the file is available and writable and the path is accessible.
4	***A property value is required but has not been obtained from any data source*** This is an internal error, probably a result of an earlier failure. A non-fatal error during the creation of the device interface file might lead to this error. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure.
5	***An error occurred when reading a device interface file*** This is an internal error, probably a result of an earlier failure. A non-fatal error during the creation of the device interface file might lead to this error. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure.

LID#	Description
6	***An error occurred when reading a device interface file*** This is an internal error, probably a result of an earlier failure. A non-fatal error during the creation of the device interface file might lead to this error. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. (This error is similar to LID#5, but refers to a different internal component recognizing the error.)
7	***A device interface file appears malformed*** This is an internal error, probably a result of an earlier failure. A non-fatal error during the creation of the device interface file might lead to this error. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure.
8	***An unrecognized escape character has been detected in a file or NVVAL data record*** This is an internal error, probably a result of an earlier failure during the creation of an intermediate file with a .bif file extension. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. After the build, make sure the file with the .bif extension exists and can be read.
9	***A FILE or NVVAL value record cannot be read due to an unsupported construct*** This is an internal error, probably a result of an earlier failure during the creation of an intermediate file with a .bif file extension. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. After the build, make sure the file with the .bif extension exists and can be read.
10	***Failure to attach to LONUCL32 service DLL*** The LonTalk Interface Developer utility or one of its components failed to locate a file by name of "LONUCL32.DLL." This file usually resides in the same folder that contains the LID.exe application, but can be in any folder in your current user search path. This file is typically installed into the LonWorks Bin folder.
12	***Failure reading stdxcvr.xml file*** The standard transceiver definition file, **stdxcvr.xml**, cannot be found or cannot be read. The **stdxcvr.xml** file is usually installed into the LonWorks Types folder. Ensure that the file exists and can be read. This error code applies only to a ShortStack Micro Server.

LID#	Description
13	***Non-standard transceivers are not supported – the Micro Server uses one of these and no alternative xcvr has been explicitly specified*** This error can occur if the Micro Server does not specify a standard transceiver as the default, and no standard transceiver has been specified. Use a standard Micro Server and specify a standard transceiver to avoid this problem. This error code applies only to a ShortStack Micro Server.
14	***Standard transceiver cannot be found by ID*** The standard transceiver requested cannot be found in the standard transceiver definition file, **stdxcvr.xml**. Ensure that the **stdxcvr.xml** file is present and can be read. This error code applies only to a ShortStack Micro Server.
15	***Standard transceiver cannot be found by name*** The standard transceiver requested cannot be found in the standard transceiver definition file, **stdxcvr.xml**. Ensure that the **stdxcvr.xml** file is present and can be read.
16	***A field in the xcvr data appears to be corrupted*** The error message contains details about the field. The standard transceiver definition file, **stdxcvr.xml**, might be corrupt. Ensure that the **stdxcvr.xml** file is present and can be read. This error code applies only to a ShortStack Micro Server.
17	***The specified clock rate is too low for the specified transceiver*** The transceiver specified requires a higher clock speed; therefore, you cannot use the specified combination of clock speed and transceiver. Change either the Micro Server clock speed or use a different transceiver. This error code applies only to a ShortStack Micro Server.
18	***An error occurred when composing the application XIF file: the data merge target is ill-chosen (must be the BIF file)*** This is an internal error that should not normally occur. However, it could be a result of an earlier failure. For example, a non-fatal error during the creation of the device interface file might lead to this error. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure.
19	***File I/O error when writing XIF file*** Refer to the error message for details about the failure cause. The error message contains details such as “disk full,” or “file access denied”.

LID#	Description
20	***Error (non-file I/O) when writing XIF file*** Refer to the error message for details about the failure cause. The error message contains details such as "disk full," or "file access denied".
21	***The xif32bin.exe utility returned an error, indicating failure when converting XIF to XFB*** The binary device interface file (.xfb extension) could not be created. Verify that a previously existing binary device interface file is not write-protected. Also make sure the XIF32Bin.exe utility, which is used to create the binary device interface file, is available in a folder that is part of the system or current user search path. By default, the utility can be found in your LonWorks Bin folder.
22	***An error occurred when reading a type info file (.NCT)*** This is an internal error, possibly resulting from an earlier failure. The .nct file is an intermediate file used by the LonTalk Interface Developer utility. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure.
23	***An error occurred when reading a type info file (.NCT)*** This is an internal error, possibly resulting from an earlier failure. The .nct file is an intermediate file used by the LonTalk Interface Developer utility. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. This error is similar to LID#22, but refers to different internal software components.
24	***Type info (.NCT) file seems corrupted*** This is an internal error, possibly resulting from an earlier failure. The .nct file is an intermediate file used by the LonTalk Interface Developer utility. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure.
25	***Unexpected end of type info file (.NCT)*** This is an internal error, possibly resulting from an earlier failure. The .nct file is an intermediate file used by the LonTalk Interface Developer utility. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure.
26	***Specified target language does not match an implemented code generator*** Choose a different target host language.

LID#	Description
27	***Unexpected file I/O error when reading a file*** Refer to the error message for details of the failure cause.
28	***Unexpected error (not a file I/O error) when reading a file*** Refer to the error message for details of the failure cause.
29	***Unexpected file I/O error when writing a file*** Refer to the error message for details of the failure cause.
30	***Unexpected error (not a file I/O error) when writing a file*** Refer to the error message for details of the failure cause. The error message contains details such as "disk full" or "file access denied".
31	***A type definition cannot be generated: the type is referenced but not defined*** A type that you have referenced is missing from the NCT file, and intermediate file used by the LonTalk Interface Developer utility. This is an internal error. Delete all intermediate files. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. If the problem persists, contact Echelon technical support, submitting all files produced by the LonTalk Interface Developer utility when running in **Trace** verbosity level.
32	***A type definition is provided but seems incomplete -- an element is missing*** This is an internal error. Delete all intermediate files. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. If the problem persists, contact Echelon technical support, submitting all files produced by the LonTalk Interface Developer utility when running in **Trace** verbosity level.
33	***Anonymous types are not supported*** Any type used for network variables or configuration properties must have a name. The use of constructs such as, "network input struct { int a, b; } nviZorro;" is not permitted.
34	***A compiler feature cannot be selected*** Refer to the error message for details of the failure cause. This error might be the result of conflicting preferences in the default command file, LonNCC32.def, located in the LonTalk Interface Developer utility's project file. Refer to the *Neuron C Programmer's Guide* and *Neuron C Reference Guide* for more details about the command line tools and script files.

LID#	Description
35	***Configuration parameters are in use, but no template file has been found*** This might be the result of an earlier error. Delete all intermediate files. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. If the problem persists, contact Echelon technical support, submitting all files produced by the LonTalk Interface Developer utility when running in **Trace** verbosity level.
36	***The program ID found in the XIF file seems malformed and cannot be used to produce the niAppinit data*** Use the LonTalk Interface Developer utility and the Standard Program ID calculator to produce a good program ID record. Delete all intermediate files. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. If the problem persists, contact Echelon technical support, submitting all files produced by the LonTalk Interface Developer utility when running in **Trace** verbosity level.
37	***Malformed communication parameters*** This is an internal error, which only occurs as a result of an unrecognized previous error. This error code applies only to a ShortStack Micro Server.
38	***Cannot calculate communication parameters*** Refer to the error message for details of the failure cause. Delete all intermediate files from the LonTalk Interface Developer project directory. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. If the problem persists, contact Echelon technical support, submitting all files produced by the LonTalk Interface Developer utility when running in **Trace** verbosity level. This error code applies only to a ShortStack Micro Server.

LID#	Description
39	***Cannot locate Micro Server's symbol file*** Refer to the error message for the filename. A Micro Server has three groups of files: image files that get loaded into the Smart Transceiver (.nei, .nxe, or .nri extension), device interface files (.xif or .xfb extension), and symbol files (.sym extension). All of these files must share the same base name, and they must reside in the same location. This error describes a problem in locating the symbol file. The expected name and location is detailed in the error message. This error code applies only to a ShortStack Micro Server.
40	***Cannot find required symbol in Micro Server's symbol file*** Refer to the error message for the symbol name. The Micro Server's symbol table can be read but lacks a required symbol. Be sure to use a standard Micro Server. This error code applies only to a ShortStack Micro Server.
42	***A type definition cannot be generated -- the type definition has more elements than expected*** Delete all intermediate files. Re-run the LonTalk Interface Developer utility in **Trace** verbosity mode and carefully examine the LonTalk Interface Developer utility Summary window to determine the root cause of the failure. If the problem persists, contact Echelon technical support, submitting all files produced by the LonTalk Interface Developer utility when running in **Trace** verbosity level.
43	***Explicit addressing is requested by the user or required by the model file but is not supported by the Micro Server*** Choose a different Micro Server, or remove the need for explicit addressing. This error concerns "explicit addressing" and not "explicit messages." A Micro Server must explicitly support explicit addressing of explicit messages (unbound messages). Explicit messages that use implicit addressing (bound messages) can be used with any Micro Server. This error code applies only to a ShortStack Micro Server.
44	***A custom image is required for the chosen transceiver type*** The ShortStack Micro Server images included with the ShortStack Developer's Kit can only be used with FT and PL Smart Transceivers. This error code applies only to a ShortStack Micro Server.
45	***The selected Micro Server image cannot be used with a clock rate derived from 6.5536 MHz*** This error code applies only to a ShortStack Micro Server.

LID#	Description
46	***One or more configuration parameters implemented within a file are present, FTP or DMF must be implemented*** Alternatively, you can declare configuration properties as configuration network variables.
47	***The file transfer protocol (FTP) and direct memory files (DMF) access mechanisms are mutually exclusive***
48	***The chosen Micro Server is based on firmware version <v>. This version does not support the direct memory files*** This error code applies only to a ShortStack Micro Server.
49	***The FTP server interface is partially implemented, missing the specified member of the device object***
50	***Data files and file directory are too big for the available space. Available: <n> bytes, required: <m> bytes (missing: <p> bytes) [LID#50]*** Possible remedies: reduce the size of files by removing extraneous data files, or by sharing CP, or implement FTP.
51	***Malformed XML data (cannot convert to expected type)***
52	***The specified application framework type is unknown***
53	***No target framework has been supplied, or the requested framework is not registered with, or not known to, the Builder***
54	***No code generator found for the selected target framework***
57	***Required source file missing***
58	***The specified Micro Server image file cannot be copied into the project folder*** This error code applies only to a ShortStack Micro Server.
59	***Too many network variables. The sum of static and dynamic variables cannot exceed <s>.*** The message text specifies the value of *s*. In general, the total number of network variables cannot exceed 4096.

LID#	Description
60	***Insufficient number of addresses*** This message includes how many addresses are required for the application, and how many were specified. Whereas one or more network variables can share an address table entry (although such sharing might limit the versality of network variable connections), each bindable message tag requires its own one address table entry. The address table must provide at least one entry for each bindable message tag, plus at least one address table entry for all network variables implemented (if any). To avoid this error, you must allocate a larger address table, or declare fewer bindable message tags. Note that although message tags cannot share an address table entry, multiple application messages can share the same message tag (if they all communicate with the same target devices).
61	***The DMF window specification is invalid, as it exceeds the 64 KB address range***
62	***Insufficient buffer space*** The message includes the total number of bytes available for transceiver buffers and how many additional bytes your selected configuration requires. This error occurs because you specified that the LonTalk Interface Developer utility declare more buffers than the available memory space within the device. To avoid this error, supply or declare more RAM, or request fewer or smaller buffers.
63	***Malformed data: <d> is not a <t>*** Data *d* does not meet expectations *t*. For unmodified data, this error condition should not occur. Re-install the product, and choose the repair option when prompted.
64	***Failure loading neuron.xml*** The **neuron.xml** database, located in the Types folder within your local LonWorks folder, is a central repository of data that describes the various attributes and capabilities of Neuron Chips and Smart Transceivers. A failure to load this database indicates a fundamental error. Re-install the product, and choose the repair option when prompted.
65	***The chosen clocking scheme (ID <s>) doesn't support external clock <c> with a clock multiplier of <m>*** This error indicates inconsistent and incorrect clocking specifications. Verify your clocking-related preferences.

LID#	Description
66	***The <s> Micro Server does not support clock multiplier <m> in the chosen configuration*** This error can occur if you specify incorrect clocking details for the LonTalk Interface Builder (the command-line interface for the LonTalk Interface Developer utility). Check your preferences. This error code applies only to a ShortStack Micro Server.
67	***The <s> Micro Server does not support clock <c> in the chosen configuration*** This error can occur if you specify incorrect clocking details for the LonTalk Interface Builder (the command-line interface for the LonTalk Interface Developer utility). Check your preferences. This error code applies only to a ShortStack Micro Server.
68	***The <s> Micro Server does not support transceiver <t> in the chosen configuration*** This error can occur if you specify incorrect clocking details for the LonTalk Interface Builder (the command-line interface for the LonTalk Interface Developer utility). Check your preferences. This error code applies only to a ShortStack Micro Server.
69	***The <s> Micro Server is not supported by the Micro Server database*** This error can occur if you incorrectly specify the Micro Server for the LonTalk Interface Builder (the command-line interface for the LonTalk Interface Developer utility). Check your preferences. This error code applies only to a ShortStack Micro Server.
70	***Error loading the Micro Server database: <detail>*** This error occurs upon a particular failure when loading the Micro Server database, as detailed in the error message. Unless the error detail suggests a different remedy, you should re-install the product and choose the **Repair** option when prompted. Also, temporarily remove any user-defined extensions to the Micro Server database (that is, remove or rename the **UserServers.xml** file). This error code applies only to a ShortStack Micro Server.
71	***Clock multiplier <m> is not supported in this configuration*** This error can occur if you specify an incorrect set of arguments for the LonTalk Interface Builder command line interface (the command-line interface for the LonTalk Interface Developer utility). Check your arguments.

LID#	Description
72	***<f> external clock is not supported in this configuration*** This error can occur if you specify an incorrect set of arguments for the LonTalk Interface Builder command line interface (the command-line interface for the LonTalk Interface Developer utility). Check your arguments. Note that the **-clock** parameter refers to the external clock, not the system clock.
4001	***An XIF file contains more fields than expected*** Refer to the warning message for line # and filename. This might result in an automatic downgrading of the device interface file to the version supported by the FTXL or FTXL tools. Check www.echelon.com for available updates.
4002	***An intermediate file cannot be removed in the sweep-phase. See message for details*** Refer to the warning message for details about the warning cause. The sweep occurs when the utility's operation is complete and the utility did not run in the **Trace** verbosity level. The warning indicates that an intermediate file cannot be removed.
4004	***The Micro Server's default clock rate does not match the explicitly specified clock speed*** A Micro Server image supports a default clock rate. You might have to reconsider your clock rate selection. Otherwise, consult your Micro Server documentation for possible clock rate restrictions and make sure your chosen clock rate matches the hardware. This warning code applies only to a ShortStack Micro Server.
4006	***A file cannot be copied*** This is possibly, but not necessarily, fatal. When the LonTalk Interface Developer utility creates the host framework, it produces several files based on input provided by the user. It also copies the necessary files into the destination folder. The utility-generated files refer to these files, which are required to build the host application. Thus, this issue is non-fatal for the LonTalk Interface Developer utility, but probably fatal when building the host application. See also warning LID#4017.
4007	***One or more configuration properties implemented within a file are present – LW-FTP must be implemented*** Alternatively, you can declare configuration properties as configuration network variables. This warning code applies only to a ShortStack Micro Server.

LID#	Description
4009	***One or more configuration properties are present but the LW-FTP-related network variables are not*** This warning indicates that the LONWORKS file transfer protocol (LW-FTP) must be implemented (because of the presence of configuration properties that are held in files), but the absence of the LW-FTP-related network variables indicates that LW-FTP has not been implemented. Change the declaration of your configuration properties to configuration network variables (if possible), or complete the implementation of LW-FTP. See *www.echelon.com* for more information about the file transfer protocol (LW-FTP). This warning code applies only to a ShortStack Micro Server.
4011	***The .NCT file references a built-in type with no host equivalent known to LonTalk Interface Developer utility*** This condition is unlikely to occur and does report an internal error. Check *www.echelon.com* for available software updates that address the problem, or contact *LonSupport@Echelon.com*. This message is a warning rather than an error because the condition does not prevent your application from working. Carefully check the type definitions provided in **LonNvTypes.h** and **LonCpTypes.h** (both generated by LonTalk Interface Developer utility) and correct the offending type. Continue using these files and build your FTXL device.
4012	***The channel type specified in the standard program ID does not match the channel type served by the transceiver in use*** This message reminds you to correct the transceiver choice or the program ID. While the mismatch does not cause the device to malfunction, it breaks the interoperability of the device and might cause a network tool to prevent installation of the device. This warning code applies only to a ShortStack Micro Server.
4013	***Explicit addressing not requested but seems required*** The presence of message tags with the **bind_info(nobind)** modifier in the model file indicates that explicit addressing is probably required. Enable explicit addressing and re-generate the application framework using the LonTalk Interface Developer utility. See also the LID#4014 and LID#4015 warnings. This warning code applies only to a ShortStack Micro Server.

LID#	Description
4014	***Explicit addressing specified but not required*** This warning reminds you that you have requested support for explicit addressing, although it does not seem to be required. Explicit addressing requires larger buffers on the host, therefore support for explicit addressing is advisable only when needed. Message tag declarations that are intended for use with explicit addressing should be marked with the **bind_info(nobind)** modifier to signal the use of explicit messaging. See also the LID#4013 and LID#4015 warnings.
4015	***Explicit addressing specified but neither supported nor required*** Although support for explicit addressing has been requested, it does not appear to be required. See also the LID#4013 and LID#4014 warnings.
4016	***FTP implementation suspect -- no message tag but SNVT_file_* implemented*** The implementation of the file transfer protocol is suspect, as the FTP-related network variables are present but no message tag has been declared.
4017	***Files cannot be made writable*** When the LonTalk Interface Developer utility creates the host framework, it produces several files based on input provided by the user. It copies the necessary files into the destination folder. These files are made writable after they are copied, unless this warning indicates it is not possible. See also the LID#4006 warning.
4018	***The –extApi command is deprecated; use –queryapi and –updateapi instead*** This warning code applies only to a ShortStack Micro Server.
4019	***No image file for the specified Micro Server could be found*** This warning code applies only to a ShortStack Micro Server.
4020	***Existing Micro Server image not overwritten*** This warning code applies only to a ShortStack Micro Server.
4021	***Application message support has been enabled to meet the application's requirements*** This warning code applies only to a ShortStack Micro Server.
4022	***Application messaging support has been requested, but no message tag is declared*** This warning code applies only to a ShortStack Micro Server.

LID#	Description
4023	***Insufficient addresses are implemented for the specified number of network variables*** For more robust device behavior, increase the number of addresses.
4024	***The DMF window is too large and is unlikely to be supported by your Micro Server hardware.*** This warning code applies only to a ShortStack Micro Server.
4025	***The program ID's channel identifier should be set to 0x04 (TP/FT-10)***
4026	***Your transceiver buffer configuration leaves a number of bytes unused***
8001	***Your device supports the file transfer protocol, but no configuration property files are available***
8002	***The Micro Server's default transceiver does not match the explicitly specified transceiver*** A Micro Server image supports a default transceiver type. This warning indicates that a different transceiver has been chosen. The LonTalk Interface Developer utility calculates the correct communication parameters for the desired transceiver. Consult your Micro Server documentation for possible restrictions of supported transceiver types. This hint code applies only to a ShortStack Micro Server.
8003	***Persistent network variables were declared – some mechanism to implement data persistency must be provided*** When the LonTalk Interface Developer utility recognizes a network variable that has been declared with an **eeprom** modifier, or when the LonTalk Interface Developer utility recognizes non-constant configuration network variables, the utility issues this warning to remind you to provide persistent data storage for these items. Such data storage also applies to configuration properties implemented within configuration files, unless they are read-only. The type of persistent data storage that you provide depends on the application. For example, you could use special, non-volatile memory devices (such as EEPROM, flash, or NVRAM), or some other non-volatile storage media (such as floppy disk drives, hard disk drives, or solid-state drives). This hint code applies only to a ShortStack Micro Server.
8004	***Copied the specified Micro Server image file into the project folder*** This hint code applies only to a ShortStack Micro Server.
8005	***Your transceiver buffer configuration leaves a number of bytes unused***

LID#	Description
8006	***Your application supports direct memory files, but no configuration property files are implemented*** Your model file contains a network variable of type **SNVT_address**, but your project does not include a configuration property file. Remove the **SNVT_address** network variable if it is not used.

5

LonWorks XML Errors (LWX)

This chapter lists and describes the errors that can be reported by the LONWORKS XML software component. This software component is used by multiple other components, including the Neuron Linker, the Project Make utility, the IzoT NodeBuilder software, and others. Therefore, these errors can be reported when using any of these tools.

LWX Errors

Table 7 lists the LWX error codes.

Table 7. LWX Error Codes

LWX#	Description
101	***Unexpected non-numeric value found in XML data <string> [LWX#101]*** This might happen if an XML file was edited outside of IzoT NodeBuilder tool. If the file is saved through the IzoT NodeBuilder tool and its components, it might be fixed up, but data may be lost. Open the file with an editor, find the noted string and see if there is anything obviously wrong with the text and try to fix it.
120	***Unexpected code path [LWX#120]*** Only a program error can cause this. Re-install the product and choose the **Repair** option when prompted.
121	***Failed to create COM object [LWX#121]*** Verify that the MSXML3.DLL file is in the system folder. Attempt to fix the problem by manually re-registering the MSXML3.DLL with the Windows REGSVR32.EXE utility. Open a command prompt and enter the following command: **Regsvr32 msxml3.dll**
122	***Overwrite tried on a write-protected file [LWX#122]*** Perhaps a file was not checked out of source code control. Clear the read-only attribute, and try again if you need to update the file.
123	***Uninitialized pointer [LWX#123]*** Only a program error can cause this. Re-install the product and choose the **Repair** option when prompted.
124	***File not found [LWX#124]*** Check your settings to make sure the file path is correct and verify that the file exists.
125	***Unexpected data [LWX#125]*** This could be due to a renamed file, a hand edited file or a program error. Re-install the product and choose the **Repair** option when prompted.
128	***Non-unique XML key found [LWX#128]*** Collections need unique key values. The file was probably edited by hand. Edit the file to remove the duplicate so it may be loaded.

LWX#	Description
131	***Expected format attribute not found [LWX#131]*** Either the file was added by hand or there is a program error. Re-install the product and choose the **Repair** option when prompted.
132	***Unrecognized boolean constant'<v>' [LWX#132]*** A malformed value was read for a Boolean element or attribute. The utility recognizes 1, TRUE and YES for logical true values, 0, FALSE and NO for logical false values. The related XML data was probable edited by hand. Try to revert or correct these changes, or re-install the product and choose the **Repair** option when prompted.

6

Neuron Assembler Errors (NAS)

This chapter lists and describes the errors that can be reported by the Neuron Assembler.

NAS Errors

Table 8 lists the NAS error codes.

Table 8. NAS Error Codes

NAS#	Description
1	***Too many predefined symbols [NAS#1]*** The assembler encountered too many symbols. Try reducing your source in size and complexity (for example, you could split your source into multiple modules.
2 **3**	***Out of memory [NAS#2]*** ***Out of memory [NAS#3]*** The assembler failed to allocate required memory from the operating system. Close the IzoT NodeBuilder tool, IzoT Commissioning tool, and other applications, and then try again.
4	***Program error code <code> [NAS#4]*** An internal error occurred.
5	***Too many errors [NAS#5]*** The assembler stopped assembling your code after too many errors were encountered. Address the errors reported and try again.
6	***Division by zero in expression [NAS#6]*** The expression results in a division by zero. Review your expression.
7	***Expression list is too long [NAS#7]*** The expression is too complex.
8	***Unknown op 'z' in emit_expr [NAS#8]*** An internal error occurred.
9	***Label <s> is already defined [NAS#9]*** Label names must be unique for each module. Rename the duplicate name.
10	***List of symbols is too long [NAS#10]*** See NAS#1
11	***Keyword EXPORT must be preceded by label or followed by symbol [NAS#11]*** A syntax error was detected in your use of the EXPORT directive. It must be followed by a symbol, or it must be preceded by a label definition.

NAS#	Description
12	***Symbol <s> is used but never defined [NAS#12]*** An unknown symbol was used. For symbols defined within the same module, check the spelling of the symbol definition and reference. For symbols referring to items defined in other modules, make sure to IMPORT the symbol.
13	***Symbol <s> past end of relocatable segment [NAS#13]***
14	***Too many search paths [NAS#14]*** Too many search paths were specified with the –s (--search) command line option. Specify fewer search paths.
15	***Include files are too deeply nested [NAS#15]*** Include files are too deeply nested, or deep nesting occurs due to include file recursion. Review your INCLUDE statements and include files.
16	***Too many input files [NAS#16]*** Too many input files. Reduce the number of include files.
17	***System open file limit exceeded [NAS#17]*** The assembler could not open a file due to an operating system restriction.
18	***Cannot open <file> [NAS#18]*** The assembler failed to open the file described in the message. Check to see if the file exists and is accessible.
19	***Cache error for file <file> [NAS#19]*** An internal error occurred. Close the tool and try again.
20	***IF[[N]DEF] too deeply nested [NAS#20]*** Too many nested IF, IFDEF or IFNDEF expressions. Reduce the complexity of your conditional assembly.
21	***ELSE without matching IF[[N]DEF] [NAS#21]*** An ELSE directive was encountered without a matching IF, IFDEF or IFNDEF directive. Review your conditional assembly expression.
22	***ENDIF without matching IF[[N]DEF] [NAS#22]*** An ENDIF directive was encountered without a matching IF, IFDEF or IFNDEF directive. Review your conditional assembly expression.

NAS#	Description
23	***IF[[N]DEF] is never terminated [NAS#23]*** A conditional expression using the IF, IFDEF or IFNDEF directive was recognized, but the terminating ENDIF was never encountered. Review your conditional assembly expression.
24	***Too many conditional assembly directives [NAS#24]*** Too many conditional assembly directives were encountered. Try reducing the complexity of your conditional assembly, or split the module into two.
25	***Unexpected character <c> in input [NAS#25]*** A general parser error occurred due to invalid input. Review your source code and correct the misspelled directive or instruction.
26	***An invalid radix character was selected [NAS#26]*** NAS supports b, d, h and o radix indicators for binary, decimal, hexadecimal and octal constants, only. The radix indicator provided was none of those.
27	***An invalid constant was input [NAS#27]*** The constant is invalid.
28	***The constant does not match the radix setting [NAS#28]*** Use digits 0 and 1 for binary constants, 0-7 for octal, 0-9 for decimal, and 0-9 and A-F (case insensitive) for hexadecimal constants.
29	***This constant is too large <value> [NAS#29]*** The constant's value exceeds the limits of its type
30	***Truncating long name <name> [NAS#30]*** The name provided is too long. NAS truncates the name and continues to operate; however, the truncation may cause follow-on errors if it results in symbol duplication.
31	***Cannot use keywords as labels [NAS#31]*** Choose a different label.
32	***Unterminated string constant [NAS#32]*** You must enclose strings with a pair of quotes.
33	***<radix> is not a valid radix name [NAS#33]*** Use only BINARY, OCTAL, DECIMAL or HEX with the RADIX directive.

NAS#	Description
34	***Invalid name for the SEG directive [NAS#34]*** You specified an invalid segment type. See the *Neuron Assembly Language Reference* for a discussion of the SEG directive.
35 **36**	***RESOURCE directive requires expression [NAS#35]*** ***RESOURCE expression must be a constant [NAS#36]*** A RESOURCE directive is being used incorrectly. RESOURCE directives are reserved for use by the Neuron C Compiler. Do not specify RESOURCE directives in your Neuron Assembly code.
37	***ORG value must be a constant [NAS#37]*** You must use a constant value (or none) with the ORG directive. You cannot use expressions with the ORG directive.
38	***Cannot evaluate IF expression [NAS#38]*** A problem occurred when evaluating an IF expression. Check your conditional assembly code.
39	***EQU requires a label [NAS#39]*** Provide the missing label.
40	***Cannot resolve EQU expression [NAS#40]*** A problem was encountered with an EQU directive. Review your code.
41	***RES value must be a constant [NAS#41]*** A problem was encountered with a RES directive. Review your code.
42	***Pointer number must be a constant [NAS#42]*** Pointer registers can only be referenced using constant identifiers in the 0..3 range.
43	***Pointer value is out of 0..3 range [NAS#43]*** Pointer registers can only be referenced using constant identifiers in the 0..3 range.
44	***Small immediate field must be a constant [NAS#44]*** Only constant values are supported with the PUSHS instruction.
45	***Constant is out of 1..8 range [NAS#45]*** A constant was outside the supported value range.
46	***Constant is out of 0..7 range [NAS#46]*** A constant was outside the supported value range.

NAS#	Description
47	***Offset must be a constant [NAS#47]***
48	***Constant is out of -1..-8 range [NAS#48]***
49	***Offset value is out of 0..255 range [NAS#49]***
50	***Offset value is out of 8..23 range [NAS#50]***
51	***First operand is out of 0..255 range [NAS#51]***
52	***Second operand is out of 0..255 range [NAS#52]*** A constant, offset value or operand was outside the supported value range.
53	***<f> is an unknown function name [NAS#53]*** See the *Neuron Assembly Language Reference* for a list of supported function names.
54	***Invalid resource name [NAS#54]*** A RESOURCE directive is being used incorrectly. RESOURCE directives are reserved for use by the Neuron C Compiler. Do not specify RESOURCE directives in your Neuron Assembly code.
55	***Constant is too large [NAS#55]***
56	***Could use small branch instruction [NAS#56]*** A performance warning. Smaller branch instructions use less code space and execute faster.
57	***Could use smaller 'PUSHS' instead [NAS#57]*** You can use PUSHS to push immediate constants in the 0..7 range. PUSHS uses less code space and executes faster than PUSH.
58	***Useless instruction [NAS#58]*** The instruction has no effect.
59	***Could use smaller 'INC' instead [NAS#59]***
60	***Could use smaller 'DEC' instead [NAS#60]***
61	***Could combine 'RET' with previous instruction [NAS#61]***
62	***Could replace 'CALLR'+'RET' with 'SBR' [NAS#62]***
63	***Could replace 'CALLR'+'RET' with 'BR' [NAS#63]***
64	***Could replace 'CALL'+'RET' with 'BRF' [NAS#64]***
65	***Could replace 'CALLF'+'RET' with 'BRF' [NAS#65]***
66	***Could use immediate-operand instruction with previous push [NAS#66]*** These suggestions are provided for more efficient code.

NAS#	Description
67	***Value <v> is larger than 0x1FFF [NAS#67]***
68	***Offset <o> is out of 0..15 range [NAS#68]***
69	***Offset <o> is out of -128..127 range [NAS#69]***
70	***Value <v> is out of 0..255 range [NAS#70]***
71	***Offset <o> is out of 8..23 range [NAS#71]***
72	***First value <v> is out of 0..255 range [NAS#72]***
73	***Second value <v> is out of 0..255 range [NAS#73]***
74	***Constant <c> too large [NAS#74]***
75	***Zero-length segment discarded [NAS#75]*** This diagnostic is not in use.
76	***Invalid global expression [NAS#76]***
77	***Cannot create listing file - disk full? [NAS#77]*** The tool cannot create the listing file. The disk might be full, or the path is inaccessible.
78	***Cannot write listing file - disk full? [NAS#78]*** The tool cannot create the listing file. The disk might be full, or the path is inaccessible.
89	***Cannot create file <file> [NAS#89]***
90	***Write error [NAS#90]*** The tool cannot create or write to a file. The disk might be full, or the path not accessible.

NAS#	Description
106 **107** **108** **109**	***Cannot write dependency file .nadep (might cause build status calculation to become incorrect) [NAS#106]*** ***Cannot add switch record to dependency file .nadep (might cause build status calculation to become incorrect) [NAS#107]*** ***Cannot add input file record to dependency file .nadep (might cause build status calculation to become incorrect) [NAS#108]*** ***Cannot add output file record to dependency file .nadep (might cause build status calculation to become incorrect) [NAS#109]*** A file cannot be referenced when being added to a dependency file, or a non-recoverable problem occurs when investigating the file described by an existing dependency file record. This might be caused by the file being present but corrupt, or being locked by some other process. See <reason> provided in the message for failure details. You can ignore this message unless it persists; however, you should try to perform an unconditional rebuild because the missing data could cause the Project Make Facility to incorrectly determine the device's build status.
110	***Unspecified error in option processing [NAS#110]***
111	***Unspecified error in execution of assembler [NAS#111]***
115	***<message as defined by user via ERROR directive> [NAS#115]*** This is a user-defined error message that you have defined in your source code with the ERROR directive.
116	***Too many libraries. Ignoring <library> [NAS#116]*** Too many libraries are listed with the LIBRARY directive. List additional libraries explicitly when linking.

7

Neuron C Compiler Errors (NCC)

This chapter lists Neuron C compiler warning and error messages, and offers suggestions for how to correct the indicated problems.

NCC Errors

Table 9 lists the NCC error codes.

Table 9. NCC Error Codes

NCC#	Description
1	***Maximum token length exceeded [NCC#1]*** Neuron C tokens are limited to a maximum of 256 characters. This applies to identifiers, numbers, string constants, and so on. There is no limit on the lengths of Neuron C comments.
2	***Character in input is not acceptable for C source [NCC#2]*** The Neuron C compiler uses only the minimum ANSI C standard character set. Additionally, the characters **$**, **@**, and **`**(accent-grave) can be used in string and character constants. All other non-standard characters are treated as white space, except for **^D** and **^Z**. Appearance of either of these two characters in the input file is taken to be an end-of-file marker.
3	***Float constants are not supported [NCC#3]***
4	***Float exponent too big [NCC#4]***
5	***The 'expand_array_info' option does not apply to a msg_tag [NCC#5]***
6	***Comment not properly terminated [NCC#6]*** An end-of-file condition was discovered in the middle of a comment. This is an unterminated comment condition and is an error. The error message contains the beginning of the comment.
7	***Comment may not be properly terminated [NCC#7]*** This warning may be useful in discovering unintentional comments in your Neuron C program. The definition of C does not permit nesting of comments. Any text of the form shown below is treated as a single comment. `/* <text> /* <text> */` Note, however, that this particular pattern may indicate a condition where there are actually two comments intended, but the first is unterminated. Thus, the compiler detects this condition and prints a warning message. The comment text is printed also, for up to 256 characters.
8	***Character constant is too long [NCC#8]*** Neuron C only supports single-character constants (this does not apply to use of the \ escape character sequence).

NCC#	Description
9	***String constant is not terminated [NCC#9]*** ANSI C does not permit a string constant to span lines; nor can a string constant be terminated by end-of-file. To create a very long string constant, use the ANSI C string constant concatenation feature, demonstrated below. Note that the parts of the string are concatenated without insertion of any white space, newline, or other separator character. "This is a long string constant " "split across two source lines." Note that this error message may also indicate mismatched quotes in strings. Use \" to include a quote character in a string constant.
10	***Preprocessor directives cannot be nested in macros [NCC#10]*** A macro cannot contain the character **#** outside of the text of a string or a character constant.
11	***Directive #else/#endif without corresponding #if/#ifdef/#ifndef [NCC#11]*** The preprocessor directives controlling conditional compilation must always exist in matching pairs, similar to the open brace **{** and close brace **}** in a C program. For example, the pair **#ifdef** and **#endif** must match. An optional **#else** may be contained in between the **#ifdef** and **#endif** directives.
12	***Unrecognized or ill-formed pragma was ignored [NCC#12]*** The pragma referenced by the error message is not one which is recognized or supported by the Neuron C compiler.
13	***Cannot enable micro_interface with Net Vars or msg_tags declared [NCC#13]*** The **#pragma micro_interface** can only appear in a program if there have not been any prior declarations of network variables or message tags. This pragma can only be used with the LonBuilder Microprocessor Interface Program (MIP).
14	***Invalid value for this pragma [NCC#14]*** The numeric value following the pragma that the message refers to is not of appropriate value. Consult the documentation for the specific pragma to ascertain the applicable valid values. Pragmas are documented in the *Compiler Directives* chapter of the *Neuron C Reference Guide*.

NCC#	Description
15	***Cannot repeat this pragma [NCC#15]*** The following compiler directives can only be used once: **codegen** **num_addr_table_entries** **num_alias_table_entries** **num_domain_entries** **one_domain** **receive_trans_count** **set_guidelines_version** **set_id_string** **set_netvar_count** **set_device_sd_string** **set_std_prog_id** **specify_io_clock**
16	***Macro name, macro parameter name, or macro argument is too long [NCC#16]*** No identifier in Neuron C can exceed 256 characters
17	***Line too long in macro definition [NCC#17]*** No input line in Neuron C can exceed 256 characters. You can use the line continuation feature of ANSI C to extend the line. This feature is activated by using a \ character at the end of the line.
18	***Invalid preprocessor directive syntax [NCC#18]*** This error indicates one of any number of syntax problems in the compiler directive of the line indicated. The proper syntax is: `#directive [value]` where the optional *value* is dependent on the particular directive.
19	***Extra entries in preprocessor directive [NCC#19]*** This error indicates that, although the preprocessor directive was of the correct syntax, there are additional entries on the line that were not part of the directive.
20	***Empty input source file*** Check for the existence of the file that is being compiled. Is the file name and path name correct?
21	***Unexpected END-OF-FILE in source file [NCC#21]*** An incomplete source construct unexpectedly ended in an end-of-file condition. This may indicate mismatched brace characters **{** and **}** or may indicate the use of a function macro with an insufficient number of right (ending) parentheses.

NCC#	Description
22	***Repeated keyword was ignored [NCC#22]*** The keyword **const** or **volatile** is used more than once in modification of a pointer type.
23 **24**	***Not enough address table entries [NCC#23]*** ***Not enough address table entries for optimum efficiency [NCC#24]*** Most LONWORKS devices are limited to 15 address table entries. Each bindable message tag consumes one address table entry, whether bound or not. Network variables can share address table entries, but there must be at least one available. If there aren't enough address table entries for all the message tags plus at least one for network variables, you get the error [NCC#23]. However, if there aren't enough entries available for each output network variable to have its own address table entry, you get the warning [NCC#24] (unless you already have the maximum number of address table entries in your program). This is because the network variable binder would then have to *share* the remaining address table entries among the network variables. Example**:** If there are three network variables (each going to a different destination) and there are only two address table entries, then at least two of the network variables would have to use the same address table entry (if they are all connected). Now let's assume that all the variables are connected, each point-to-point to a different device. If each variable had its own address table entry, the LonTalk messages would all use subnet/device (that is, point-to-point) addressing. However, for the two variables sharing the *same* entry, a group will be constructed. This means that, when either variable is updated, the updates will go to all members in the group. This does not necessarily cause a problem, as the nodes that don't have the variable will discard the update. The major inefficiency the compiler is warning about, though, is that each destination in the group, regardless of whether it uses the message, will respond with an acknowledgment message. This situation thus leads to increased unnecessary acknowledgements, and extra network traffic.
25	***Cannot open assembly output file [NCC#25]*** The compiler cannot open the output file for code generation. This could be caused by an existing file being marked as read-only, or a missing folder, or a problem with the operating system.
26	***Cannot open bplate.ns [NCC#26]*** During compiler initialization, the compiler attempts to open several support files. One of these files is named **bplate.ns**. This file should reside in NodeBuilder system include directory (default location is \Lonworks\NeuronC\Include). This message could indicate a disk error.

NCC#	Description
27	***Special event & init code block exceeds size limitation [NCC#27]*** The tasks corresponding to the **reset**, **online**, **offline**, and **wink** events, as well as any **when** clause arbitrary expressions all generate code in a special area known as the APINIT block. If **#pragma disable_mult_module_init** (see the *Compiler Directives* chapter of the *Neuron C Reference Guide*) is used, any non-zero initialization of global RAM variables and I/O objects place code here as well. This block is limited in size to 255 bytes. If you exceed the size of this block, try moving the bulk of code in any tasks that correspond to **reset**, **online**, **offline**, and **wink** events to functions that are called from these tasks. If you are using the **#pragma disable_mult_module_init** directive, remove the pragma.
28	***Incorrect I/O object type for io_changes event [NCC#28]*** See the description of the event in the *Predefined Events* chapter of the *Neuron C Reference Guide*. Verify that your I/O object type supports this event.
29	***Use only 15000, 10000, or 1000 for I/O object's baud [NCC#29]*** The **bitshift** I/O object types can have their bit rates specified with either the **baud** or **kbaud** I/O declaration modifier. If **kbaud** is used, the only legal values are 15, 10, and 1. If **baud** is used, the only legal values are 15000, 10000, and 1000. The default bit rate for these I/O object types is 15kbps, and need not be specified.
30	***Use only 15, 10, or 1 for kbaud rate value [NCC#30]*** The **bitshift** I/O object types can have their bit rates specified with either the **baud** or **kbaud** I/O declaration modifier. If **kbaud** is used, the only legal values are 15, 10, and 1. If **baud** is used, the only legal values are 15000, 10000, and 1000. The default bit rate for these I/O object types is 15kbps, and need not be specified.
31	***Too many 'when' clauses [NCC#31]*** Neuron C places entries representing the when clauses in a table that is interpreted by the Neuron Chip firmware scheduler. The table's entries are variable sized, as some event expressions are more complex than others. The table size is limited to 256 bytes. When the table is full, no more when clauses can be accepted. Note that the limit is on the number of when clauses and not on the number of when tasks.
32	***Cannot open binder interface file(s) [NCC#32]*** This problem could occur when the compiler attempts to open files with .BIF or .BF2 extension, but the file cannot be opened properly with write access. It is possible that the file is marked read-only, or that the output folder does not exist, or there is a disk or operating system problem.

NCC#	Description
33	***Cannot open assembly include file named in pragma directive [NCC#33]*** There is a **#pragma include_assembly_file** that can be used to include an assembly source file in the compiler code generator's assembly code output file. The named file cannot be found or opened.
34	***Attempt to divide by the constant zero [NCC#34]*** The compiler detected that a constant expression contains a division by zero. Constant expressions are evaluated at compile time by the Neuron C compiler. Correct the expression.
35	***SD string supplied exceeds 1023 character limit [NCC#35]*** The **sd_string** option for a network variable declaration is limited to a string of no more than 1023 characters (plus NUL terminator). It is possible, using the **bind_info(expand_array_info)** declaration option for a network variable array, that the string would be limited to less than 1023 characters in some situations.
36	***Message object reference has no value [NCC#36]*** The message objects, **msg_out**, **msg_in**, **resp_out**, and **resp_in**, have no value in themselves. They only have meaning when they are accessed using the dot operator (**.**) and a field name. Only certain predefined field names apply.
37	***This field must be indexed [NCC#37]*** The message objects, **msg_out**, **msg_in**, **resp_out**, and **resp_in**, have no value in themselves. They only have meaning when they are accessed using the dot operator (**.**) and a field name. Only certain predefined field names apply. The particular data field referred to by this message is an array, and access to it must be by index, except when used with **memcpy( )**.
38	***Possible data truncation [NCC#38]*** This message results from an automatic conversion of a **long** variable to a **short**. To make this warning go away, modify the variable declarations or use an explicit cast operator, which disables the compiler warning.
39	***Cannot open debug output info file [NCC#39]*** This problem could occur when the compiler attempts to open the output file with .DBG extension, but the file cannot be opened properly with write access. It is possible that the file is marked read-only, or that the output folder does not exist, or there is a disk or operating system problem.

NCC#	Description
40	***Enum list has more values than the debug info supports [NCC#40]*** The range of enum values in Neuron C is from -128 to 127 because ANSI C dictates an *enum* should be implemented using a (signed) *int* base type. According to the definition of ANSI C, multiple enumerated constant names may appear in an **enum** type for the same constant value; thus there is really no limit to the number of names in an **enum** value list. However, the Neuron C Debugger only supports a maximum of 255 enumerated constant names in a given **enum** type. An **enum** that contains more names than this is still perfectly acceptable to the compiler; however, the debugger is only capable of using the first 255 names in the **enum** type.
41	***Too many function parameters for debug info [NCC#41]*** The Neuron C Debugger can only support functions with 14 or fewer parameters. Use of functions with more than 14 parameters may result in strange or incorrect results when using the debugger to display stack contents, and so on. Note that this is a limitation on number of parameters, and *not* on the number of bytes used to store those parameters.
42	***Too many include search directories specified [NCC#42]*** A maximum of 20 directories may be specified in the include-directory search list, 18 of which are available to the programmers.
43	***Cannot open output dependency-file [NCC#43]*** The compiler opens several output files as part of its initialization. All the files are placed in the intermediate folder. In the IzoT NodeBuilder development tool, the intermediate folder is the IM subfolder in each target folder ("Release", "Development", and so on). This error may indicate that the disk is full, or may indicate a disk error, or may indicate a read-only disk condition (if for example, the project directory and its subdirectories are on a floppy disk and the disk is write-protected). Also, confirm that the project folder contains a subfolder named IM.
44	***Too many include files [NCC#44]*** A maximum of 254 files may be opened in a single compilation. The source file and the three compiler helper files count as four files altogether, thus there may be no more than 250 application include files. (An include file included from another include file counts as a separate file.)
45	***System open file limit exceeded [NCC#45]*** This problem should not be seen under modern Windows operating systems because there is no hard limit on open files.
46	***Class 'register' was ignored [NCC#46]*** This keyword has no effect in Neuron C.

NCC#	Description
47	***Type qualifier 'volatile' was ignored [NCC#47]*** This keyword has no effect in Neuron C. To work with variables that are asynchronously accessed from *interrupt*-tasks and *when*-tasks, use the **__lock** construct for coordinated access.
48	***Floating point is not supported [NCC#48]*** Neuron C does not support floating point built-in data types and operators. Use the floating point library functions instead. See the *Functions* chapter in the *Neuron C Reference Guide* for more information on using the floating point library, and the standard include file **<float.h>**.
49	***Invalid array bounds [NCC#49]*** In Neuron C, an array bound constant-expression must resolve to a positive integer no larger than 32767.
50	***Bitfield size must be from 0 to 8 bits [NCC#50]*** A bitfield must fit inside an **int** type. Since in Neuron C an **int** is 8 bits, the bitfield is also limited to 8 bits. Note that in ANSI C a bitfield size of 0 bits is legal for an unnamed bitfield. Such a bitfield forces alignment to the next storage unit boundary (which is a byte in Neuron C, since the Neuron Chip architecture does not force any multi-byte data alignment.).
51	***Bitfield type is incorrect [NCC#51]*** A bitfield may only be declared using only a combination of the type keywords **int**, **signed**, **unsigned**, **long**, **short**, and **char**. Bitfields may not be arrays, pointers, structures, unions, or any other Neuron C type.
52	***Bitfield size cannot be 0 unless unnamed [NCC#52]*** In ANSI C a bitfield size of 0 bits is legal for an *unnamed* bitfield. Such a bitfield forces alignment to the next storage unit boundary (which is a byte in Neuron C, since the Neuron Chip architecture does not force any multi-byte data alignment.).
53	***Keyword 'polled' is ignored for input network variable [NCC#53]*** The **polled** keyword only applies to output network variables. This restriction does not apply to the case of model file compilation that is used within the ShortStack, FTXL, or LIBILON development environment.
54	***Repeated bind_info option [NCC#54]*** The **bind_info** keyword is followed by a parenthesized list of one or more options, some with associated values. Each keyword may appear at most once.

NCC#	Description
55	***Storage class on struct/union field not permitted [NCC#55]*** A field in a **struct** or **union** may not have a storage class, and may not contain the word **typedef**. Nor can it be a message tag, a timer object, nor a network variable, nor can it have **bind_info**. Note also that a **union** may not contain bitfields.
56	***Enum constant out of range [NCC#56]*** In Neuron C as in ANSI C, an enumerated type, declared using the **enum** keyword, is equivalent to an **int** type. The **int** type is **signed**. In Neuron C, an **int** is implemented using 8-bit signed two's complement representation. Thus, the valid range of enumerated type values is –128 to +127.
57	***Enum value wrapped around to negative [NCC#57]*** The enumerated type automatically assigns consecutive integers as the values of the named constants unless specifically given a value for a constant. When *literal-constant-n* has the value 127, and *literal-constant-n+1* appears, the compiler automatically assigns it the "next" value. In Neuron C, this value is -128 (since the **enum** type is **signed** in ANSI C). The compiler issues a warning diagnostic for this condition, and proceeds.
58	***Nothing was declared [NCC#58]*** Similar to *Declaration defaults to 'int', No declaration for formal parameter - int assumed*, in a situation like the following: long; int j; This is legal C, but may not be what the programmer intended. Thus, for the "first" declaration (the statement **long;**) this diagnostic is produced. Many C compilers do not issue a warning in these circumstances. Note that Neuron C will *not* issue a warning in the following cases, since something *is* being declared (namely the enum's or the struct field names, respectively): enum { FALSE, TRUE } ; struct { int a; int b; };
59	***Expression type mismatch [NCC#59]*** These error diagnostics result from combining expressions of conflicting types, such as assigning an **int** to a pointer, or a pointer of one type to a pointer to another type, or in using objects that have no value (such as message tags or I/O object names) in expressions. In many cases in ANSI C, you must use an explicit type cast. However, note that casting should be avoided if possible, as it is often masking poor programming practice.

NCC#	Description
60	***Invalid subscript operation [NCC#60]*** The compiler outputs this diagnostic when an array-index operator **[** *expr* **]** is applied to a variable that is not a pointer or an array.
61	***Object is not a struct/union pointer [NCC#61]*** The compiler outputs this message when the left-hand side of the **->** operator is not a pointer to a struct or union type.
62	***Object is not a structure or union [NCC#62]*** The compiler outputs this message when the left-hand side of the dot (**.**) operator is not a struct or union type.
63	***Invalid cast operation [NCC#63]*** Some conversions between data types are not permitted, even through an explicit cast. For example, an object cannot be cast if its base type is not known. Nor can an object be cast to **void** and then used in an expression.
64	***Cannot remove 'const' attribute via cast operation [NCC#64]*** To prevent data that is declared constant from being modified, Neuron C will not permit pointers including the **const** attribute from being cast such that the **const** attribute is removed. Neither does the compiler permit an implicit conversion of pointer (for example, by a function call) such that the **const** attribute would be removed. However, use of the **#pragma relaxed_casting_on** changes this error into a warning. See the *Compiler Directives* chapter in the *Neuron C Reference Guide* for more information on this pragma.
65	***Cannot compute 'sizeof' for object [NCC#65]*** The compiler attempted to calculate the size of a type, but did not have enough information. This could result from a **sizeof** expression for an object like a timer object, or a message tag, or a **typedef** name which is a **bind_info**, or some similar circumstance.
66	***Message object reference cannot be assigned to [NCC#66]*** The message objects **msg_in** and **resp_in** are read-only. Attempts to use these objects on the left side of an assignment statement result in the diagnostic message above.
67	***Pulsecount I/O object cannot use clock 0 [NCC#67]*** The **pulsecount** I/O object does not support use of clock 0. Its use will produce an indeterminate number of output pulses.

NCC#	Description
68	***Message event code must be in range 0..127 [NCC#68]*** The event **msg_arrives** accepts one optional parameter, which is a message event code. This code must be a compile-time constant integer expression with a value from 0 to 127, inclusive.
69	***Parameter must be a msg_tag [NCC#69]*** The events **msg_completes**, **msg_succeeds**, and **msg_fails** all accept one optional parameter, which must have previously been declared as a message tag.
70 **71** **72** **73** **74** **75**	***Parameter must be an I/O object name [NCC#70]*** ***I/O events only apply to input objects [NCC#71]*** ***Incorrect I/O object type for changes-by event [NCC#72]*** ***Incorrect I/O object type for changes-to event [NCC#73]*** ***Incorrect I/O object type for I/O update-occurs event [NCC#74]*** ***Too many I/O object change events used [NCC#75]*** The Neuron C event expressions for **io_update_occurs** and **io_changes** (with its various options) only apply to I/O objects that are inputs. Furthermore, some events are not applicable to some input object types. Only one form of event expression can be used per I/O object. A maximum of 15 I/O objects can have **io_update_occurs** and **io_changes** events. The syntax of the event expressions requires the event type to be followed by the object name, in parentheses. For more information, see the *I/O Objects* chapter of the *Neuron C Reference Guide*.
76	***Event 'nv_update_occurs' only applies to input variables [NCC#76]*** The **nv_update_occurs** event accepts one optional parameter, which must be the name of a previously declared input network variable.
77	***Parameter must be a network variable [NCC#77]*** The **nv_update_completes**, **nv_update_fails**, and **nv_update_succeeds** events all accept one optional parameter, which must be the name of a previously declared network variable.
78	***Parameter must be a timer name [NCC#78]*** The event **timer_expires** accepts one optional parameter, which, if supplied, must be the name of a previously declared timer object.
79	***Invalid use of VOID type [NCC#79]*** The **void** type has no size. It cannot be used as an argument of the **sizeof** operator, nor can it be used to declare a variable. Its only legal uses are in declaring function return types, declaring that a function has no parameters, and in combination with ***** to define a type **void *** (a wildcard pointer type).

NCC#	Description
80	***Use only 4800, 2400, 1200, or 600 for I/O object's baud [NCC#80]*** The **serial** I/O object type can only have a bit rate value of 600, 1200, 2400, or 4800. The bit rate value defaults to 2400 if not specified.
81	***Use only 20000, 10000, or 1000 for I/O object's baud [NCC#81]*** The **neurowire** I/O object types can have their bit rates specified with either the **baud** or **kbaud** I/O declaration modifier. If **kbaud** is used, the only legal values are 20, 10 and 1. If **baud** is used, the only legal values are 20000, 10000, and 1000. The default bit rate for these I/O object types is 20 kbps, and need not be specified.
82	***Use only 20, 10, or 1 for kbaud rate value [NCC#82]*** The **neurowire** I/O object types can have their bit rates specified with either the **baud** or **kbaud** I/O declaration modifier. If **kbaud** is used, the only legal values are 20, 10 and 1. If **baud** is used, the only legal values are 20000, 10000, and 1000. The default bit rate for these I/O object types is 20 kbps, and need not be specified.
83	***Invalid use of type [NCC#83]*** The compiler attempted to calculate the size of a type, but did not have enough information. This could result from a **sizeof** expression for an object like a timer object, or a message tag, or a typedef name which is a **bind_info**, or some similar circumstance.
84	***Offline does not apply to a msg_tag [NCC#84]*** Some of the options in the **bind_info** declaration modifier only apply to network variables, some only apply to output network variables, and some only apply to message tags. The offline keyword only applies to a **network input config** variable.
85	***Service types may not be specified for a msg_tag [NCC#85]*** Some of the options in the **bind_info** declaration modifier only apply to any network variable, some only apply to an output network variable, and some only apply to a message tag. Service types only apply to output network variables.
86	***Network variable array bound is incorrect [NCC#86]*** This error message can arise from a few different situations. First, the declaration of a network variable array may be a single-dimensioned array only (no larger-dimensioned arrays are supported). The other sources of this message are from attempting to use an index expression with a network variable that is not an array. This message can also indicate that the array bound portion of a network variable declaration, or a network variable event expression, is not in the valid range, or not of the proper format (for example, zero or a negative number is used).

NCC#	Description
87	***Too many msg-tags declared [NCC#87]*** A maximum of 15 message tags can be declared per device. These can be any combination of bindable and nonbindable message tags.
88	***Network variables cannot be declared as non-bindable [NCC#88]*** Some of the options in the **bind_info** declaration modifier only apply to any network variable, some only apply to an output network variable, and some only apply to a message tag. The **nonbind** modifier can only be used with a message tag declaration.
89	***Input network variables cannot have service-type [NCC#89]*** Some of the options in the **bind_info** declaration modifier only apply to any network variable, some only apply to an output network variable, and some only apply to a message tag. Service types only apply to output network variables.
90	***Base type of network variable is too large [NCC#90]*** A network variable array element, structure, or union is limited to 31 bytes.
91	***Too many initializers [NCC#91]*** A set of initializers (in braces **{** and **}**) has too many members for the aggregate (array, structure, or union) being initialized.

NCC#	Description
92	***Too many network variables declared [NCC#92]*** Each device has a maximum number of network variables that it can support *in principle*. That maximum number is a function of the chip model, the system firmware version, and the device technology. For example, most Series 3100 Neuron Chips and some Series 3100 Smart Transceivers are limited to 62 network variables. Series 5000 and Series 600 Chips, and other Neuron Chips and Smart Transceiver that support version 16 system firmware (or later), support up to 254 static network variables. Other devices, such as those based on a ShortStack Micro Server or those implemented on a SmartServer, could implement a different network variable maximum. These can be any combination of input and output variables. Each element of an array network variable counts separately. The NCC#92 message indicates that the application may implement too many network variables because the implemented number of network variables exceeds the maximum number of network variables for the selected target platform. Note that the absence of error NCC#92 does not guarantee the successful compilation of your code. Several conditions can lead to later build failure related to an excessive network variable count. These include the use of a system firmware version that is limited to fewer network variables than foreseen at compile time, and include memory allocation problems at link time. See Chapter 8 of the *Neuron C Programmer's Guide* for more information on managing memory resources.
93	***Network variable declaration not permitted if micro_interface [NCC#93]*** Once the **#pragma micro_interface** directive appears, the program cannot declare any network variables or message tags. See the *Compiler Directives* chapter in the *Neuron C Reference Guide*.
94	***Network variable base type cannot contain unbounded array [NCC#94]*** Network variable arrays must be declared with a fixed bound that is a compile-time constant.
95 **96**	***Network variable base type cannot contain function [NCC#95]*** ***Network variable base type cannot contain pointer [NCC#96]*** A network variable type cannot contain pointer types, addresses or function address types.
97	***Too many timers declared [NCC#97]*** Neuron C supports a maximum of 15 application **timer** objects of all types together.

NCC#	Description
98	***I/O objects can only be declared at file scope [NCC#98]*** Some Neuron C objects may only be declared at file scope. Thus, they are restricted to being globals, not automatics. These objects are timer objects, message tags, I/O objects, and network variables.
99 **100** **101**	***This I/O object type can only be 'output' [NCC#99]*** ***This I/O object type can only be 'input' [NCC#100]*** ***Must specify 'input' or 'output' for this I/O object type [NCC#101]*** Some Neuron C I/O object types can only be **input**, some can only be **output**, and some can be either. For the case where the direction is known from the I/O object type, the programmer need not specify the direction, but if specified, it must be the correct direction. For the case where the direction is not known from the I/O object type, the programmer must specify it.
102 **103** **104** **105** **106** **107** **108** **109** **110** **111**	***I/O object type restricted to pins IO_0 through IO_4 [NCC#102]*** ***I/O object type restricted to pin IO_0 [NCC#103]*** ***I/O object type restricted to pins IO_0 through IO_7 [NCC#104]*** ***I/O object type restricted to pins IO_0 or IO_1 [NCC#105]*** ***I/O object type restricted to pins IO_4 through IO_7 [NCC#106]*** ***I/O object type restricted to pins IO_4 or IO_6 [NCC#107]*** ***I/O object type restricted to pin IO_10 [NCC#108]*** ***I/O object type not allowed on pin IO_7 or IO_10 [NCC#109]*** ***I/O object type restricted to pin IO_8 [NCC#110]*** ***I/O object type restricted to pin IO_4 [NCC#111]*** Different I/O object types are permitted on different subsets of the Neuron Chip's I/O pins. For more information, see the *I/O Model Reference*.
112	***All names beginning with the characters 'SNVT_' are reserved [NCC#112]*** The program should not declare any identifiers, types, and so on, that begin with the characters **SNVT_**, to avoid any future compatibility problems with Standard Network Variable Types.
113	***Two-way I/O device should not be declared 'input' or 'output' [NCC#113]*** The declaration syntax of I/O objects permits the specification of input or output. However, some devices are actually bi-directional, for example the **parallel** I/O object. Neither the input nor the output keyword should be specified in the declaration of a bi-directional I/O object.

NCC#	Description
114	***Pin IO_4 needs 'mux' or 'ded' specification [NCC#114]*** For I/O object types that use a timer/counter, the timer/counter used is dependent on the pin assigned to the I/O object. There are two timer/counters, the dedicated (abbreviated **ded**) and the multiplexed (abbreviated **mux**). The dedicated circuit uses pin **IO_1** for output and pin **IO_4** for input. The multiplexed circuit uses pin **IO_0** for output and multiplexes among pins **IO_4**, **IO_5**, **IO_6**, and **IO_7** for input. For input objects using a timer/counter, the programmer need not specify which timer/counter circuit is being used except when the input I/O object is assigned to pin **IO_4**. Then, either the **mux** or **ded** keyword must be included in the declaration of the I/O object.
115	***Pins IO_5...IO_7 must use 'mux' timer [NCC#115]*** For I/O object types that use a timer/counter, the timer/counter used is dependent on the pin assigned to the I/O object. There are two timer/counters, the dedicated (abbreviated **ded**) and the multiplexed (abbreviated **mux**). The dedicated circuit uses pin **IO_1** for output and pin **IO_4** for input. The multiplexed circuit uses pin **IO_0** for output and multiplexes among pins **IO_4**, **IO_5**, **IO_6**, and **IO_7** for input. For input objects using a timer/counter, the programmer need not specify which timer/counter circuit is being used except when the input I/O object is assigned to pin **IO_4**. Then, either the **mux** or **ded** keyword must be included in the declaration of the I/O object.
116	***I/O object requires 'sync' pin specification [NCC#116]*** The **triac** I/O object type declaration must include an assignment for a **sync** pin. Since the **triac** type uses a timer/counter output, the **sync** pin must use a corresponding input on the same timer/counter. If the dedicated timer/counter is used, only **IO_4** can be used for the **sync** pin. If the multiplexed timer/counter is used, any of **IO_4**, **IO_5**, **IO_6**, or **IO_7** can be used.
117	***I/O object requires 'sync' pin on one of IO_4...IO_7 [NCC#117]*** The **neurowire** I/O object type declaration must also include specification of a pin to be used for an I/O object **select**. Only a pin from **IO_0** through **IO_7** may be used for a **select** pin.
118	***I/O object requires 'sync' pin on IO_4 [NCC#118]*** The **neurowire** I/O object type declaration must also include specification of a pin to be used for an I/O object **select**. Only a pin from **IO_0** through **IO_7** may be used for a **select** pin.
119	***I/O object requires 'select' pin specification [NCC#119]*** The **neurowire** I/O object type declaration must also include specification of a pin to be used for an I/O object **select**. Only a pin from **IO_0** through **IO_7** may be used for a **select** pin.

NCC#	Description
120	***The 'select' pin must be one of IO_0...IO_7 [NCC#120]*** The **neurowire** I/O object type declaration must also include specification of a pin to be used for an I/O object **select**. Only a pin from **IO_0** through **IO_7** may be used for a **select** pin.
121	***I/O object requires 'master', 'slave', or 'slave_b' [NCC#121]*** The **parallel** I/O object type declaration must be qualified with one of the keywords **master**, **slave**, or **slave_b** to specify which parallel I/O protocol is to be used.
122 **123**	***I/O object type not available on requested pin [NCC#122]*** ***Pin/resource conflict with a previous I/O object [NCC#123]*** In several cases, more than one Neuron Chip I/O object type can be assigned to a single pin within a single application. The rules for overlaying I/O object declarations are discussed in Chapter 2 of the *Neuron C Programmer's Guide*.
124	***Incorrect 'clock' select value [NCC#124]*** For I/O objects that accept a **clock** modifier in their declaration, the legal values are from 0 to 7, inclusive, except for the **pulsecount** output object, which uses only 1 to 7. The **clock** value must be a constant expression.
125	***Incorrect 'numbits' value or type [NCC#125]*** The **bitshift** I/O object type declaration can optionally specify the number of bits to be specified. This **numbits** modifier has as an argument that must be a compile-time integer constant expression with a value from 1 to 128.
126	***Bad I/O modifier for this I/O object type [NCC#126]*** Several of the I/O object declarations permit or require modifiers, like **mux**, **ded**, **sync**, and so on. These are permitted or required on a per I/O object-type basis. At most one of each type of modifier is permitted in a single declaration.
127	***Duplicate I/O object modifier not allowed [NCC#127]*** Several of the I/O object declarations permit or require modifiers, like **mux**, **ded**, **sync**, and so on. These are permitted or required on a per I/O object-type basis. At most one of each type of modifier is permitted in a single declaration.
128	***I/O object type cannot have an initial-pin-level [NCC#128]*** Most output object types permit specification of an initial pin-level value to be assumed by the pin on power up or chip reset, until the application program takes over. However, the I/O object for which this message is being output does not permit an initial pin level.

NCC#	Description
129 **130** **131**	***Initial-pin level must be in range 0...255 [NCC#129]*** ***Initial-pin level must be in range 0...15 [NCC#130]*** ***Initial-pin level must be 0 or 1 [NCC#131]*** Most output object types permit specification of an initial pin-level value to be assumed by the pin on power up or chip reset, until the application program takes over. All single-pin initial levels must be either 0 or 1. The **nibble** I/O object type, which uses four consecutive pins, can have initial values from 0 to 15, with the values being mapped as a binary number onto the four pins. Likewise, the **byte** I/O object type can have initial values from 0 to 255.
132	***Unacceptable function return type [NCC#132]*** A function in Neuron C cannot return an object that is a **struct** or **union** type, nor can it return an array type. However, a function *can* return pointers to such objects.
133	***Explicit addressing requires inclusion of <msg_addr.h> [NCC#133]*** An attempt to use the **msg_out.dest_addr** field or the **msg_in.addr** field has been detected, but cannot be compiled because the include file **<msg_addr.h>** was not included by the programmer. Note that the include directive must appear prior to the first such field reference. The include file is not needed for references to other fields of the **msg_out** or **msg_in** objects.
134	***Call only applies to bindable msg_tag [NCC#134]*** The **is_bound( )** built-in function returns TRUE (nonzero) if the requested object has been bound. Otherwise, it returns FALSE. The function applies only to network variables and to bindable message tags. A bindable message tag is a message tag declared *without* the **bind_info (nonbind)** option.
135	***Parameter must be either a msg_tag name or an NV name [NCC#135]*** The **is_bound( )**, **addr_table_index( )**, and **nv_table_index( )** built-in functions return TRUE (nonzero) if the requested object is bound (connected). Otherwise, they return FALSE. The functions apply only to network variables and to bindable message tags. A bindable message tag is a message tag declared without the **bind_info (nonbind)** option.
136	***Incorrect number of parameters [NCC#136]*** The compiler outputs these diagnostics when the number of actual parameters, or the actual parameter types, do not match those in the function prototype and they cannot be automatically converted.
137	***Parameter to 'poll' must be input network variable [NCC#137]*** The built-in function **poll( )** takes as its only argument the name of an input network variable. See the *Functions* chapter of the *Neuron C Reference Guide* for a definition of this function.

NCC#	Description
138	***Invalid 2nd argument to 'sleep' [NCC#138]*** The built-in function **sleep( )** has an optional second parameter which may be a previously declared Neuron C I/O object name or an I/O pin name. If an I/O object name is specified, the object's primary pin will be monitored for a wakeup condition. Alternately, a pin name may be explicitly specified. In both cases, this pin must be one of **IO_4**, **IO_5**, **IO_6**, or **IO_7**.
139	***Sleep wakeup I/O object must be an input pin on one of IO_4..IO_7 [NCC#139]*** The built-in function **sleep( )** has an optional second parameter which may be a previously declared Neuron C I/O object name or an I/O pin name. If an object name is specified, the object's primary pin will be monitored for a wakeup condition. The primary pin must be one of **IO_4**, **IO_5**, **IO_6**, or **IO_7**.
140	***Incorrect message object field reference [NCC#140]*** The message objects, **msg_out**, **msg_in**, **resp_out**, and **resp_in** have no value in themselves. They only have meaning when they are accessed using the dot operator (**.**) and a field name. Only certain predefined field names apply. The data field is an array, and access to it must be by index, except when used with the **memcpy( )** function.
141	***Not a field in specified struct/union [NCC#141]*** The compiler outputs this message when the right-hand side of the **->** or **.** operator is not a field in the struct or union type that corresponds to the left-hand-side expression.
142	***Invalid storage class combination [NCC#142]*** This diagnostic results from incorrect or conflicting combinations of storage class keywords, such as **eeprom** and **ram**. The error is used for more than just conflicting memory types. For example, an attempt to use the **cp** or **cp_family** keyword with a **timer** or a **msg_tag**, or any invalid combination of declaration class keywords could cause this error message.
143	***Repeated storage class keyword was ignored [NCC#143]*** Repeated keywords are ignored (for example, **const const** is the same as **const**), but a diagnostic message is printed.
144	***The 'quad' type is not supported. [NCC#144]*** Neuron C does not support the **quad** type, but **quad** is a reserved word for future support of a 32-bit **signed int** type. The keyword **quad** refers to the four bytes of data for the 32-bit signed integer. This type could also be called a **double long int**.

NCC#	Description
145	***Invalid data type combination [NCC#145]*** This diagnostic message results from incorrect or conflicting type combinations. For example, **short** and **long** is a conflicting type combination. Combining **timer** object declarations with the keyword **msg_tag**, for example, is an incorrect result type.
146	***Repeated data type keyword was ignored [NCC#146]*** Repeated keywords are ignored (for example, **int int** is the same as **int**), but a diagnostic message is printed.

147	***Type defaults to 'int' [NCC#147]***
	The definition of ANSI C permits a declaration at file scope without a type. Likewise, functions may be declared without a return type. Such declarations must default to **int**, by the ANSI definition. However, such declarations are poor programming practice, and may even indicate an error, thus the compiler issues a warning diagnostic.
	Consider the following example:
	`unsigned long x1, x2; x3;`
	Note the semicolon following **x2**. This is most likely a typographical error, however, ANSI C permits this and results in **x3** being declared by default as an **int**. Due to white space rules, this appears the same to the compiler as the following declaration:
	`unsigned long x1, x2;`
	`x3;`
	This is almost certainly *not* what the programmer intended, yet most C compilers do not issue a warning in these circumstances.

148 **149**	***Class 'config' can only be used with network variables [NCC#148]*** ***Class 'config' applies only to 'input' variables [NCC#149]*** The **config** keyword only applies to input network variables. However, in Neuron C Version 2, use of the **config_prop** (or **cp**) keyword declares a fully managed configuration property, whereas the **config** keyword declares a legacy configuration network variable. The legacy variable requires that the programmer must manually code the SD information necessary to make the **config** network variable known to a network management tool. More information on configuration properties can be found in the *Neuron C Programmer's Guide* and the *Neuron C Reference Guide*.
150	***Cannot re-declare 'bind_info' [NCC#150]*** The **bind_info** modifier can appear at most once in the declaration of a network variable or a message tag. The **bind_info** cannot be combined with other **bind_info** by concatenation.

NCC#	Description
151	***I/O function call requires arguments [NCC#151]*** Insufficient arguments (or no arguments) were passed to the I/O built-in call flagged by the compiler diagnostic. All I/O functions require at least one argument, namely the I/O object name.
152	***Name is not an I/O object name [NCC#152]*** The first argument passed to the flagged I/O built-in call is not a properly declared I/O object name. Note that in ANSI C, a general rule is that an object must be declared before its first use.
153	***I/O function not valid for this I/O object [NCC#153]*** Some built-in functions, such as **io_set_clock( )** and **io_select( )**, cannot be used on all I/O object types.
154	***This event cannot be duplicated [NCC#154]*** There are three special events in Neuron C which can only appear in at most one **when** clause. These events are **reset**, **offline**, and **online**.
155	***The 'priority' is ignored for this 'when' clause [NCC#155]*** There are three special events in Neuron C, namely **reset**, **offline**, and **online,** for which the declaration of priority has no effect. This is because, due to the special times at which these clauses are executed, they always have priority.
156	***Function must return a value [NCC#156]*** The function, whose declared return data type is *not* **void**, does not have a return statement (in every possible path to the end of the function) that returns a value of the appropriate type.
157	***Expression for switch must be a 'short' [NCC#157]*** The **switch** statement can handle values only in the range of the **int** data type, which is from -128 to 127 inclusive in Neuron C.
158	***Improper context for 'break' statement [NCC#158]*** A **break** statement can only occur inside a **do**, **for**, or **while** loop, or inside a **switch** statement.
159	***Object being declared cannot be initialized [NCC#159]*** Declaration-time initialization cannot be used for typedefs, timer objects, message tags, function parameters, structure tags, union tags, and enum tags.

NCC#	Description
160	***This declaration may only be at file scope [NCC#160]*** Some Neuron C objects may only be declared at file scope. Thus, they are restricted to being globals, not automatics. These objects are timer objects, message tags, I/O objects, and network variables.
161	***Type mismatch in function redeclaration [NCC#161]*** This diagnostic indicates that a function prototype does not match a subsequent prototype, or the definition of the function. The prototypes and definition must match in terms of their storage class (for example, **static**, **eeprom**, **ram**) as well as their return types and their number and types of parameters.
162	***Array must have bound [NCC#162]*** Use of the array type in a declaration must include a constant expression which is the array bound. The only time this bound may be omitted is in the declaration of a function parameter. In this case, use of the bound is ignored, and the parameter is actually treated as a pointer.
163 **164**	***Invalid struct/union field declaration [NCC#163]*** ***Invalid struct/union field type [NCC#164]*** A field in a **struct** or **union** may not have a storage class, and may not contain the word **typedef**. Nor can it be a message tag, a timer object, nor a network variable, nor can it have **bind_info**. Note also that a union may not contain bitfields.
165	***Invalid type for bitfield [NCC#165]*** A bitfield may only be declared using only a combination of the type keywords **int**, **signed**, **unsigned**, **long**, **short**, and **char**. Bitfields may not be arrays, pointers, structures, or any other Neuron C type.
166	***Field name in struct/union cannot be repeated [NCC#166]*** Each field name at a given level of a **struct** or **union** declaration must be unique. Names are case sensitive.
167	***Extern declarations cannot have initializers [NCC#167]*** The semantics of an **extern** declaration and an initialized declaration are incompatible. An **extern** declaration is intended to reference an object defined elsewhere (usually in another module, although it may be a forward reference). An initialized declaration is intended to be the defining declaration of the object. Only one such initialized declaration should appear for each object.
168	***Const variables require initialization [NCC#168]*** A variable declared with the **const** attribute must have an initializer. Note that this does *not* apply to a **typedef** that includes the **const** attribute.

NCC#	Description
169	***Call to 'io_out' requires output value parameter [NCC#169]*** The built-in function call **io_out( )**, which is used to initiate an output to a Neuron I/O object, must be given at least two parameters, the first being the I/O object name, and the second being the value to be output.
170	***Cannot have 'io_changes' & 'io_update_occurs' on same I/O object [NCC#170]*** The Neuron C event expressions for **io_update_occurs** and **io_changes** (with its various options) only apply to I/O objects that are inputs. Furthermore, some events are not applicable to some input object types. Only one form of event expression can be used per I/O object. A maximum of 15 I/O objects can have **io_update_occurs** and **io_changes** events. The syntax of all the event expressions requires the event type to be followed by the object name, in parentheses.
171	***Improper context for 'continue' statement [NCC#171]*** A **continue** statement can only occur inside a **do**, **for**, or **while** statement. It causes execution to immediately branch back to the first statement of the loop statement that contains the **continue** statement.
172	***Function definition does not allow return value [NCC#172]*** The function, whose declared return data type is **void**, has a statement of the form **return** *expression*.

173	***Expression has no effect - discarded [NCC#173]***
	The compiler outputs this warning diagnostic when the optimizer discards an expression. Examples of such expressions are:
	`x = y-1, z;  /* y-1 is discarded */`
	`a+3;  /* a+3 is discarded */`
	`x == 1? y: z;  /* z is discarded */`

174	***Return value of function was ignored [NCC#174]***
	A function that has a **return** type (other than **void**) is used in an expression, but the caller discards the return value without it being used or stored. The warning can be removed by casting the return of the function to **void**.
	Example:
	`int f(void) {return 0;}`
	`when (reset) {`
	`  (void)f();`
	`}`

NCC#	Description
175	***This event will never be reached [NCC#175]*** This message warns of the use of a specific, qualified event following a generic, unqualified event in the same class. As the generic one will catch the event first, the specific one will never evaluate to TRUE. This condition can only occur when using the **scheduler_reset** feature. (Failure to use the **scheduler_reset** feature with multiple event expressions that are not exclusive can result in unstable behavior.)
176	***This event duplicates or overlaps a previous one [NCC#176]*** In many cases, use of a **when** clause containing an event that is a duplicate or an overlap of a previous event expression would prevent the associated task from being executed, or may cause anomalous behavior, with one task being executed sometimes, and the other being executed the rest of the time. (This latter behavior would occur as the result of round-robin execution by the Neuron Chip firmware scheduler, if the **scheduler_reset** feature were not used.)
177	***Recommend use of 'scheduler_reset' feature [NCC#177]*** The compiler makes this recommendation when there is a possibility of anomalous execution of different tasks because the tasks' respective **when** clauses are not mutually exclusive.
178	***Cannot have any non-polled output network variables when more than 14 bindable message tags are defined [NCC#178]*** A Neuron Chip has up to 15 address table entries. Each bindable message tag consumes one address table entry, whether bound or not. Network variables can share address table entries, but there must be at least one available. This message indicates that there are no entries available for network variables because message tags are consuming all of the address table entries.
179	***Incorrect use of 'void' in function prototype [NCC#179]*** The **void** type has no size. It cannot be used as an argument of the **sizeof** operator, nor can it be used to declare a variable. Its only legal uses are in declaring function return types, declaring that a function has no parameters, and in combination with ***** to define a type **void *** (a wildcard pointer type).
180	***Function parameter may not be 'struct' or 'union' type [NCC#180]*** Neuron C does not support passing **struct** or **union** types by value as function parameters. You may use a pointer to the structure or union as a function parameter.

NCC#	Description
181	***Function declarations must use prototypes [NCC#181]*** Neuron C is more restrictive than ANSI C in this area. The Neuron Chip's stack machine architecture does not permit calling undeclared functions with unknown numbers of parameters.

182	***No declaration for formal parameter - int assumed [NCC#182]***
	The definition of ANSI C permits a declaration at file scope without a type. Likewise, functions may be declared without a **return** type. Also, it is possible to construct a typecast that does not actually contain a type. Such declarations must default to **int**, by the ANSI C definition. However, such declarations are poor programming practice, and may even indicate an error, thus the compiler issues a warning diagnostic.
	Consider the following example:
	`unsigned long x1, x2; x3;`
	Note the semicolon following x2. This is most likely a typo, however, ANSI C permits this and results in x3 being declared by default as an int. Due to white space rules, this appears the same to the compiler as the following declaration:
	`unsigned long x1, x2;`
	`x3;`
	This is almost certainly *not* what the programmer intended, yet most C compilers do not issue a warning in these circumstances.

183	***Mixing function prototypes and old-style parameter list not allowed [NCC#183]*** Pre-ANSI-C C function declaration syntax is supported, but it is not recommended. Do not combine these two styles within a single function definition.
184	***No formal parameter matches the parameter declaration [NCC#184]*** This diagnostic results from an error of the form shown below, where there is no declaration for the parameter named **b**. `void f(a,b) int a; { <fn body> }`
185	***Invalid parameter declaration in function [NCC#185]*** This diagnostic results from certain errors in function definition syntax such as in the example below: `void f(int, long) { <fn body> }`
186	***Cannot have a 'timeout' pin on 'neurowire master' object [NCC#186]*** The **neurowire** slave I/O object declaration permits a **timeout** value. The **neurowire** master I/O object declaration does not.

NCC#	Description
187	***Expression must evaluate to a constant [NCC#187]*** Expressions in certain Neuron C declarations and initialization statements must evaluate to compile-time integer constants.
188	***Cannot modify a constant object [NCC#188]*** The Neuron C compiler enforces the **const** keyword strictly. In addition, data or objects declared using **const** might be placed in read-only memory areas by the compiler. However, **const network input** variables are not placed in read-only memory, because their values are updated by network variable messages from other devices. Furthermore, note that constant configuration parameters are placed in read-only memory unless the directive **#pragma codegen put_read_only_cps_in_data_memory** is used.
189	***Cannot modify via pointer-to-constant-object [NCC#189]*** To prevent data that is declared **const** from being modified, Neuron C will not permit constant objects to appear on the left-hand side of an assignment statement, nor will it permit modification of the constant object by a pointer with the **const** attribute, or by the **++** or **- -** operators. Note that, in the case of network variables, a network variable declared as **const** (or **config**, which implies **const**) cannot be modified in the device where it is so declared, but it *can* be modified by other nodes in the network.
190	***Object is not a suitable assignment target [NCC#190]*** The left-hand side of assignment operators, and the target of increment or decrement operators must be nonconstant variables, or fields of nonconstant structures or unions, or elements of arrays.

191	***Object of call is not a function [NCC#191]***
	The syntax encountered is function call syntax, for example:
	`expression ( [ expression-list ] )`
	however, the *expression* being called is not a function (or a pointer to a function). Note that this error could occur by omitting an operator.
	If the following were intended:
	`int a, b, c, d;`
	`a = b * (c + d);`
	but the following were actually written (omitting the multiplication operator):
	`int a, b, c, d;`
	`a = b (c + d);`
	This would appear to be a function call, but **b** is not a function.

NCC#	Description
192	***Call to function without prototype [NCC#192]*** Neuron C is more restrictive than ANSI C in this area. The Neuron Chip's hardware architecture does not permit calling undeclared functions with unknown numbers of parameters.
193	***Function does not allow parameters [NCC#193]*** The compiler outputs these diagnostics when the number of actual parameters or the actual parameter types, do not match those in the prototype, and they cannot be automatically converted.
194	***Not enough parameters passed to function [NCC#194]*** The compiler outputs these diagnostics when the number of actual parameters or the actual parameter types, do not match those in the prototype, and they cannot be automatically converted.
195	***Object cannot be a function parameter [NCC#195]*** Objects that have no type (such as message tags or I/O objects) cannot be function parameters. Likewise, if **p** were declared **void ***, ***p** would not be a valid function parameter (nor would it be any other valid expression, for that matter).
196	***Cannot convert address of const into non-const pointer [NCC#196]*** To prevent data that is declared **const** from being modified, Neuron C will not permit pointers including the **const** attribute from being cast such that the **const** attribute is removed. Neither does the compiler permit an implicit conversion of pointer (for example, through a function call) such that the const attribute would be removed. However, these changes are permitted (and this message will appear as a warning rather than an error) if the compiler directive **#pragma relaxed_casting_on** is specified in the program. See the *Compiler Directives* chapter of the *Neuron C Reference Guide* for more details.
197	***Implicit pointer conversion is not permitted [NCC#197]*** ANSI C does not permit a pointer of one type to be implicitly converted to a pointer of another type by assignment or by passing as a function parameter. Use explicit casting.
198	***Type mismatch in function parameter [NCC#198]*** The compiler outputs these diagnostics when the number of actual parameters or the actual parameter types, do not match those in the prototype, and they cannot be automatically converted.
199	***Too many parameters passed to function [NCC#199]*** The compiler outputs these diagnostics when the number of actual parameters or the actual parameter types, do not match those in the prototype, and they cannot be automatically converted.

NCC#	Description
200	***Type mismatch in assignment expression [NCC#200]*** These error diagnostics result from combining expressions of conflicting types, such as assigning an **int** to a pointer, or a pointer of one type to a pointer to another type, or in using objects that have no value (such as message tags or I/O object names) in expressions. In many cases in ANSI C, you must use an explicit type cast. However, note that casting should be avoided if possible, as it is often poor programming practice.
201	***Invalid use of pointer in binary operation [NCC#201]*** The only binary operations permitted on pointers are **+**, **-**, and comparisons. A pointer can be added to a constant (or vice versa), and ANSI C scaling rules apply to the constant. Likewise, a constant can be subtracted from a pointer (but *not* vice versa). Finally, two pointers of the same type can be subtracted, one from the other. The result is a difference scaled by the size of the object type pointed to. Pointers cannot be used in unary expressions other than with increment and decrement operators.
202	***Type mismatch for binary operation [NCC#202]*** This error can occur when the types of the operands being combined in the binary operation are not compatible. For example, adding an **int** to a structure, or comparing a pointer to a **char**, and so on.
203	***Invalid type for subscript operation [NCC#203]*** The object being subscripted (the "array") must be either an array or a pointer. The type of the subscript must be an integer type.
204	***Invalid type for array index [NCC#204]*** The array index of the subscript operator must be an **int** or **char** type. It may be **short** or **long**, **signed** or **unsigned**.
205	***Invalid operation on pointer [NCC#205]*** The only binary operations permitted on pointers are **+**, **-**, and comparisons. A pointer can be added to a constant (or vice versa), and ANSI C scaling rules apply to the constant. Likewise, a constant can be subtracted from a pointer (but *not* vice versa). Finally, two pointers of the same type can be subtracted, one from the other. The result is a difference scaled by the size of the object type pointed to. Pointers cannot be used in unary expressions other than with increment and decrement operators.
206	***Invalid indirection expression - not a pointer [NCC#206]*** This error occurs when the operand of the ***** indirection operator is not a pointer. This operator can only be applied to a pointer variable or a constant typed as a pointer.

NCC#	Description
207	***Invalid operand for address operator [NCC#207]*** The operand of the **&** address operator is not a variable, or is a variable type for which addressing is not permitted. For example, you cannot take the address of a numeric constant. In Neuron C, you also cannot take the address of a timer object, a message tag, an I/O object, or a functional block.
208	***The 'io_select' call for this device cannot specify a clock value [NCC#208]*** The **io_select( )** function permits an optional second parameter used to change the timer/counter internal clock setting (in a range from 0-7). However, this clock option can only be used when selecting an I/O object that permits a clock setting in that object's declaration. See the *I/O Model Reference* for more information.
209 **210**	***Bad type for operator [NCC#209]*** ***Bad type for conditional expression [NCC#210]*** These error diagnostics result from combining expressions of conflicting types, such as assigning an **int** to a pointer, or a pointer of one type to a pointer of another type, or in using objects that have no value (such as message tags or I/O object names) in expressions. However, note that casting should be avoided if possible, as it is often poor programming practice.
211 **212**	***Long constant value being converted to short [NCC#211]*** ***Possible data loss converting long to short [NCC#212*** These diagnostics result from an automatic conversion of a **long** variable to a **short**. To make these warnings go away, modify the variable declarations or use an explicit cast operator.
213	***Cannot use address or index operator with message object [NCC#213]*** The message objects, **msg_out**, **msg_in**, **resp_out**, and **resp_in**, have no value in themselves. They only have meaning when they are accessed using the dot operator (**.**) and a field name. Only certain predefined field names apply. The data field is an array, and access to it must be by index, except when used with **memcpy( )**.
214	***If one priority count is zero, both must be zero [NCC#214]*** The Neuron C application controls the counts and sizes of various buffers used in sending and receiving messages and network variable updates. There are two priority buffer pools, one at the network level, and one at the application level. Either or both pools must be empty, or both pools must contain buffers. A zero count cannot be specified for one priority pool and a nonzero count for the other.

NCC#	Description
215	***Array in struct or union must have bounds [NCC#215]*** An array declared at file scope (outside any other declarations or functions) may be declared without an explicit bound expression, provided an initializer is present. In ANSI C and in Neuron C, the compiler sets the array bounds implicitly by using the count of initial value expressions in the initializer list. However, this feature cannot be used with an array nested inside a structure or union declaration.
216	***Authenticated network variables require 'ackd' service type [NCC#216]*** Some of the options in the **bind_info** declaration modifier only apply to network variables, some only apply to output network variables, and some only apply to message tags. The service type declaration is required to be acknowledged when the authentication **bind_info** feature is used in a network variable declaration.
217	***Case value is out of range [NCC#217]*** Valid range is -128 to +127. A **switch** statement expression and the matching **case** label expressions are all of **int** type, which in Neuron C has the range shown.
218	***Use of Neuron C feature is not permitted [NCC#218]*** This message occurs when compiling a file with a .C extension. The Neuron C compiler will flag all uses of Neuron C features with this error message. Normally, a Neuron C program has a .NC extension.
219	***Pragmas 'hidden' and 'no_hidden' only allowed in 'echelon.h' [NCC#219]*** The pragmas **#pragma hidden** and **#pragma no_hidden** are for internal use from the standard include file **<echelon.h>** only. Do not use them elsewhere.
220	***Rate estimate value is out of valid range [NCC#220]*** The **bind_info** permits specification of average and maximum message rate estimates for each tag or network variable. The valid range of rate estimate values is from 0 to 18780, in units of *tenths* of a message per second. Thus, a specified value of 1043 indicates an actual value of 104.3 messages per second.
221 **222**	***Cannot have more than one default label per switch statement [NCC#221]*** ***Cannot have duplicate case labels with same value [NCC#222]*** A **switch** statement cannot have ambiguous labels. It can have no more than one **default** label, and no two **case** labels may have the same value.

NCC#	Description
223	***Improper function definition - missing parameter list [NCC#223]*** This message generally results from a syntax error of a specific kind. The compiler's syntax-directed parser is fooled by the error into thinking there is a function definition in progress, but the expected parameter list, in parentheses, which follows a function definition, is not found. For example: `static stuff i; // 'stuff' not defined` When a situation such as this arises, the compiler assumes 'stuff' is a function name, since it has not been previously defined. This results in a syntax error when 'i' is then read, since a function definition is supposed to be followed by a parameter list.
224	***Improper initializer format [NCC#224]*** A set of initializers in braces has too many levels of braces, or is otherwise incorrectly formulated. Initializers of aggregates (arrays, structures, or unions) should have a set of braces for each level of aggregate, but individual values should not have their own braces.
225	***Cannot set array bound from initializers [NCC#225]*** The C language provides two methods of specifying the bounds of an array. The first method, explicit bounds, uses a constant expression in brackets following the array name. The second method, implicit bounds, uses the number of initializers in the initializer list to automatically set the array bounds. This method indicates a problem in the initializer list such that the array bound cannot be automatically set.
226	***I/O object requires 'master' or 'slave' [NCC#226]*** The **neurowire** I/O object being declared must have either **master** or **slave** keywords after the **neurowire** keyword.
227	***Cannot have a 'select' pin on 'neurowire slave' object [NCC#227]*** The **neurowire** master I/O object declaration requires a **select** value. The **neurowire** slave I/O object declaration does not.
228	***The 'timeout' pin must be one of IO_0 ... IO_7 [NCC#228]*** The **neurowire** slave's **timeout** pin option can only be one of the pins **IO_0** through **IO_7**.
229	***Neurowire slave device cannot have a baud or kbaud specifier [NCC#229]*** The **neurowire** master device generates the clock used in the transfer. Therefore, specification of a bit rate in the declaration of a slave **neurowire** device is meaningless.

NCC#	Description
230	***Must specify 'enable_multiple_baud' prior to any I/O function [NCC#230]*** The **#pragma enable_multiple_baud** directive must appear prior to the use of any I/O function (for example, **io_in( )**, **io_out( )**). If this error message appears, move the **#pragma enable_multiple_baud** directive to the beginning of your program.
231	***Specify '#pragma enable_multiple_baud' for correct I/O operation [NCC#231]*** Two or more I/O devices have been declared with conflicting bit rates. In order for the compiler to generate correct code, it must know about the conflicting devices in advance. Thus, the **#pragma enable_multiple_baud** directive must be specified in advance.
232	***Hex escape char code constant is too large [NCC#232]*** A hex escape character inside a character or string constant exceeds the value **0xFF**. The Neuron C behavior is correct for ANSI C, but programmers usually find the ANSI C behavior in this respect counter-intuitive.
233	***Buffer size too small for network management messages [NCC#233]*** The compiler issues warnings when any of the buffer size pragmas are used, and the resulting settings would be too small to accommodate all possible network management messages from being properly received or responded to.
234	***Buffer size too small for interoperability [NCC#234]*** The compiler issues warnings when any of the buffer size pragmas are used, and the resulting settings would prohibit satisfying the interoperability criteria.
235	***Codegen buffer is full [NCC#235]*** The compiler memory limitation has been exceeded and the procedure must be split into two or more procedures.
236	***Too many static declarations in this compilation [NCC#236]*** A maximum of 32,767 static variables can be declared in a single module. Each configuration parameter (member of a CP family) also counts as a static variable.

NCC#	Description

237	***Unusual use of function address as value [NCC#237]***
	This message would occur in a situation like the following:
	`int f(void) {return 0;}`
	`...`
	`int g(void) {`
	`int x;`
	`x = 1;`
	`if (f) {  // Unusual use of function`
	`  // address`
	`  x = 2;`
	`}`
	`return x;`
	`}`
	Technically, C permits such a use as shown. Such an expression has little use in Neuron C, however. The programmer most likely wanted to specify "if (f( )) { ...". In other words, the syntax of ANSI C permits a construct that is most likely a programming error, and the Neuron C compiler flags it for you.

238	***File write error - is disk full? [NCC#238]*** The compiler encountered an error writing to the output file(s). Check that the output media is not write-protected, and that sufficient disk space exists. It is possible, for extremely large programs, that a megabyte or more of temporary disk space would be needed during compilation.
239	***A msg_tag declaration is not permitted if micro_interface [NCC#239}*** Once the **#pragma micro_interface** appears, the program cannot declare any network variables or message tags.
240	***This event expression is not permitted - firmware restriction [NCC#240]*** The special event keywords **offline**, **online**, and **wink** cannot be combined into other expressions when used in the **when** clause. See the *Additional Predefined Events* section in the *Neuron C Programmer's Guide* for more explanation.
243	***Keyword 'sync' was ignored for polled output network variable [NCC#243]*** A **sync** output network variable is propagated each time its value is updated. A **polled** output network variable is never sent unless a reader device requests its value with a network variable poll. The two options are not compatible.

NCC#	Description
244	***Cannot disable netvar_processing with Net Vars declared [NCC#244]*** The directive **#pragma netvar_processing_off** is not permitted once network variables have already been declared in a program. Likewise, declarations of network variables are not permitted in a program once this directive has been encountered. This pragma can only be used with the LonBuilder Microprocessor Interface Program (MIP).
245	***Network variable declaration not permitted if netvar_processing off [NCC#245]*** Once the directive **#pragma netvar_processing_off** is encountered in a program, network variable declarations are not permitted. This pragma can only be used with the LonBuilder Microprocessor Interface Program (MIP).
246	***This I/O object type requires specification of a 'timeout' pin [NCC#246]*** The I/O object being declared requires specification of an additional pin to use as an external timeout signal. See the description of the I/O object being declared in the *I/O Model Reference* for more information.

NCC#	Description
247	***Statement deleted by optimizer [NCC#247]***

The Neuron C compiler's optimizer deletes statements for a variety of reasons. For example, the statement may represent unnecessary work, the statement may represent dead (unreachable) code, or the optimizer has combined the statement with another statement. This informatory message is issued for two reasons. First, to let the programmer know why a breakpoint cannot be set on what looks like an acceptable statement. Second, to point out code that may never be executed due to erroneous program logic. Consider the following examples:

```
<statements>
goto LABEL;
<statements>   //deleted by optimizer
LABEL:
<statements>
```

Since the **goto** statement causes a branch around code such that it can never be executed, the compiler issues a "Statement deleted" message for each statement. The last example demonstrates an error in program logic discovered by the compiler. The next example demonstrates unnecessary statements deleted by the optimizer:

```
int i;
switch (i) {
case 0:
   <statements>
break;
case 1:
   <statements>
break;
case 2:
   <statements>
break;      // deleted by optimizer
}
```

In the above example, the last **break** statement in the switch statement is unnecessary, since execution flow is the same with or without the **break** statement. (However, it is recommended that the statement be coded as shown anyway, because doing so will make future maintenance of the code easier and less error-prone. It is good programming practice.)

The optimizer recognizes that any branch is unnecessary and eliminates the unnecessary branch instruction. The informatory message is reported to explain why a breakpoint cannot be set at what appears to be a valid statement. No breakpoint can be set, since no code actually exists for this statement. Note that informatory messages are not reported by the compiler unless they are enabled by the **#pragma fyi_on** directive.

NCC#	Description

248	***Comparison is ineffective - result of comparison is a constant [NCC#248]***
	A conditional expression (comparison) which involves a constant value that is out of the range of a variable value is an ineffective comparison; one that always resolves to either the constant TRUE or the constant FALSE. The Neuron C compiler detects such a condition and issues a warning assuming that the comparison might be erroneous. This situation most commonly occurs when an **unsigned short** is compared with a negative number, or a **short** is compared with a constant that is a **long**. Recall that Neuron C defines **short** to be 8 bits and **long** to be 16 bits.
	`void f(void) {`
	`  int i;`
	`  if (i < 300) {`
	`  // The comparison shown above is an`
	`  // ineffective comparison, since 'i'`
	`  // can only be in the range -128..127.`
	`  // The constant 300 is a long.  This`
	`  // is most likely a programming error.`
	`  }`
	`}`

249 **250**	***Loop, branch, or 'when' condition is always TRUE [NCC#249]*** ***Loop, branch, or 'when' condition is always FALSE [NCC#250*** The Neuron C compiler detects when some conditions are always FALSE or always TRUE. This warning is generated to help programmers ensure that their program is correct. A condition that is always FALSE, or TRUE, may result from an erroneous conditional expression.

NCC#	Description

251	***Assignment operator at top level of conditional expression [NCC#251]***
	This warning is issued by the Neuron C compiler when it detects a use of the assignment operator **=** in the top level of a conditional expression, for example, in an **if** statement.
	`int a, b;`
	`.`
	`.`
	`.`
	`if (a = b)   // This is an assignment`
	Although such a code construct is legal and useful in C (it assigns the value of 'b' to 'a' and tests that value for nonzero, simultaneously), it is also one of the most common coding errors in C programs for novice and experienced programmers alike. Almost all C programmers find themselves occasionally making this error when an equality operator **==** is actually intended.
	The purpose of this warning is to assist programmers in finding programming errors. The warning may be silenced by recoding the conditional expression for an explicit test against zero. The Neuron C compiler's optimizer will produce identical code with or without the explicit test, due to special optimization logic for this case, thus the explicit test will not decrease program efficiency.
	`if ((a = b)!= 0)`

252	***Use of 'snvt_si_eecode' and 'snvt_si_ramcode' are exclusive [NCC#252]*** The compiler directives **#pragma snvt_si_eecode** and **#pragma snvt_si_ramcode** are mutually exclusive, since the two directives cause the compiler to force the placement of the SNVT/Self-Identification information in mutually exclusive areas of memory.
253	***Triac level I/O object cannot use the 'clockedge(+-)' option [NCC#253]*** The **triac** I/O object in level mode can only use the **clockedge(+)** or **clockedge(-)** option.
254	***Triac I/O object declaration defaults to triac 'pulse' object [NCC#254]*** Either **pulse** or **level** mode can be selected explicitly using the appropriate keyword in the I/O declaration for a **triac** object. A declaration without an explicit keyword will default to using **pulse** mode. To silence the warning, modify the declaration to explicitly specify the **triac** object's mode of operation.
255	***One or more unterminated #if/#ifdef/#ifndef directives [NCC#255]*** The preprocessor directives controlling conditional compilation must always exist in matching pairs, similar to the open brace **{** and close brace **}** in a C program. The pairs **#ifdef** and **#endif** must match, for example, with an optional **#else** contained in between.

NCC#	Description
256	***Attempt to #undef a name which is not a macro [NCC#256]*** The preprocessor **#undef** command can only be applied to an identifier which has previously been defined as a macro using the **#define** command.

257	***Cannot redefine typedef name at file scope [NCC#257]***
	The rules of ANSI C permit redefinition of a **typedef** inside a nested scope (for example, in a function), but not at file scope. For example:
	`typedef unsigned int ui;`
	
	`// The following redefinition is not permitted`
	`unsigned short ui;`
	
	`void f (void) {`
	`  // The following defines ui as a variable,`
	`  // which hides the typedef inside the function`
	`  unsigned short ui;`
	`}`
	
	`// The typedef ui is now no longer hidden`
	`ui x;`

258	***Conditional compilation directives nested too deeply [NCC#258]*** The maximum nesting level of the **#ifdef/#ifndef/#if** directives is 16. Change your conditional compilation strategy if you are attempting to use more than 16 levels of nested directives.
259	***Misplaced #elif/#else directive [NCC#259]*** The directive in question does not follow the appropriate **#ifdef**, or **#ifndef** directive.
260	***Too many arguments in macro being defined [NCC#260]*** The maximum number of macro arguments that Neuron C supports is 16.
261	***Macro argument name cannot be repeated [NCC#261]*** Function-style macros (those which have a parameter list) must use a unique name for each parameter.
262	***Incorrect number of arguments for macro [NCC#262]*** The invocation of the macro in question does not supply the correct number of arguments. The number of arguments must be the same as the number of formal parameters in the definition of the macro.
263	***Array is too large for fastaccess feature [NCC#263]*** An array declared with the optional **fastaccess** feature cannot exceed a total size of 254 bytes.

NCC#	Description
264	***The fastaccess feature only applies to arrays [NCC#264]*** The optional **fastaccess** feature should only be used in declarations of array variable types. The feature does not apply to the indexing operator applied to pointers.
265 **266** **267**	***The stack frame of this procedure is too large (>200 bytes) [NCC#265]*** ***The stack frame of this procedure exceeds 100 bytes [NCC#266]*** ***The stack frame of this procedure exceeds 7 bytes [NCC#267]*** On Neuron Chips, references to variables near the top of the stack use the most efficient instructions. The further down in the stack one goes, the less efficient the instructions become. This inefficiency affects both the code size and the code performance. A stack frame larger than seven bytes begins to incur this penalty. The stack frame includes the parameters and the local variables. Neuron Chips do not have very large memory areas set aside for stacks. Procedures whose stack frame exceed 200 bytes would fail to work; therefore, this is a compile-time error. Procedures with stack frames in excess of 100 bytes are flagged with a special warning message because they use more than half of the stack resources, and nesting these functions would cause a stack overflow. The compiler does not specifically check for nesting, but it attempts to use this warning to catch large procedures. Note that the Neuron chip's hardware architecture suggests preferring a large number of small functions over fewer, but larger, functions. Smaller functions typically lead to much smaller stack frames for each function, which also allows for more efficient code and thus results in a smaller code footprint. See Chapter 8 of the *Neuron C Programmer's Guide* for more information on managing memory resources.
268	***Recommend use of an unqualified 'msg_arrives' event [NCC#268]*** A program which receives explicit messages through the **msg_arrives** event will be given all such messages which come into the device, whether their message codes are expected or not. Any unexpected messages must be handled by the program through a "catch-all" unqualified **msg_arrives** event, otherwise such messages will get stuck at the head of the message queue. See the chapter on messages in the *Neuron C Programmer's Guide* for more information on processing incoming messages.

NCC#	Description
273	***Procedure code generation label resources exhausted [NCC#273]*** Code generation is performed on a procedure-by-procedure basis. The compiler reuses certain internal resources during code generation of each procedure. One of these resources is internal label markers. There are only a limited number of labels that can be used in a given procedure. These labels are used in each possible branching scenario, in loops, **if** statements, **?:** operators, and **switch** statements. By far the most common cause of this message is a very large procedure containing a **switch** statement with many cases. The simplest recourse is to split the **switch** statement into two or more **switch** statements, and move each to a separate subprocedure.
274	***Use of possibly unitialized variable [NCC#274]*** The Neuron C compiler tracks the use of automatic variables (those which are local to a function or procedure, or a sub-scope of a function or procedure). If such a variable is accessed (read) prior to its having been stored (written), this warning is issued. Structure fields and array elements are not individually tracked.
275	***Recommend use of 'fails' or 'succeeds' event instead [NCC#275]*** This message indicates that the call to a completion event can be changed to a fails or succeeds event as an efficiency consideration. See the discussion on events in the *Neuron C Programmer's Guide* or the *Neuron C Reference Guide*.
276	***The 'preempt_safe' keyword has no effect on this 'when' clause [NCC#276]*** Some **when** clauses and their associated tasks will be executed regardless of preemption mode. Use of the **preempt_safe** keyword for this type of **when** clause is unnecessary, and has no effect.
291	***Improper binary constant '<value>' [NCC#291]*** A binary constant begins with 0b and is followed by one or more binary digits (0 or 1).
292	***Incomplete hexadecimal constant '<value>' [NCC#292]*** A hexadecimal constant begins with 0x and must be followed by one or more hexadecimal digits (0-9 and the letters a-f or A-F).
293	***Improper octal constant '<value>' [NCC#293]*** An octal constant begins with 0, and must be followed by one or more octal digits (0-7).

NCC#	Description
294	***Unterminated character constant: '<value>' [NCC#294]*** The proper format of a character constant is '*char*'. The '*char*' can either be a single character (except another quote), or any of a number of ANSI C escape sequences. Consult a basic text on ANSI C, such as one of those listed in the *Preface* of the *Neuron C Reference Guide*, for more information.
295	***Integer constant '<string>' is too large [NCC#295]*** Integer constants are limited to 5 characters, with a value of 65535 or less, since the maximum size of an integer is 16 bits.
297	***Cannot open <filetype> file: '<filename>' [NCC#297]*** The 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.
298	***Recursive include file nesting (file '<filename>') not allowed [NCC#298]*** An include file cannot include itself, nor can any file B included from file A then re-include file A, and so on.
299	***Unsupported preprocessor directive: '<name>' [NCC#299]*** The following ANSI C preprocessor directives are *not* supported in Neuron C: **#elif** **#file** **#if** **#line**
300	***Access to stack variable is beyond end of stack [NCC#300]*** The indicated fetch or store to a local variable or parameter on the stack is beyond the possible end of the Neuron's stack. No instruction can be generated for the indicated fetch or store. This could occur with an array variable on the stack being indexed beyond the array bounds. This could also occur as a result of incorrect direct address calculations on a stack variable, such as a structure.
301	***Invalid use of reserved word >> <token> << [NCC#301]*** This error message occurs when the token causing the syntax error is a reserved word (keyword) that is used out of context. Check the syntax of not only the reported token, but also the syntax of a few of the previous tokens. (A token is a grammatical unit, such as a keyword, or a comma or semicolon.) Don't forget that Neuron C has some extra reserved words, in addition to those defined by ANSI C (these are listed in the *Reserved Words* appendix in the *Neuron C Reference Guide*). Check that you haven't accidentally used a reserved word as a variable name or other identifier.

NCC#	Description
302	***Syntax error when reading >> <token> << [NCC#302]*** Any syntax error is reported in this manner. Check the syntax of not only the reported token, but the last few previous tokens. The Neuron C grammar is explained in the *Syntax Summary* appendix in the *Neuron C Reference Guide*.
303	***Macro text for '<name>' is too long for debug info [NCC#303]*** The Neuron C debugger can understand macro names, but can only handle the first 16Kbytes of text used in the definition of a macro.
304	***Invalid typedef id '<name>' [NCC#304]*** Reference to symbol <*name*> seemed to be a reference to a **typedef** identifier, but no such **typedef** was declared.
305	***Integer constant '<value>' is too large [NCC#305]*** Integer constants are limited to 65535, since the maximum size of an integer is 16 bits.
306	***Symbol '<name>' is not defined [NCC#306]*** An identifier was used in an expression that was not previously declared or defined. ANSI C requires that all identifiers be declared before their first use.
307	***Symbol '<name>' is a restricted symbol [NCC#307]*** The <*name*> shown cannot be used for a user-declared identifier, macro, and so on. All such restricted names begin with '_', although not all names beginning with the '_' character are reserved. To avoid this problem, as well as to avoid future compatibility problems, don't declare any names beginning with the '_' character.
308	***Event conflict for I/O object '<name>' [NCC#308]*** A single I/O object cannot be used in both an **io_update_occurs** event *and* an **io_changes** event.
309	***Incomplete binary constant '<token>' [NCC#309]*** A binary constant begins with 0b and must be followed by one or more binary digits (0 or 1).

NCC#	Description
310	***The symbol '<name>' was declared but never used [NCC#310]*** The compiler issues this warning for any run-time object that is declared, but never used in the executable code. Run-time objects include anything that consumes Neuron memory or other run-time resources, such as I/O objects, variables, functions, timers, and so on. No warning is issued for compile time objects, such as typedefs, structure tags, and so on, which are not used. Some objects may be intentionally declared, but unused. If it is desired to suppress this warning for a certain symbol, the following pragma may be inserted following the declaration of the object in question: `#pragma ignore_notused` *`symbol`* This directive may be used multiple times, for each symbol for which the warning should be suppressed.
311 **312** **313**	***Redefinition of '<macroname>' hides function [NCC#311]*** ***Redefinition of '<macroname>' hides enum value [NCC#312]*** ***Redefinition of '<macroname>' hides label [NCC#313]*** The macro being defined, by the rules of ANSI C, will supersede any other declarations of the same name. This may not be what the programmer intended, so if a conflict is detected, to help with detection of inadvertent programming errors, the above warning(s) will be printed.
314	***Cannot redefine '<name>' [NCC#314]*** The name has been declared as a certain type of identifier more than once in the scope. For example, it is illegal to define a variable twice at file scope, or a function, or a macro, and so on (however, there may be multiple **extern** declarations for the same variable). Note that a macro definition is always considered to be at file scope, regardless of its placement in a file.
315 **316**	***Name of network variable, '<name>', exceeds 16 chars [NCC#315]*** ***Name of msg_tag, '<name>' exceeds 16 chars [NCC#316]*** Any network variable or message tag name is limited to 16 characters. Likewise, if a **typedef** name is used in declaration of a network variable, it too must be 16 characters or less in length.
317	***Label '<name>' is not defined [NCC#317]*** The specified label used in a **goto** statement is not defined in the current procedure or task.

NCC#	Description
318	***Improper context for '<keyword>' label [NCC#318]*** Certain statements in ANSI C do not have meaning except within some defined construct. The **continue** statement can only be used inside a loop statement, which is either a **for**, a **while**, or a **do-while**. The **break** statement can only be used inside a loop statement or inside a **switch** statement. The words **case** and **default** are reserved words in ANSI C, used as labels inside of the scope of a **switch** statement. They cannot be used outside of a **switch** statement. This message indicates that the program being compiled has violated one of these rules. Consult a basic text on ANSI C, such as one of those listed in the *Preface* of the *Neuron C Reference Guide*, for more information.
319 **320** **321** **322** **323** **324**	***Need to check nv_update_succeeds for <nv-name> [NCC#319]*** ***Need to check nv_update_succeeds for <nv-name>[<index>] [NCC#320]*** ***Need to check nv_update_fails for <nv-name> [NCC#321]*** ***Need to check nv_update_fails for <nv-name>[<index>] [NCC#322]*** ***Need to check msg_succeeds for tag <tag-name> [NCC#323]*** ***Need to check msg_fails for tag <tag-name> [NCC#324]*** Some events must occur in pairs. For a given network variable or message tag, checking one event requires checking of a corresponding event, to have a correct program.
325	***The 'num_addr_table_entries' was adjusted upward to <value> [NCC#325]*** If there are one or more **msg_tag** declarations, or network variables are declared, the compiler computes a minimum allowable number of address table entries. There must be one entry per bindable **msg_tag** declared, plus at least one entry if network variables are used. For some possible connections, this may not be enough entries. However, this *value* is an absolute minimum.
330	***Symbol Table is full [NCC#330]*** The compiler symbol table limit has been reached.
331	***Optional parameters are not supported [NCC#331]*** The standard C feature of a variable argument list through the ellipsis, also called optional function parameters, is not supported in Neuron C.
332	***Problem reading 'snvt.typ' [NCC#332]*** This error is not applicable in version 4.00 or later versions of the Neuron C Compiler.

NCC#	Description

334	***Specify 'idempotent_duplicate_<off,on>' pragma with 'micro_interface' [NCC#334]***
	
	The Microprocessor Interface Program option for a Neuron C compilation requires specification of one of the following two pragmas:
	
	`#pragma idempotent_duplicate_off`
	`#pragma idempotent_duplicate_on`
	
	For more information, see Chapter 1 of the *Neuron C Programmer's Guide*.

335	***Byte/Nibble output has no effect on timer/counter output pins [NCC#335]*** A timer/counter output device has precedence over a **byte** or **nibble** output device. The pin used by the timer/counter device, either **IO_0** or **IO_1**, will not be affected by any output operations on the **byte** or **nibble** device. However, the remaining pins of the **byte** or **nibble** device will still function. The declaration is permitted for situations such as a timer/counter device on pin **IO_0**, and a 7-bit output device on pins **IO_1** through **IO_8**, which could use a byte device declared on pin **IO_0** to accomplish the function.
336	***Struct assign for EEPROM dest limited to 255 bytes [NCC#336]*** The Neuron Chip firmware functions that copy blocks of memory to EEPROM destination addresses support a maximum length of 255 bytes per copy. Larger blocks can be copied by using multiple calls, substructure assignments, and so on. See Chapter 8 of the *Neuron C Programmer's Guide* for more information on managing memory resources.
337	***Priority bind_info options ignored for NV '<name>' [NCC#337]*** Any explicit priority **bind_info( )** options for network variables are ignored when the number of priority buffers is zero. The number of priority buffers is set to zero explicitly by the priority buffer count pragmas (see the *Compiler Directives* chapter of the *Neuron C Reference Guide* and Chapter 8 of the *Neuron C Programmer's Guide* for more information).
338	***The '#pragma codegen' directive must precede affected code [NCC#338]*** The **codegen** pragma (see the *Compiler Directives* chapter of the *Neuron C Reference Guide*) affects code generation for certain Neuron C features. Selection of a **codegen** option must precede any generated code that would be affected by the option. It is best to place these pragmas at the beginning of a program.

NCC#	Description
339	***This program requires aliases: '#pragma num_alias_table_entries' [NCC#339]*** The program has referenced a firmware or library function which operates on or requires one or more alias table entries, but the pragma which allocates these alias table entries was not supplied, or the number of alias table entries was specified as 0 (zero). See the *Compiler Directives* chapter of the *Neuron C Reference Guide* for more information on the pragma.
340	***Parameter to 'propagate' must be output network variable [NCC#340]*** The built-in function **propagate( )** takes as its only argument the name of an output network variable. See the *Functions* chapter of the *Neuron C Reference Guide* for more information on this built-in function.
342	***I/O object type restricted to pins IO_0 through IO_6 [NCC#342]*** Different I/O object types are permitted on different subsets of the Neuron Chip's I/O pins. For more information, see the *I/O Objects* chapter of the *Neuron C Reference Guide*.
343	***The '#pragma set_std_prog_id' conflicts with ID set via compile option [NCC#343]*** The compiler permits the program ID to be set in various ways, by compiler directive (pragma) as well as by command-line or programmatic interface option. The compiler will tolerate multiple attempts to set the program ID, provided there is no conflict.
345	***Array index is out of range of bounds declaration [NCC#345]*** This warning is generated whenever a constant index is applied to an array such that the index is negative, or is beyond the declared bounds of the array.
346	***This combination of debug options is not available [NCC#346]*** The **#pragma debug** directive can be specified a number of times in a given program, with various options. Some of these options can be combined, and others cannot. Consult the *Compiler Directives* chapter of the *Neuron C Reference Guide* for a complete explanation of the directive.
347	***Need at least 3 application input buffers with debug kernel [NCC#347]*** In order for the network debug kernel to function properly, a device must have at least 3 application input buffers. The program being compiled with the network debug kernel option selected does not have at least 3 application input buffers.

NCC#	Description
349 **350**	***Read error on cached file [NCC#349]*** ***Write error on cached file [NCC#350]*** A read error or a write error was reported while accessing a file. Check for adequate disk space, or the possibility of loss of network connection (if a networked file), or removal of removable media.
352	***Array size exceeds 65535 [NCC#352]*** The array variable being declared exceeds a total size of 65535 bytes. No array, struct, or union variable in Neuron C can exceed 65535 bytes.
353	***Variable being declared is too large for RAMNEAR. Use 'far'. [NCC#353]*** The variable being declared is larger than 256 bytes in size. The total available size of the RAMNEAR area in the Neuron is 256 bytes, thus this variable declaration *must* be placed in RAMFAR. Use the **far** keyword in the declaration ("near" is the default). See Chapter 8 of the *Neuron C Programmer's Guide* for more information.
354	***Variable being declared is too large for EENEAR. Use 'far'.*** The variable being declared is larger than 255 bytes in size. The total available size of the EENEAR area in the Neuron is 255 bytes, thus this variable declaration *must* be placed in EEFAR. Use the **far** keyword in the declaration ("near" is the default). See Chapter 8 of the Neuron *C Programmer's Guide* for more information.
357	***The 'uninit' storage class requires use of 'eeprom' or 'config' or 'cp' [NCC#357]*** The **uninit** storage class is used to tell the compiler not to provide any initial value for the data item being declared, thus the loader will not overwrite that area of memory when reloading the program. This feature is only available for data items using EEPROM or Flash memory (the **uninit** storage class does not apply to RAM variables). The **uninit** feature requires the declaration to use one of the storage classes **eeprom**, **config**, or **config_prop** (abbreviated **cp**). See the *Neuron C Programmer's Guide* for more information.

NCC#	Description
358	***The 'offchip' storage class requires use of 'far' [NCC#358]*** The Neuron Chip provides a “near” area of EEPROM memory and a "near" area of RAM memory. Each of the “near” areas is located **onchip**, and “near” is the default memory area used by a data declaration. When writing a program for a Neuron 3150, which has external (off-chip) memory, the **offchip** keyword can be used in the data declaration to force the linker to place the data item in **offchip** memory. Since the data declaration would default to the near area, which is located on-chip, the **far** keyword must also be used in the data declaration. Add the **far** keyword to the declaration. See Chapter 8 of the *Neuron C Programmer’s Guide* for more information on managing memory resources.
359	***The keywords 'offchip' and 'onchip' are mutually exclusive [NCC#359]*** As the message states, at most one of these storage classes may be used in a data declaration.
387	***File’s basename exceeding 52 characters may cause linker problems [NCC#387]*** In certain situations, including the use of configuration parameters and/or variables declared static, the compiler may construct a “made-up-name” for the variable that is, in part, based upon the basename portion of the filename of the file that is being compiled. (The basename is the part of the filename preceding a "**.**" character.) Because the maximum length of a symbol in the compiler, assembler, and linker is 64 characters, and taking into account certain additional characters added by the compiler in the process of creating a "made-up-name", if the basename of the file exceeds a length of 52 characters, the symbols passed to the assembler and linker may be too long, and link errors may result. To avoid this problem, limit the basename of the file to 52 characters or less.
388	***System error in Device Resource Files access [NCC#388]*** The Neuron C compiler for the Neuron C Version 2 language uses the LONMARK® Device Resource Files, including the catalog, and files such as *.FPT, *.TYP, and so on. The Neuron C compiler uses the services of the Device Resource Files API (DRF API) to provide access to the Resource Files. If the DRF API reports an unexpected problem, the compiler prints this message and stops the compilation. Some possible causes of this problem are incorrect filenames or directory paths in the catalog. Perform a catalog refresh to correct this situation. (The IzoT Commissioning Tool will automatically refresh the catalog when it starts up, as will the NodeBuilder Resource Editor).

NCC#	Description
389	***Typename '<name>' not found in Device Resource Files; 'SNVT*', 'SCPT*', "UNVT*", "UCPT*' are reserved [NCC#389]*** The Neuron C Compiler for the Neuron C Version 2 language uses the LONMARK Device Resource Files to resolve all names beginning with the prefixes 'SNVT*', 'SCPT*', 'UNVT*', and 'UCPT*', found in a Neuron C program. The program should avoid using any names beginning with any of these prefixes, for compatibility with names in the Resource Files. Programs that were originally written in Neuron C Version 1 and use **typedef** to define SCPT, UNVT, and UCPT types can still be compiled by using the **#pragma names_compatible** compiler directive. In this case, and matching type names in resource files will be hidden. See the *Compiler Directives* chapter of the *Neuron C Reference Guide* for more information. For more information about compiling legacy Neuron C applications with the more recent version of the Neuron C Compiler (NCC version 4.0 or later), and more information about converting a legacy application into one that uses the Neuron C Version 2 language LONMARK features, please refer to the IzoT *NodeBuilder FX User's Guide*.
390	***Could not open the LmRF catalog [NCC#390]*** LmRF catalog means LONMARK Resource File catalog. The Neuron C compiler for the Neuron C Version 2 language uses the LONMARK Device Resource Files, including the catalog, and files such as *.FPT, *.TYP, and so on. The Neuron C compiler uses the services of the Device Resource Files API (DRF API) to provide access to the Resource Files. If the DRF API cannot open the catalog, the compiler prints this message and stops the compilation. Possible causes of this problem are a missing \Lonworks\Types folder, or a missing or corrupt catalog file (LDRF.CAT which is stored in the Types folder). Perform a catalog refresh to correct this situation. The IzoT Commissioning Tool will automatically refresh the catalog, or create one if necessary, when it starts up; as will the NodeBuilder Resource Editor.
391	***Use of NVT or CPT 'float' type requires prior #include <float.h> [NCC#391]*** Whenever a Network Variable Type (NVT) or a Configuration Property Type (CPT) is retrieved from a LONMARK Device Resource File by the Neuron C Compiler, and the type contains one or more references to the **float_type** definition, the compiler checks that the floating point support include file has been included in the compilation. Add **#include <float.h>** to the beginning of your program.

NCC#	Description
392	***Use of NVT or CPT 'quad' type requires prior #include <s32.h> [NCC#392]*** Whenever a Network Variable Type (NVT) or a Configuration Property Type (CPT) is retrieved from a LONMARK Device Resource File by the Neuron C Compiler, and the type contains one or more references to the **s32_type** definition, the compiler checks that the signed 32-bit support include file has been included in the compilation. This situation will occur when using any CPT of inherited type, because prior to inheritance of type information, all inherited type CPTs default to the 32-bit signed type **s32_type**. This is true even if the program does not use any signed 32-bit data or arithmetic support functions. Add **#include <s32.h>** to the beginning of your program.
393 **394** **395** **396**	***Cannot add file record to dependency file (might cause build status calculation to fail) [NCC#393]*** ***Cannot add switch record to dependency file (might cause build status calculation to fail) [NCC#394]*** ***Cannot add parameter record to dependency file (might cause build failure) [NCC#395]*** ***Cannot write .ncdep dependency file (might cause build status calculation to fail [NCC#396]*** These problems are reported by the dependency utility. Although it will not stop the compilation, the warning alerts the user to a potential problem the next time a Project Make (or a build) is run, because the Project Make utility may not be able to correctly calculate whether the project will need to be rebuilt. To clear this condition, try a "clean" followed by an "unconditional build".
397	***Reference in NVT or CPT not found [NCC#397]*** Some Network Variable Types (NVTs) or Configuration Property Types (CPTs) can reference other NVTs. References can be nested. This message indicates that the original type, or one of the types that it references, references another type that does not exist in the applicable and available resource files. Use the NodeBuilder Resource Editor to examine the original type, and then to observe whether all the types it references do, in fact, exist. This problem may be caused by any of the following: one or more resource files are missing, or the catalog does not list these resource files, some resource files may be out of date, or possibly a resource file has been mis-edited.
398 **399**	***Unspecified error in option processing [NCC#398]*** ***Unspecified error in execution of compiler [NCC#399]*** These messages result from an unknown program failure or anomaly that has been caught by self-checking code in the compiler.

NCC#	Description
401	***Repeated declaration of CP family does not match previous declaration [NCC#401]*** In support of code modularity, it is possible that more than one file in a compilation may declare a CP family of the same name, and with identical properties. Rather than force the reorganization of the code or the creation of artificially cryptic names, the compiler supports multiple identical declarations, and will merge these declarations into a single declaration of the family. This feature requires the two (or more) declarations being merged to be identical in every aspect.
402 **403**	***Invalid reference '<name>' – must be an NV-CP or a CP family name [NCC#402]*** ***The property '<name>' is not an NV-CP or a CP family name [NCC#403]*** The compiler has determined that the <name> shown must be a reference to a configuration property. The configuration property must have previously been declared as a network variable using the **config_prop** or **cp** option keyword in the declaration, or as a configuration parameter family using the **cp_family** keyword. For more information on this topic, see the chapter on configuration properties in the *Neuron C Programmer's Guide*.
404 **405**	***A device property cannot be 'static' [NCC#404]*** ***A device property cannot be 'global' [NCC#405]*** The **device_properties** list cannot contain any properties using the **static** or **global** keyword as a modifier. Device properties are unique to the device, and cannot be shared.
406	***Device property is a duplicate [NCC#406]*** The *LONMARK Application Layer Interoperability Guidelines* specify that no more than one property of any particular SCPT or UCPT type may be used as a device property. CPT names are the keys that LNS uses to retrieve variable values. Consult that document for more information.

NCC#	Description

407	***A variable property list can only be used with a network variable [NCC#407]***
	The Neuron C Version 2 syntax permits nonsense declarations like the following examples, but the compiler later determines that the **nv_properties** clause can only apply to a network variable declaration.
	Improper uses of the **nv_properties** clause:
	`static int abc nv_properties { heartbeatTime };`
	`SNVT_temp_f temperature`
	`  nv_properties { heartbeatTime };`
	Proper use of the **nv_properties** clause:
	`network output SNVT_temp_f temperature`
	`  nv_properties { heartbeatTime };`
	In the second "improper use" example above, the declaration may be confused with a network variable declaration, however, it is not. The declaration actually is just a variable declaration using the type of the SNVT (this is a legitimate declaration, but not a network variable, so it cannot have a property list).

408	***Array index for property or member cannot be used in this context [NCC#408]*** A reference to a property name in a property list contained an array index expression, but the property is not an array.

409	***A 'cp' network variable cannot have a variable property list [NCC#409]***
	The Neuron C Version 2 syntax permits nonsense declarations like the following example, but the compiler later determines that the **nv_properties** clause can only apply to a network variable declaration that is *not* a configuration property. The *LONMARK Application Layer Interoperability Guidelines* do not permit a configuration property to have properties of its own.
	Improper use of the **nv_properties** clause:
	`network input cp SCPTdefOutput defaultValue`
	`  nv_properties { heartbeatTime };`

410	***A 'config' network variable cannot have a variable property list [NCC#410]***
	The Neuron C Version 2 syntax permits nonsense declarations like the following examples, but the compiler later determines that the **nv_properties** clause can only apply to a network variable declaration that is *not* declared with the **config** keyword. The *LONMARK Application Layer Interoperability Guidelines* do not permit a configuration property to have properties of its own.
	Improper use of the **nv_properties** clause:
	`network input config SCPTdefOutput defaultValue`
	`  nv_properties { heartbeatTime };`

NCC#	Description
411	***Existing LonMark SD string info in NV will be overridden [NCC#411]*** When a Neuron C program uses any of the new Neuron C Version 2 features that support the construction of a LONMARK device, the compiler will create the SD and SI information for the device. When the compiler attempts to insert this information into the SI and SD strings, if there is already text present from the user's program, the compiler will insert the LONMARK information at the front of the string, and will then use a semicolon character to separate the automatically-generated string from the user-specified string. If the compiler detects that the existing string information started with the special characters that would identify that information as LONMARK information, the compiler then issues this warning, to let the programmer know that the existing LONMARK information in the program was overridden.
412	***Too many errors – compilation halted [NCC#412]*** To prevent a serious error early in the compilation, such as a missing include file, or a missing brace or parenthesis, from creating a flood of useless error messages, the compiler limits the number of error messages reported to a small number. The next error after this limit is reached causes the compiler to issue this message and halt the compilation.

417	***Invalid declaration type for 'properties' clause [NCC#417]***
	
	The Neuron C Version 2 syntax permits nonsense declarations like the following examples, but the compiler later determines that the **nv_properties** clause can only apply to a network variable declaration.
	
	Improper uses of the **nv_properties** clause:
	
	`static int abc nv_properties { heartbeatTime };`
	`SNVT_temp_f temperature`
	`  nv_properties { heartbeatTime };`
	
	Proper use of the **nv_properties** clause:
	
	`network output SNVT_temp_f temperature`
	`  nv_properties { heartbeatTime };`
	
	In the second "improper use" example above, the declaration may be confused with a network variable declaration, however, it is not. The declaration actually is just a variable declaration using the type of the SNVT (this is a legitimate declaration, but not a network variable, so it cannot have a property list).

NCC#	Description

418	***Cannot have multiple 'properties' clauses on a variable [NCC#418]***
	The Neuron C Version 2 syntax permits nonsense declarations like the following example, but the compiler later determines that there are too many properties clauses in the declaration.
	Improper use of the **nv_properties** clause:
	`network output SNVT_temp_f temperature`
	`  nv_properties { heartbeatTime }`
	`  nv_properties { defaultOutput };`
	The properties clause is a comma-separated list, and corrected use of the declaration above is shown below:
	`network output SNVT_temp_f temperature`
	`  nv_properties { heartbeatTime,`
	`  defaultOutput };`

419	***A 'cp' network variable cannot be an 'output' NV [NCC#419]*** Configuration property network variables can only be declared as input network variables. See the chapter describing the use of configuration properties in the *Neuron C Programmer's Guide*.
420	***Network variable's property is a duplicate [NCC#420]*** The *LONMARK Application Layer Interoperability Guidelines* specify that no more than one property of any particular SCPT or UCPT type may be used for a network variable. Consult that document for more information.
421	***Overuse of 'cp' network variable [NCC#421]*** A configuration property implemented using a network variable may not appear in more than one property list, unless the property list also uses the static or global keyword. The same network variable cannot be used as a property for the device, network variables, and functional blocks simultaneously. See the chapter describing the use of configuration properties in the *Neuron C Programmer's Guide*.
422	***Invalid property reference [NCC#422]*** The item appearing in the property list is not a configuration property.
423	***The scope value of the LonMark Resource File reference is out of range [NCC#423]*** The valid range of the scope value is 0 .. 6. The compiler has encountered a resource file containing a scope value that is out of that range. Use the NodeBuilder Resource Editor to examine the resource file and correct the scope value if possible.

NCC#	Description
424	***Function type not correct for a fblock director function [NCC#424]*** The **director** function for a functional block must be declared using a specific function prototype. The function returns **void**, and it has two parameters. The prototype must be in the form shown below: `void f (unsigned index, int cmd);` For more information, see the *Neuron C Programmer's Guide* or the *Neuron C Reference Guide*.
425	***Director symbol is not defined or is not a function [NCC#425]*** The initial declaration of the director function for a functional block must appear prior to the declaration of the functional block. Since the director function may very well make reference to the functional block declaration, the initial declaration will probably need to be a function prototype declaration, or a forward reference. For more information, see the *Neuron C Programmer's Guide.*
426	***Too many fblocks declared [NCC#426]*** The Neuron Chip can support a maximum of 254 functional blocks. Note that each element of an array of fblocks counts as one functional block.
427	***The context for the ':::' operator is not a valid context [NCC#427]*** The **::** operator can be used to access the members and the properties of a functional block, as well as the properties of a network variable, or properties of the device. This message indicates that the context, or the portion of the property expression that precedes the **::** operator, is neither a functional block nor a network variable. A functional block array or a network variable array requires an index expression in the context of the **::** operator. To access device properties, use the **::** operator without a context preceding it. See the examples and description of the **::** operator in the *Neuron C Programmer's Guide* and the *Neuron C Reference Guide* for more information.
428	***The specified fblock does not have a director function [NCC#428]*** An attempt to access the director property of the functional block was made, but no director function was declared for the functional block.
429	***Incorrect number of arguments for the director function [NCC#429*** The function returns **void**, and it has two parameters. The prototype for a director function must be in the form shown below: `void f (unsigned index, int cmd);` For more information, consult the *Neuron C Programmer's Guide* or the *Neuron C Reference Guide*.

NCC#	Description
430	***Use of fblock_director function but no fblocks declared [NCC#430]*** The compiler has detected an attempt to use the **fblock_director( )** but no functional blocks were declared in the program.
431	***FPT name used to declare fblock is not in resource files [NCC#431]*** Part of the declaration of a functional block refers to an FPT record in a LONMARK Resource File, however, the name used in the program was not found in the resource file. Check the spelling of the name (it is case-sensitive) and check that the resource file containing the FPT is installed in the resource file catalog on the computer. The catalog can be checked with the NodeBuilder Resource Editor. Verify that the program ID chosen for the project allows for the FPT resource files to be accessed by the compiler. For example, in case the desired FPT is implemented in an FPT device resource file, scope 3 or higher, that applies to all code by manufacturer with ID 0x12345, this FPT cannot be referenced from a project that uses a different manufacturer IF value in its program ID.
432	***The FPT used in fblock declaration is obsolete [NCC#432]*** The LONMARK Device Resource Files permit FPT, NVT, and CPT definitions to be marked as obsolete. This means that a replacement FPT, NVT, or CPT is available, and the use of the obsolete item is discouraged (though it is permitted). Contact LONMARK International for more information on the obsolete FPT, NVT, or CPT.
433	***NV member of an fblock cannot also be a CP [NCC#433]*** A network variable that uses the **config_prop** (abbreviated **cp**) keyword in its declaration is a configuration property network variable. Such a network variable cannot be used as a member of a functional block, but can be used as a property of a functional block.
434	***The specified member of the fblock is not an NV [NCC#434]*** Members of a functional block can only be network variables.
435	***NV member of an fblock cannot also be declared 'config' [NCC#435]*** A network variable that uses the **config** keyword in its declaration is a Neuron C Version 1 configuration property. Such a network variable cannot be used as a member of a functional block, nor can it be used as a property of a functional block.
436	***The specified NV has already been used as an fblock member [NCC#436]*** A network variable (or element of a network variable array) can be a member of at most one functional block.

NCC#	Description
437	***The member '<name>' already has an NV implementation [NCC#437]*** Each member of a profile can be implemented by a network variable in the device's functional block declaration, but the member can only have one implementation. See the chapter discussing the use of functional blocks in the **Neuron C Programmer's Guide**.
438	***The name '<name>' is not a member of the FPT '<FPT-name>' [NCC#438]*** An **implements** statement appears in the member list of an **fblock** declaration, and the member name (which follows the keyword **implements**) does not exist in the FPT record. The FPT record is read from the FPT resource file. Use the NodeBuilder Resource Editor to examine the list of members for the FPT, and check the spelling of the member name. If you are trying to implement a custom member that is not in the FPT record, you may use the **implementation_specific** keyword to accomplish that. See the *Neuron C Reference Guide* for more details.
439	***The FPT '<FPT-name>' specifies that NV member '<name>' must be implemented [NCC#439]*** Network variable members of the FPT are each marked as either mandatory or optional. All mandatory members must have an implementation, as declared using the **implements** statement in the member list in the **fblock** declaration.
440 **441**	***The FPT specifies that the NV member must be 'input' [NCC#440]*** ***The FPT specifies that the NV member must be 'output' [NCC#441]*** Network variable members of the FPT are each marked as either input or output. The network variables that are used to implement the FPT members must match the specification in the FPT record for that member. Check the declared network variable direction.
442	***User-defined interoperable node SD string will be ignored [NCC#442]*** When a Neuron C program uses any of the Neuron C Version 2 features that support the construction of a LONMARK device, the compiler creates the SD and SI information for the device. When the compiler attempts to insert this information into the SI and SD strings, if there is already text present from the user's program, the compiler inserts the LONMARK information at the front of the string, and uses a semicolon character to separate the automatically-generated string from the user-specified string. If the compiler detects that the existing string information started with the special characters that would identify that information as LONMARK information, the compiler issues this warning. The existing LONMARK information in the string is ignored.

NCC#	Description
443	***Insertion of LonMark device SD info truncates existing string [NCC#443]*** When a Neuron C program uses any of the new Neuron C Version 2 features that support the construction of a LONMARK device, the compiler will create the SD and SI information for the device. When the compiler attempts to insert this information into the SI and SD strings, if there is already text present from the user's program, the compiler will insert the LONMARK information at the front of the string, and will then use a semicolon character to separate the automatically-generated string from the user-specified string. If the compiler detects that the addition of this information will cause the string to exceed the limit of 1023 characters, the compiler will truncate the string and issue this message.
444	***The fblock external name string is invalid or is too long [NCC#444]*** The *LONMARK Application Layer Interoperability Guidelines* state that the external name of a functional block (the name which appears in the LONMARK information in the device's SD string) shall be 16 characters or less. There are certain restrictions to the set of acceptable characters, too. See the guidelines document, referenced above, for more information.
445	***NV array element used as member of fblock requires an index [NCC#445]*** A simple (non-array) functional block declaration requires network variable members that are not arrays. These members can either be simple network variables, or elements of network variable arrays. In the case of an element of a network variable array, an index expression must be part of the declaration in the **implements** statement, to identify the array element to be used.
446	***NV array elements used as members of fblock array require a starting index [NCC#446]*** A functional block array declaration must have its members implemented using network variable arrays. The network variable arrays may be larger than the functional block array, and the indices need not be identical (between the functional block array and the various network variable arrays). The reference to each array network variable in the **implements** statements of the member list requires a starting index as part of the declarations in the **implements** statements, to identify which array element of the network variable corresponds to the 0^{th} array element of the functional block. The compiler then automatically distributes the following elements of the network variable arrays to the following elements of the functional block array.
447	***The fblock reference requires an index[NCC#447]*** Each element of a functional block array must be treated separately. An index is always required to select an element of a functional block array.

NCC#	Description

448	***The fblock's NV member array is not big enough for the FB array[NCC#448]***
	
	A functional block array declaration must have its members implemented using network variable arrays. The network variable arrays may be larger than the functional block array, but may not be smaller. The network variable array elements' indices need not start at 0 in correspondence with the functional block array, but they must be consecutive. Thus, the network variable array must be large enough to accommodate the functional block array. For example, consider the following improper declaration:
	
	`network output SNVT_volt v[4];`
	`fblock SFPTwhatever {`
	`  v[2] implements memberV;`
	`} fb[3];`
	
	The functional block array fb has three members, and the network variable array v has four members. However, the declaration does not use v[0] nor v[1], and it matches v[2] with fb[0], and v[3] with fb[1]. There are not enough elements in the array v, because v[4] would be needed for fb[2], but the array's last member is v[3].

449	***The fblock or NV context requires an index[NCC#449]*** A functional block array or a network variable array used as the context (left-hand side) of the property operator **::** must have an index expression. An entire array cannot be used in this manner, only an individual element.
450	***The fblock array requires NV array(s) as members [NCC#450]*** A functional block array declaration must have its members implemented using network variable arrays. The network variable arrays may be larger than the functional block array, and the indices need not be identical (between the functional block array and the various network variable arrays). The reference to each array network variable in the **implements** statements of the member list requires a starting index as part of the declarations in the **implements** statements, to identify which array element of the network variable corresponds to the 0^{th} array element of the functional block. The compiler then automatically distributes the following elements of the network variable arrays to the following elements of the functional block array.
451	***A 'device_specific' CP is required to be 'const' as well [NCC#451]*** The *LONMARK Application Layer Interoperability Guidelines* document requires a configuration property that is declared with the **device_specific** flag to also be declared **const**. CPs with these flags on are always read from the device, they are considered read-only by LNS; thus the required **const**.

NCC#	Description
453	***One or more code constructs have no effect and were ignored [NCC#453]*** When compiling a program that is used as a model file for ShortStack, FTXL, or *i*.LON SmartServer host development, the compiler could encounter executable code or other Neuron C constructs that have no effect for model file compilation. The construct is ignored by the compiler.
454	***The construct is not acceptable in the current mode of operation code [NCC#454]*** Certain Neuron C features are not permitted in a program that is used as a model file for host development with ShortStack, FTXL, or the *i*.LON SmartServer. For example, the compiler directive **#pragma num_alias_table_entries** is not permitted. Consult your platform's documentation for more information.
455	***Cannot open NCT file [NCC#455]*** The output *.NCT file (used during model file compilation) cannot be opened. Perhaps a file already exists by that name, but it is open by another Windows program, or it is marked read-only. Or, perhaps the output directory does not exist.
456	***The directive '#pragma num_alias_table_entries' is required [NCC#456]*** A Neuron C program must specify to the Neuron C Version 2 compiler how much room is to be reserved for the alias table. The compiler does not attempt to compute a default, so the programmer *must* specify a value for this pragma. Previous versions of the Neuron C Compiler defaulted this value to zero, but that is really not an appropriate default value. The value 0 (zero), however, can still be used to effectively disable the use of the alias feature; please see the *Neuron C Reference Guide* for more about compiler directives.
457	***The type used in the declaration is unnamed or unsupported [NCC#457]*** Some types are not supported when compiling a program that is used as a model file for host development with ShortStack, FTXL, or the *i*.LON SmartServer. Consult the product documentation for more information.
458	***The fblock's property is a duplicate [NCC#458]*** The *LONMARK Application Layer Interoperability Guidelines* specify that no more than one property of any particular SCPT or UCPT type may be used for a functional block. Consult that document for more information.

NCC#	Description
459	***Node object must be first fblock for LNS versions before 3.2 [NCC#459]*** For LNS versions prior to version 3.2, if the device has a device object, that device object must be the first functional block declared in the device (and therefore be the functional block with global index zero). You should implement the device object as the first functional block in the device.
460	***Cannot use an inheriting type to declare this object [NCC#460]*** Some CPTs require inheritance of type information from a network variable. Type inheritance is a feature that only applies to configuration properties. Use of such a CPT for any other purpose (for example, declaring a local or static variable in the program, or declaring a function parameter or pointer) results in a declaration with incomplete type information(since only a configuration property can inherit a type from a network variable). A declaration with an incomplete type is not valid. The CPT used in the declaration can only be used as part of a configuration property declaration.
461	***Cannot use an inheriting type CP as a device property [NCC#461]*** A configuration property declared using a CPT type that inherits from a network variable could not be a device property, because in that situation there is no network variable from which to inherit.
462	***Global property cannot inherit conflicting types [NCC#462]*** When a configuration property is declared using the **global** keyword, it is shared among multiple network variables (or functional blocks). Some CPTs are incomplete type definitions, and the configuration properties that use these CPTs in their declarations inherit their types from the network variables they apply to. If a **global** configuration property is used as a property of two or more network variables of different types, there is a resulting conflict in the type inheritance. This situation is not permitted.
463 **464**	***The 'const' attribute has been removed by cast operations [NCC#463]*** ***Pointer to constant data has been cast into pointer to non-const data [NCC#464]*** These warning messages inform the programmer of a potential programming error, because an attempt to write to read-only memory may occur. Writes to read-only memory do not cause problems, other than that the expected write does not occur.

NCC#	Description
465 **466**	***Interoperable user-defined SD string is not acceptable in model file compilation [NCC#465]*** ***Interoperable user-defined node SD string is not acceptable in model file compilation [NCC#466]*** The specified SD string is not acceptable in a program that is used as a model file for ShortStack, FTXL, or *i*.LON SmartServer host development. Consult the product documentation for more information.
467 **468**	***Invalid fblock member number specification [NCC#467]*** ***The implementation-specific member's number conflicts with another member [NCC#468]*** A Neuron C program's implementation of an FPT (a profile) can add one or more members not present in the FPT. These members are called implementation-specific. Such members must specify a member name and a member number since there is no FPT record to provide this information for the compiler. The member name and member number supplied in the **implementation_specific** statement must be unique, and must not conflict with any FPT members, nor any other implementation-specific members. Since a user FPT can inherit from a standard FPT, the implementation-specific members must be unique within both the user FPT and the standard FPT in this situation. See also NCC#605.
469	***The number of FPT members exceeds the compiler's capacity [NCC#469]*** The compiler accepts a maximum of 4,096 members per FPT.
470	***The program ID for this program requires the changeable interface bit [NCC#470]*** A program that has one or more network variables declared as changeable-type must also set the changeable interface bit in the program ID, to tell a network management tool that the program interface is changeable. Use the SPID Calculator in IzoT NodeBuilder tool to set the changeable interface bit of the program ID. See the *LONMARK Application layer Interoperability Guidelines* for more information on this topic.
471	***The reference '<name>' is not a member of the fblock's FPT [NCC#471]*** The context property expression uses a member name that does not exist in the **fblock** declaration. A context property expression is of the form shown below: *fblock-name-with-index* **::** *member-name* Check the **fblock** declaration and use a valid member name from the functional block's member list. You cannot use a member from the FPT that is not implemented in the **fblock**.

NCC#	Description
472	***An inheritable-type CP must be initialized in the properties clause [NCC#472]*** A configuration property declared with a SCPT that provides type inheritance cannot be initialized in the CP family or network variable declaration, because the type is not known and the initializer cannot be processed. In fact, a CP family declared with a CPT that provides type inheritance could have different family members with different types. These types of properties must be initialized in the properties clause. Properties declared with fixed, that is, non-inheriting types can be initialized in either or *both* places, with the initializer in the properties clause superseding the one in the declaration when both are present.
473	***Global property cannot have conflicting initializers [NCC#473]*** When a configuration property is declared using the **global** keyword, it is shared among multiple network variables or functional blocks. A global property that appears in two or more property lists could be initialized differently in those property lists. This situation is not permitted.
474	***The implementation-specific member's name conflicts with another member [NCC#474]*** See the error description for [NCC#467], above.
475	***Debug option set by pragma instead of by command option [NCC#475]*** This warning is provided because the IzoT NodeBuilder (3 or later) Development Tool has user-interface controls for the debug kernel options, but the program is overriding them. Without this warning, a user of IzoT NodeBuilder (3 or later) might think they turned off use of the debug kernel when, in fact, the program is still turning these options on.
476	***Codegen option set by pragma instead of by command option [NCC#476]*** This warning is provided because the NodeBuilder 3 has user-interface controls for certain code generation options, such as the disabling of the compiler's optimizer, but the program is overriding them. Without this warning, a user of the NodeBuilder 3 might think they turned off certain compiler features when, in fact, the program is still turning these options on.
477	***Declaration of 'cp' or 'cp_family' requires use of a CPT [NCC#477]*** A declaration of a configuration parameter *must* use a SCPT or UCPT type in its declaration. This is required by the *LONMARK Application Layer Interoperability Guidelines*.

NCC#	Description
478	***Declaration of 'cp' network variable should use a CPT referencing an NVT [NCC#478]*** When a program is using a standard program ID, all configuration property network variables should be declared with a SCPT or UCPT type that is a reference of a network variable type (SNVT or UNVT). Otherwise, the network variable will appear to a network management tool to be of "unknown type" rather than "interoperable type".
479	***Network variable type is not a SNVT or UNVT [NCC#479]*** When a program is using a standard program ID, all network variables should be declared with a SNVT or UNVT type. Otherwise, the network variable will appear to a network management tool to be of "unknown type" rather than "interoperable type".
480	***External resource string not found in any available string resource file [NCC#480]*** The **external_resource_name** feature in the declaration of a functional block permits the lookup of a string (in US English) in a LONMARK Device Resource File, and the compiler will automatically convert this string into its <scope>:<index> reference. This message indicates that no device resource file applying to this device contained the string.
481	***String constant is too long [NCC#481]*** No Neuron C string constant may exceed 65,000 characters.
482	***The external name of fblock '<fb-name-1>' is a duplicate of '<fb-name-2>' [NCC#482]*** Functional blocks must have unique external names. (All elements of a functional block array have the same name, but the network management tool makes these names unique by using the array index as part of the name.)
483	***The fblock's FPT attempts to inherit from a standard FPT, but no standard FPT was found with a matching key [NCC#483]*** A user-level FPT may indicate (in the LONMARK Device Resource File that contains it) that it inherits members and properties from a standard FPT. The inheritance is by *key*, a 16-bit value associated with each FPT. Standard FPTs have key values from 0-19999, and user FPTs that inherit from standard FPTs must have matching keys. User FPTs that do not inherit should have keys 20000 and above. This message either indicates a problem with the key used for FPT inheritance, or it indicates that the standard FPT is missing. Perhaps the standard file is out of date. The latest standard LONMARK Device Resource Files can be downloaded from the *www.lonmark.org* website.

NCC#	Description
484	***The FPT used in the fblock inherits from an obsolete FPT [NCC#484]*** The LONMARK Device Resource Files permit FPT, NVT, and CPT definitions to be marked as obsolete. This means that a replacement FPT, NVT, or CPT is available, and the use of the obsolete item is discouraged (though it is permitted). In this case, the user FPT is inheriting from a standard FPT that has been marked as obsolete by LONMARK International. Contact LONMARK International for more information about the obsolete FPT, NVT, or CPT.
485	***Cannot inherit type for property from principal NV because principal NV of FPT is unimplemented [NCC#485]*** A configuration property that uses type inheritance from a network variable can also inherit from a functional block. In this latter case, the FPT must designate one of its network variables as the *principal* network variable. The principal network variable designation is solely for the purposes of type inheritance. In the case of FPT inheritance, if the user FPT overrides the standard FPT's principal network variable, but does not designate one of its own, this leaves the principal NV as unimplemented. Then, a configuration property for that functional block, using type inheritance, has nothing to inherit from.
486	***The fblock uses an FPT that has advanced features and requires LNS versions 3.2 or later [NCC#486]*** FPT inheritance and FPT records whose member numbers do not match the member indices are not fully supported in LNS prior to LNS version 3.2. This might result in parts of the devices not being fully or correctly recognized when deploying this device in a network managed with an earlier version of LNS.
488	***NV '<nv-name>' is not an fblock member, and is not a 'cp', so the 'changeable_type' keyword is ignored [NCC#488]*** The only support for changeable-type network variables is through the LONMARK information for the variables. The only network variables with LONMARK information are members of functional blocks and configuration property network variables. Put the network variable in a functional block's member list, or declare it as a configuration property network variable of the device.
489	***NV '<nv-name>' requires a SCPTnvType property, since it is changeable-type [NCC#489]*** The changeable-type network variable mechanism requires that such network variables each have a SCPTnvType property. The SCPTnvMaxLength property is only needed if the network variable also supports changeable-types of various lengths.

NCC#	Description
490	***NV '<nv_name>' is an array of changeable-type NVs, so 'expand_array_info' is recommended [NCC#490]*** A network variable array that is declared as changeable-type should use the **bind_info(expand_array_info)** feature so each element of the variable can have a different type. Otherwise, certain network variable information is shared amongst the members of the array, and in that case, all elements of the array must change type together.
491	***The 'nv_len' property must be used with a 'changeable_type' NV [NCC#491]*** Only network variables that are declared as changeable-type my use the **nv_len** property.
492	***The 'nv_len' property requires a prior '#include <modnvlen.h>' [NCC#492]*** Use of the **nv_len** property requires the include file shown.
493	***The FPT specifies that the NV member must be 'polled' [NCC#493]***
494	***The FPT specifies that the NV member use the 'ackd' service type [NCC#494]***
495	***The FPT specifies that the NV member use the 'unackd_rpt' service type [NCC#495]***
496	***The FPT specifies that the NV member use the 'unackd' service type [NCC#496]***
497	***The FPT specifies that the NV member use request/response service type [NCC#497]***
498	***The FPT specification of the NV member's type does not match the NV [NCC#498]*** The FPT record (used in the functional block declaration) may specify several restrictions on the network variable that implements the member. The messages above result from compiler validations that test whether the network variable used in the **fblock** member list implements the member as required by the FPT.
499	***NV array element used as a property requires an index [NCC#499]*** A functional block declaration may have its properties implemented using configuration parameters or configuration network variables. A simple (non-array) functional block requires any configuration property network variables to be simple NVs, or NV array elements. These properties can either be simple network variables, or elements of network variable arrays. In the case of an element of a network variable array, an index expression must be part of the declaration in the **fb_properties** list, to identify the array element to be used.

NCC#	Description
500	***NV array elements used as properties of an array require a starting index [NCC#500]*** A functional block array declaration may have its properties implemented using configuration parameters or configuration property network variable arrays. The network variable arrays may be larger than the functional block array, and the indices need not be identical (between the functional block array and the various network variable arrays). The reference to each array network variable in the **fb_properties** list requires a starting index as part of the declarations in the **implements** statements, to identify which array element of the network variable corresponds to the 0th array element of the functional block. The compiler then automatically distributes the following elements of the network variable arrays to the following elements of the functional block array.
501	***Non-shared NV used as property of array must itself be an array [NCC#501]*** A functional block array declaration may have its properties implemented using configuration parameters or configuration property network variable arrays. The non-shared properties using network variables must be implemented using network variable arrays. The network variable arrays may be larger than the functional block array, and the indices need not be identical (between the functional block array and the various network variable arrays). The reference to each array network variable in the fb_properties list requires a starting index as part of the declarations in the **implements** statements, to identify which array element of the network variable corresponds to the 0th array element of the functional block. The compiler then automatically distributes the following elements of the network variable arrays to the following elements of the functional block array.

NCC#	Description

502	***NV array used as property is too small [NCC#502]***
	A functional block array declaration may have its properties implemented using configuration parameters or configuration property network variable arrays. The network variable arrays may be larger than the functional block array, but may not be smaller. The network variable array elements' indices need not start at 0 in correspondence with the functional block array, but they must be consecutive. Thus, the network variable array must be large enough to accommodate the functional block array. For example, consider the following erroneous declaration:
	`network output SNVT_volt v[4];`
	`network input cp SCPTdefOutput defOut[6];`
	`fblock SFPTwhatever {`
	`  v[1] implements memberV;`
	`} fb[3]`
	`  fb_properties {`
	`  defOut[4]`
	`  };`
	The functional block array fb has three members, and the network variable array v has four members. The compiler matches v[1] with fb[0], v[2] with fb[1], and v[3] with fb[2], and this is fine. However, the property array starts with defOut[4] matched with fb[0], and defOut[5] is therefore matched with fb[1] (The array elements defOut[0], defOut[1], defOut[2], and defOut[3] are not used in this declaration.) There are not enough elements in the array defOut, because defOut[6] would be needed for fb[2], but the array's last member is defOut[5].

503	***Global property cannot have conflicting range modification strings [NCC#503]*** A global property may be shared between two or more network variables, or between two or more functional blocks. The shared property may have a range-modification specifier assigned in each property list where it appears, but these range-modification specifiers are not permitted to conflict.
504	***The file for scope '<value>' of the external name reference is not available [NCC#504]*** The **external_resource_name** feature of the **fblock** declaration specifies a *<scope>:<index>* reference to a string, but when the compiler attempted to check the validity of the reference by checking the existence of the string, the applicable resource file for scope *<value>* was not available.
505	***The string at index '<value>' is not present in the file for scope '<value>' [NCC#505]*** The external_resource_name feature of the fblock declaration specifies a *<scope>:<index>* reference to a string, but when the compiler attempted to check the validity of the reference by checking the existence of the string, the string was not present in the specified resource file.

NCC#	Description
506	***The 'cp' network variable is of inheriting type, but no type has been inherited at this point [NCC#506]*** A reference to a configuration property network variable has been encountered but the configuration property has not yet inherited a type. The reference in the executable code cannot be compiled, because the variable does not yet have a type.
507	***The FPT requires that this property be declared as 'const' [NCC#507]***
508	***The FPT requires that this property be declared as 'device_specific' [NCC#508]***
509	***The FPT requires that this property be declared as 'manufacturing_only' [NCC#509]***
510	***The FPT requires that this property be declared as 'object_disabled' [NCC#510]***
511	***The FPT requires that this property be declared as 'offline' [NCC#511]***
512	***The FPT requires that this property be declared as 'reset_required' [NCC#512]*** The FPT record (used in the functional block declaration) may specify several restrictions on the configuration properties that appear in its property list. The messages above result from compiler validations that test whether the configuration property used in the fb_properties list of the **fblock** declaration properly implements the FPT property.
513	***The FPT '<FPT-name>' specifies that the CP member '<name>' is mandatory, but no corresponding property was found [NCC#513]*** Configuration properties of the FPT are each marked as either mandatory or optional. All mandatory properties must have an implementation, by appearing in the **fb_properties** list of the functional block, or in the **nv_properties** list of the member NV to which the property applies.
514	***The FPT's inherited mandatory CP member '<name>' could not be implemented because the inherited NV member '<name>' was overridden [NCC#514]*** A user FPT has inherited a standard FPT with a mandatory configuration property applying to a network variable, but the standard FPT's network variable member is overridden in the user FPT. The user FPT needs to also override the problematic configuration property, and either make it optional, or make it apply to a mandatory member of the user FPT or to another NV member of the standard FPT that is not overridden.
515	***The FPT's mandatory CP member '<name>' applies to an unimplemented NV member '<name>' [NCC#515]*** You should review the definition of this CP member within the functional profile definition.

NCC#	Description
516	***The FPT specifies that mandatory CP member '<name>' applies to NV member '<name>', but no corresponding property was found [NCC#516]*** The compiler did not find the mandatory property appearing in the **nv_properties** list of the member network variable.
517	***Global property cannot have conflicting initializers from FPT definitions [NCC#517]***
518	***The 'cp' network variable is of inheriting type, but no type was inherited [NCC#518]*** At the end of the program compilation, the compiler verifies that all configuration property network variables have inherited a type (by being used as the property of a network variable, or as the property of a functional block with a principal NV). Any configuration property network variables that have not inherited a type cannot be allocated space, and the compilation cannot complete successfully.
519	***The 'changeable_type' network variable array element '<nv-name>[<index>]' was not used; therefore the type for that element cannot be changed [NCC#519]*** If an element of a network variable array declared as changeable-type is not used, it will have no LONMARK information, thus, a network management tool cannot change its type.
520	***An address constant initializer cannot be used in model file compilation mode, and was ignored [NCC#520]*** A program that is used as a model file for host development with ShortStack, FTXL, or the *i*.LON SmartServer cannot use address constants as initializers.
521	***Network variable property cannot inherit conflicting types [NCC#521]*** A configuration property network variable may be shared among multiple network variables (or among multiple functional blocks). Some SCPT types are not actual type definitions, and the configuration properties that use these SCPTs in their declarations inherit their types from the network variables they apply to. If a configuration property network variable were to be used as a property of two or more network variables of different types, there would be a conflict in the type inheritance. This situation is not permitted.
522	***SD string supplied exceeds the 1023 character limit after the 'expand_array_info' fixup. [NCC#522]*** The **bind_info(expand_array_info)** causes certain fixups to be applied to the SD strings of network variable array elements to make each string unique. These alterations (fixups) could increase the length of the string beyond the 1023 character limit for such strings.

NCC#	Description
525	***Cannot have the '#pragma skip_ram_test_except_on_power_up' because it conflicts with '#pragma ram_test_off' [NCC#525]*** You cannot choose both options, since they would be logically contradictory with each other.
526	***Cannot have the '#pragma ram_test_off' because it conflicts with '#pragma skip_ram_test_except_on_power_up' [NCC#526]*** You cannot choose both options, since they would be logically contradictory with each other.
528	***The FPT does not permit this property to be an array [NCC#528]*** Neuron C Version 2.1 introduced configuration properties that are arrays. In other words, the entire array is treated as a single property. The FPT resources (SFPT*, UFPT*) designate, for each property, whether that property may not, may, or must be an array. This error message indicates that the program attempted to use an array property for an FPT CP member, but the FPT does not permit that particular CP member to be an array.
529	***The FPT requires that this property be an array of <count>elements [NCC#529]*** Neuron C Version 2.1 introduced configuration properties that are arrays. In other words, the entire array is treated as a single property. The FPT resources (SFPT*, UFPT*) designate, for each property, whether that property may not, may, or must be an array. In the case of properties that must be an array, the FPT can specify a fixed array bound, or a variable array bound within a range. This error message indicates that the program attempted to instantiate a property for an FPT CP member, but the FPT requirement that the array bound be of a certain fixed size was not met.
530	***The FPT requires that this property array have at least <minimum> elements [NCC#530]*** Neuron C Version 2.1 introduced configuration properties that are arrays. In other words, the entire array is treated as a single property. The FPT resources (SFPT*, UFPT*) designate, for each property, whether that property may not, may, or must be an array. In the case of properties that must be an array, the FPT can specify a fixed array bound, or a variable array bound within a range. This error message indicates that the program attempted to instantiate a property for an FPT CP member, but the property does not meet the minimum array bound requirement of the FPT (the property is too small).

NCC#	Description
531	***The FPT requires that this property array have no more than <maximum> elements [NCC#531]*** Neuron C Version 2.1 introduced configuration properties that are arrays. In other words, the entire array is treated as a single property. The FPT resources (SFPT*, UFPT*) designate, for each property, whether that property may not, may, or must be an array. In the case of properties that must be an array, the FPT can specify a fixed array bound, or a variable array bound within a range. This error message indicates that the program attempted to instantiate a property for an FPT CP member, but the property does not meet the maximum array bound requirement of the FPT (the property is too large).
532	***The expand_array_info option is not compatible with use of the NV as CP array [NCC#532]*** Use of a network variable array as an array configuration property (the entire array is a single property) cannot be used with a network variable that is declared with the **expand_array_info** option.
533	***The spi I/O object cannot use the 'clockedge(+-)' option [NCC#533]*** For a **spi** I/O object, you must use either **clockedge(+)** or **clockedge(-)**.
534	***The select pin must be IO_7 for the 'spi' device type [NCC#534]*** If a **spi** I/O object declaration uses the optional **select** pin, it must use **IO_7**.
535	***The baud rate specified is not a supported rate for the 'sci' device type [NCC#535]*** The **sci** I/O object supports only a certain predefined set of baud rates. See the description of the **sci** I/O object in the *I/O Model Reference*.
536	***The FPT requires that this property be an array of <minimum>..<maximum> elements [NCC#536]*** Neuron C Version 2.1 introduced configuration properties that are arrays. In other words, the entire array is treated as a single property. The FPT resources (SFPT*, UFPT*) designate, for each property, whether that property may not, may, or must be an array. In the case of properties that must be an array, the FPT can specify a fixed array bound, or a variable array bound within a range. This error message indicates that the program attempted to instantiate a property for an FPT CP member, but the property's array bound does not fall within the required range of the array bound as designated by the FPT.

NCC#	Description
537	***If I/O clock is specified, the pragma must precede the SCI device declaration [NCC#537]*** The **sci** I/O object declaration can (and in most cases does) specify an initial baud rate. This baud rate is used to construct a register setting that is dependent on the device's input clock. The **#pragma specify_io_clock** is used to communicate the input clock value to the compiler, and it must appear in the program before the declaration of the I/O object. See the *I/O Model Reference* for more information.
538	***The #pragma codegen no_cp_template_compression is incompatible with #pragma codegen cp_family_space_optimization selected earlier [NCC#538]*** You cannot choose both options, since they would be logically contradictory with each other.
539	***The #pragma codegen cp_family_space_optimization is incompatible with #pragma codegen no_cp_template_compression selected earlier [NCC#539]*** You cannot choose both options, since they would be logically contradictory with each other.
540	***I/O object type restricted to pins IO_0 or IO_8 [NCC#540]*** The particular I/O object being declared must either be declared on **IO_0** or **IO_8**.
541	***The output pin is restricted to pins IO_0 through IO_7 [NCC#541]*** The particular I/O object being declared must be declared on one of the pins **IO_0** through **IO_7**.
542	***Timing values for touch I/O object must be in range 1..256 [NCC#542]*** As the message says, if you supply the optional timing values in the declaration of the touch I/O object, these values must be in the range of 1 to 256. However, note that a zero value is interpreted as 256, so to use 256 as a timing value you must actually supply a zero.
543	***The SCPTnvType and SCPTmaxNVLength can only apply to changeable_type NVs [NCC#543]*** Properties of type **SCPTnvType** or **SCPTmaxNVLength** are only permitted as a property of one or more network variables declared as **changeable_type**. See *How Devices Communicate Using Network Variables* in the *Neuron C Programmers Guide* for more information.

NCC#	Description
544	***The #pragma system_image_extensions nv_length_override must precede all uses of the 'nv_len' property [NCC#544]*** Use of the directive **#pragma system_image_extensions nv_length_override** selects use of the user-written extension function **get_nv_length_override()** when determining the length of a network variable. Therefore this pragma must be used to select this method prior to any attempts to get the length of the network variable. The pragma must appear in the program before any use of the **nv_len** property. See *How Devices Communicate Using Network Variables* in the *Neuron C Programmers Guide* for more information.
545	***Reading the nv_len property within the get_nv_length_override function is prohibited [NCC#545]*** The **get_nv_length_override()** function is provided by the application programmer when using the system extension option **nv_length_override**. This function is also called by the compiler to evaluate the **nv_len** property for a network variable. Therefore, the function may not contain any such references to the **nv_len** property, else there would be an endless loop. See *How Devices Communicate Using Network Variables* in the *Neuron C Programmers Guide* for more information.
546	***The #pragma system_image_extensions nv_length_override must precede the get_nv_length_override function's definition [NCC#546]*** Use of the directive **#pragma system_image_extensions nv_length_override** selects use of the user-written function **get_nv_length_override()** when determining the length of a network variable. Therefore this pragma must be used to select this method prior to the function definition, so the compiler can properly recognize the special nature of this function definition. See *How Devices Communicate Using Network Variables* in the *Neuron C Programmers Guide* for more information.
547	***AUTO initializers not implemented [NCC#547]*** The Neuron C Compiler syntax does not permit use of initializers for automatic variables in their declaration. (Automatic variables are variables declared within a function scope, or a nested scope, or inside the task body associated with a **when** clause.) Initialize the variables with separate statements following the declaration of all automatic variables in a function.

NCC#	Description
548	***The property '<property-name>' cannot be shared by changeable_type NVs of differing initial types (<NV-name-1> and <NV-name-2>) [NCC#548]*** A configuration property of **SCPTnvType** that is shared by more than one **changeable_type** network variable must be shared only by network variables with the same initial type. The message lists two network variables that share the property, but have differing initial types. See *How Devices Communicate Using Network Variables* in the *Neuron C Programmers Guide* for more information.
549	***The property '<property-name>' cannot be shared by NVs with different SCPTmaxNVLength properties (<NV-name-1> and <NV-name-2>) [NCC#549]*** A configuration property of **SCPTnvType** type that is shared by more than one **changeable_type** network variable must be shared only by network variables that all have the same property of **SCPTmaxNVLength** type, if any of those network variables use a property of **SCPTmaxNVLength** type. The message lists two network variables that share the property, but have differing SCPTmaxNVLength.
550	***The property '<property-name>' cannot be shared by NVs of differing types (<NV-name-1> and <NV-name-2>) [NCC#550]*** A configuration property that inherits its type from the network variable that it applies to may not be shared by two or more network variables of different type. The message lists two network variables that share the property, but have differing types. See *Using Configuration Properties to Configure Device Behavior* in the *Neuron C Programmer's Guide* for more information.
551	***The property '<property-name>' cannot be shared by NVs with different SCPTnvType properties (<NV-name-1> and <NV-name-2>)" [NCC#551]*** A configuration property that inherits its type from the network variable that it applies to may not be shared by two or more network variables of different type. The message lists two network variables that share the property, but have differing SCPTnvType properties. See *How Devices Communicate Using Network Variables* and *Using Configuration Properties to Configure Device Behavior* in the *Neuron C Programmer's Guide* for more information.
552	***Symbol in preprocessor directive is not defined [NCC#552]*** The message indicates that the symbol specified as the parameter for the compiler directive **#pragma ignore_notused** is not defined. The directive is ignored after the warning message is displayed.

NCC#	Description
553	***Symbol '<symbol>' in preprocessor directive is not permitted in this directive [NCC#553]*** The message indicates that the symbol specified as the parameter for the compiler directive **#pragma ignore_notused** cannot be used with this feature. This restriction applies to symbols that are the names of macros, **typedef** names, and names of types from resource files (SNVT*, SCPT*, UNVT*, UCPT*, SFPT*, and UFPT*).
554	***The NVT (Network variable Type) is obsolete [NCC#554]*** A resource file can optionally designate any of the network variable types that it contains as being obsolete. This designation indicates that the network variable type should no longer be used in new development. This designation does not prevent its use, but it does display this warning message.
555	***The CPT (Configuration Property Type) is obsolete [NCC#555]*** A resource file can optionally designate any of the configuration property types that it contains as being obsolete. This designation indicates that the configuration property type should no longer be used in new development. This designation does not prevent its use, but it does display this warning message.
556	***Array size exceeds 65500 [NCC#556]*** A configuration property array may not exceed 65500 bytes in total size, determined as the product of the element size and the array bound. This size is of no practical limit, since a Neuron does not have this much contiguous memory space available for configuration properties.
557	***The CP array construct is not acceptable in model files [NCC#557]*** A program that is used as a model file for host development with ShortStack, FTXL, or the *i*.LON SmartServer cannot use configuration property arrays.
559	***Duplicate/repeated cp_info keyword is ignored [NCC#559]*** A configuration property declaration may optionally contain a parenthesized list of keywords representing option flags. If one or more of these keywords is repeated or duplicated, this warning message is displayed and the repetition or duplication has no other effect.
562	***The device SD string exceeds the limit of 1023 characters. Consider using shorter external names, or fblock arrays [NCC#562]***
563	***Too many instructions. You need to reduce the size of your application, or acquire an unlimited version of the Neuron C Compiler [NCC#563]*** you can purchase an unrestricted compiler with the IzoT NodeBuilder FX tool.

NCC#	Description
567 **568** **569**	***Network variables maximum exceeds supported maximum in current context [NCC#567]*** ***Alias maximum exceeds supported maximum in current context [NCC#568]*** ***Fblock maximum exceeds supported maximum in current context [NCC#569]*** These errors indicate an attempt to configure the Neuron C compiler in an unsupported fashion.
571	***Too many libraries [NCC#571]*** You cannot specify more than 20 libraries with the #pragma library directive, but you can explicitly reference more libraries in the NodeBuilder project, or when explicitly calling the Neuron Linker on the console.
572	***Use of this pragma is restricted. Please contact Echelon for assistance [NCC#572]*** You are using a licensed feature, but you do not appear to have a valid license for this feature.
573	***Network variable size is greater than 31 bytes. This NV type is not interoperable with LonMark-compliant devices [NCC#573]*** Under certain conditions, network variable types can exceed the 31 byte size limit. These network variables are not interoperable.
574	***NV declaration should not be based on a CPT. Are you missing a 'cp' modifier?[NCC#574]*** Declare network variables using network variable types, and declare configuration properties using configuration property types. For configuration properties implemented as configuration network variables, use configuration property types that reference a network variable type. Do not declare network variables using configuration property types unless you declare a configuration network variable.
575	***The system timer interrupt is already used with a different interrupt task [NCC#575]*** You can only declare one interrupt task associated with the system timer interrupt.
576	***All available timer/counter interrupts are already used with different interrupt tasks [NCC#576]*** You can only declare up to two interrupt tasks associated with the hardware timer or counter units.

NCC#	Description
577	***All available I/O interrupts are already used with different interrupt tasks [NCC#577]*** You can only declare up to two interrupt tasks associated with I/O.
578	***Parser error (unexpected semantic device type) [NCC#578]***
579	***This I/O device cannot trigger an interrupt task; use a supported timer/counter configuration or declare an I/O interrupt instead [NCC#579]*** Only the hardware timer and counter units can generate interrupts that can be referenced through the I/O model, but you can reference all available I/O pins in declarations of I/O interrupt tasks.
580	***A single I/O interrupt cannot be triggered through both high and low levels; use a single level trigger and define two I/O interrupts if needed [NCC#580]***
581	***Malformed system timer frequency value [NCC#581]*** When specifying the periodic system timer's frequency in the declaration of a system timer interrupt task, you can describe the frequency as a floating-point constant in typical C notation. The value of this constant describes the requested timer frequency in Hertz. Optionally, you can use one of the following postfix modifiers for readability: "Hz," "kHz," "MHz" or "GHz." Note those postfix modifiers are case-sensitive. The following frequency definitions result in the same value: "1e3," "1000," "1kHz," "1e-3MHz," "1000Hz". The NCC#581 error related to an unrecognized postfix modifier.
582	***System timer frequency is out of range [NCC#582]*** The system timer's frequency range is 2441.406 .. 625000 Hz. A 20% error on either end of the scale is supported (1953.1248 .. 687500Hz). See also NCC#583.
583	***The resulting system timer interrupt frequency will be <f>, causing an error of <p>% from the specified frequency [NCC#583]*** The periodic system timer cannot be configured for every possible value with the supported range of 2441.406 .. 625000 Hz. Instead of requiring that you type one of 256 possible exact frequency values, the Neuron C compiler accepts any value (within range), and corrects it to the nearest true frequency value. Variations between the desired and true frequency under 1% are ignored. The NCC#583 warning occurs if the true frequency is more than 1% different from the value specified in source code.

NCC#	Description
584	***The hardware timer/counter unit is already used with a different interrupt task [NCC#584]*** You cannot reuse the same hardware timer or counter with a second interrupt task.

585	***Nested 'lock' construct [NCC#585]***
	The Series 5000 and Series 6000 Chips support exactly one hardware semaphore, and *__lock* constructs may not be nested, therefore.
	Note this diagnostic only applies to explicit nesting. The following hypothetical construct triggers this error:
	`void f(void) {`
	`  __lock {`
	`  __lock {`
	`  …`
	`  }`
	`  }`
	`}`
	However, it is possible for *__lock* constructs to nest at runtime by executing two nested functions which both implement a lock. This error condition can only be detected at runtime; the system firmware resolves the resulting deadlock with a watchdog timer reset, and logs a system error code.

586	***You cannot combine a falling edge with a dual-edge triggered I/O interrupt [NCC#586]***
587	***Debugging of optimized code is not supported and not recommended. You should disable debugger support, or disable the optimizer [NCC#587]***
588	***Intermixing 'pragma optimization' with deprecated codegen options for optimization control is not recommended [NCC#588]*** You should use **#pragma optimization**.
589	***This codegen option is deprecated and may be discontinued in a future release. Use 'pragma optimization' instead [NCC#589]***
590	***Your optimization preferences in source code conflict with those expressed on the command line or the IDE, and will be ignored [NCC#590]***
591	***You cannot change a buffer size or count value once it has been designated 'final.' Consider using the 'minimum' modifier or eliminate the superfluous buffer control directive [NCC#591]***
592	***The 'minimum' buffer size or count value exceeds a previously requested 'final' value for the same aspect This is at the location of the 'minimum' request [NCC#592]***

NCC#	Description
593	***The 'final' buffer size or count value doesn't meet the earlier 'minimum' specification for the same aspect. This is at the location of the 'final; request [NCC#593]***
594	***The 'level' modifier is no longer supported, use 'pulse' or the stretchedTriac output model instead [NCC#594]***
595	***This I/O object requires the 'frequency' modifier to denote the typical power line frequency [NCC#595]***
596	***The specified frequency is out of range for the stretched triac I/O model [NCC#596]***
597	***The 'twostopbits' option and the parity feature are mutually exclusive [NCC#597]***
598	***The CP is declared with both 'object_disabled' and 'offline'. 'offline' has higher priority; 'object_disabled' becomes ineffective [NCC#598]***
599	***The configuration network variable is declared constant, but may be updated through network variable updates [NCC#599]***
600	***The 'specify_io_clock' directive has no effect unless an SCI I/O object is being declared, and will be ignored [NCC#600]***
601	***I2C model modifiers cannot be repeated [NCC#601]***
602	***Modifiers <a> and <b> are mutually exclusive [NCC#602]***
603	***The I2C I/O object <o> cannot use the version 1 I2C I/O model (as requested with '#pragma codegen use_i2c_version_1'), because it uses features not supported in version 1. The version 2 I2C I/O model is being used instead [NCC#603]***
604	***The guideline version string <s> does not describe a guidelines version in the major.minor format [NCC#604]*** This warning can occur when specifying a guidelines version with the #pragma set_guidelines_version directive. The Neuron C Compiler supports any format for that string, but issues this warning if the format is unexpected. A typical guidelines version string follows a major.minor format (for example, "3.4").

NCC#	Description
605	***The application is built for interoperability guidelines <v>, but implements an implementation-specific member NV or CP. This application will not pass certification [NCC#605]*** If you use implementation-specific network varibales or configuration properties in your device interface, your device will not comply with interoperability guidelines version 3.4 (or later) and therefore cannot be certified by LONMARK International. A better alternative for adding members to a functional profile is to create a user-defined functional profile template (UFPT) that inherits from an existing standard functional profile template (SFPT), and then add new mandatory or optional member network varibales and configuration properties to the UFPT. This method results in a new functional profile that you can easily reuse in new devices. See the *NodeBuilder Resource Editor User's Guide* for more information on creating new functional profiles. Alternatively, you can remove the implementation-specific network varibles and configuration propetrties from the device interface (because they are not fully interoperable).
606	***SCPTnwrkCnfg must be implemented as a configuration network variable [NCC#606]*** This warning is given when a **SCPTnwrkCnfg** configuration property that has been implemented as a CP family is listed in a properties clause. This warning is displayed in applications designed for interoperability application layer guidelines 3.4 or better.
607	***The use of infinite locks in a release build is not recommended [NCC#607]*** This warning is given when you are building a release target that contains one or more infinite *__lock* constructs (a *__lock* construct that uses the **#pragma deadlock_is_infinite** directive). Use finite *__lock* constructs instead. To do this, replace all **#pragma deadlock_is_infinite** directives in *__lock* constructs with the **#pragma deadlock_is_finite** directive. Rebuild and then reload the release target. For more information on the **#pragma deadlock_is_infinite** and **#pragma deadlock_is_finite** directives, see the *Neuron C Reference Guide*.
611	***The –target command is not required in this mode, and will be ignored [NCC#611]*** This warning is given when the compiler is invoked from the console using the --target command, but tasked with a pure C compilation (not a .NC source file). The specification of a compilation target is not required in pure C compilation and has no effect.

NCC#	Description
612	***The –-target command is required in this mode [NCC#612]*** This warning is given when the compiler is invoked for Neuron C compilation from the console without using the --target command. The Neuron C compiler version 6 or better requires the specification of the compilation target for Neuron C (.NC) sources.
613	***The target does not support freely programmable I/O pull-ups, or can't support all which were requested [NCC#613]*** This warning is given when the *pragma enable_io_pullups* directive is not supported with the chosen compilation target, or the target does not support the optional pull-up selection provided with this directive.
616	***Can't execute external preprocessor lonmcpp32 [NCC#616]*** This error results from a failure to launch *lonmcpp32.exe*, the Neuron C preprocessor executable. To remedy this problem, try re-installing the Neuron C development tool.
617	***Directive '#pragma push' can't execute: stack is full [NCC#617]*** The error occurs when a segment stack overflow occurs with *pragma segment* directives. This directive is reserved for internal use.
618	***Directive '#pragma pop' can't execute: stack is empty [NCC#618]*** The error occurs when a segment stack underflow occurs with the *pragma segment* directive. This directive is reserved for internal use.
622	***pragma disable_mult_module_init cannot be used with –-sysinit [NCC#622]*** The offending feature combination is reserved for internal use only.
624	***Pointers to near system functions require resident mode [NCC#624]*** In transient compilation mode, function pointers and indirect function calls through pointers to *near* system functions are not supported.
627, 628	***"The locate directive is malformed: <text>. Use #pragma locate <name> {seg <s>} {org <o>} [NCC#627], [NCC#628]*** Both errors indicate a syntax error in the specification of the *pragma locate* directive. The two errors show the same message but relate to different locations within the directive's parser. See the Neuron C Reference Guide for more details about this directive.
629	***The locate directive cannot affect functions already implemented, or declared 'extern' [NCC#629]*** See the Neuron C Reference Guide for more details about this directive.

NCC#	Description
630	***The locate directive can only locate locally defined items which are not already implemented [NCC#630]*** See the Neuron C Reference Guide for more details about this directive.
631	***The target does not support DHCP (ignoring directive #pragma dhcp enabled) [NCC#631]*** See the Neuron C Reference Guide for more details about this directive.
632	***The target does not support enhanced mode (ignoring directive #pragma enhanced_mode enabled) [NCC#632]*** See the Neuron C Reference Guide for more details about this directive.
633	***Pragma malformed. Use #pragma dhcp (enabled|disabled) [NCC#633]*** See the Neuron C Reference Guide for more details about this directive.
634	***Pragma malformed. Use #pragma enhanced_mode (enabled|disabled) [NCC#634]*** See the Neuron C Reference Guide for more details about this directive.

8

Neuron Exporter Errors (NEX)

This chapter lists and describes the errors that can be reported by the Neuron Exporter.

NEX Errors

Table 10 lists the NEX error codes.

Table 10. NEX Error Codes

NEX#	Description
1	***Numerical value out of range: <parameter>=<value> (<min>..<max>) [NEX#1]*** Numeric value out of range. See error message for details. A command was specified correctly, but the parameter value given was invalid. Verify that to correct your build scripts as appropriate.
2	***Invalid record in <file>, line <lineno> [NEX#2]*** Invalid record in input file, see error message for details.
3	***Invalid record in <file>, line <line#>: too short [NEX#3]*** Invalid record in input file (record too short), see error message for details.
4	***Invalid record in <file>, line <line#>: bad checksum [NEX#4]*** Invalid record in input file (bad checksum), see error message for details.
5	***Unable to locate file '<file>' [NEX#5]*** Unable to locate input file. See error message for details. You can attempt to fix this problem by building the target unconditionally, and by making sure the file does exist prior to invoking the tool.
6	***Unable to allocate memory [NEX#6]*** Out of memory. When running in a DOS virtual machine, make sure to offer sufficient memory. When running in a Win32 environment, this should not occur.
7	***No valid records found in '<file>' [NEX#7]*** No valid records in input file; see error message for details.
8	***Unable to access file '<file>' for writing [NEX#8]*** Unable to access file for writing. The file could be write-protected, or locked by another process.
9	***Attempt to write to unopened file [NEX#9]*** Attempt to write to an unopened file. This is an internal error condition; please contact LonSupport.
10	***Writing to file '<file>' failed [NEX#10]*** Writing to file failed. The file could be locked by some other process, and the file could be write-protected.

NEX#	Description
11	***Attempt to read from an unopened file. [NEX#11]*** This is an internal error condition; please contact LonSupport.
12	***Reading from file '<file>' (read behind end of file) [NEX#12]*** Reading from file failed (read behind end of file). This is an internal error condition; please contact LonSupport.
13	***Unexpected EOF in file '<file>' at line #<line> [NEX#13]*** Unexpected end of file. See error message for details. Attempt fix-up with rebuild.
14	***Line # <line> in input file '<file>' is too long [NEX#14]*** A record in an input file was longer than expected. See error message for details. Attempt fix-up with rebuild.
15	***Line # <line> in input file '<file>' is invalid [NEX#15]*** A record in an input file is invalid. See error message for details. Attempt fix-up with rebuild.
16	***Line # <line> in input file '<file>', byte count is invalid [NEX#16]*** A record in an input file is invalid due to an incorrect byte count. See error message for details. Attempt fix-up with rebuild.
17	***Line # <line> in input file '<file>', checksum is invalid [NEX#17]*** A record in an input file is invalid, bad checksum. See error message for details. Attempt fix-up with rebuild.
18	***System image symbol table '<file>' is invalid: symbol '<symbol>' not defined [NEX#18]*** Missing a required symbol from the system image's symbol table. See error message for details. Does the chosen system image support the desired memory configuration? Change the firmware version and image file to the default and perform an unconditional build.
19	***Specified transceiver type has invalid clock/bit rate combination [NEX#19]*** Specified transceiver type has invalid clock/bit rate combination. Verify that your hardware clock rate and transceiver preferences are correct.
20	***Device's input clock is below minimum allowed for channel [NEX#20]*** Device's input clock is below the minimum allowed for the desired channel. Choose a different transceiver or a faster clock rate.

NEX#	Description
21	***Unable to compute device's Communication Parameters [NEX#21]*** Unable to compute device's communication parameters. Verify device clock speed and transceiver preferences.
22	***Out of memory while creating file '<file>' [NEX#22]*** Out of memory when creating an output file. See error message for failure details.
23	***Cannot find file '<file>' [NEX#23]*** Cannot find a required file. Attempt to fix with rebuild.
24	***Cannot create file '<file>' [NEX#24]*** Cannot create a file. Verify that the media is writable and the destination folder has not been write-protected.
25	***Cannot read file '<file>' [NEX#25]*** Cannot read from a file. Attempt to fix with rebuild.
26	***Cannot write file '<file>' [NEX#26]*** Cannot write to a file. Verify that the media is writable and the destination folder has not been write-protected.
27	***Unable to copy file '<source>' to '<destination>' [NEX#27]*** Unable to copy a file. Verify that the media is writable and the destination folder has not been write-protected.
28	***XIF version is <version>. Can only convert XIF versions 3.0 or later to XFB [NEX#28]*** Cannot process the external interface file due to outdated version. See error message for failure details. Rebuilding unconditionally should fix this by creating an up-to-date version 4 XIF file.
29	***Line <line#> (<line>): Should have <count> fields, but actually has <count> [NEX#29]*** Bad external interface file, field count mismatch. See error message for failure details. Attempt to fix with rebuild.
30	***NV '<nvname>', <count> fields expected, <count> found in NV type info [NEX#30]*** Bad external interface file, NV field count mismatch. See error message for failure details. Attempt to fix with rebuild.

NEX#	Description
31	***Transceiver type <id> is invalid [NEX#31]*** Invalid transceiver type. Verify that to specify the correct transceiver when launching the Exporter.
32	***Unable to access Standard Xcvr Type file: '<file> [NEX#32]*** Unable to access the standard transceiver database file. See error message for failure details. An updated version of the standard transceiver database file might be available from the www.lonmark.org website.
33	***Invalid xcvr type ID specified: '<id>' [NEX#33]*** Invalid transceiver type. There is no such transceiver in the standard transceiver database **stdxcvr.xml**.
34	***Unable to access Neuron Type file: '<file>' [NEX#34]*** Unable to access the Neuron type database. See error message for failure details.
35	***Invalid neuron model ID specified: '<id>' [NEX#35]*** Invalid Neuron model specified; there is no such Neuron model defined in the **neuron.xml** database. See error message for failure details.
36	***Malformed record '<tag>' in dependency file <file> [NEX#36]*** Malformed record in dependency file <file>. See error message for failure details. Attempt to fix with an unconditional rebuild.
37	***Unexpected end of dependency file <file> [NEX#37]*** Unexpected end of dependency file <file>. Attempt to fix with an unconditional rebuild.
38	***Missing separator in clause '<tag>', dependency file <file> [NEX#38]*** Missing separator in dependency file <file>. See error message for failure details. Attempt to fix with an unconditional rebuild.
39	***Bad section name '<section>' in dependency file <file> [NEX#39]*** Bad section name in dependency file. See error message for failure details. Attempt to fix with rebuild.
40	***Error processing dependency file <file> [NEX#40]*** Error processing dependency file <file>. See error message for failure details. Attempt to fix with rebuild.

NEX#	Description
41	***Malformed or missing record '<tag>=' in dependency file <file> [NEX#41]*** Malformed or missing record in dependency file <file>. See error message for failure details. Attempt to fix with rebuild.
42	***Malformed or missing parameter record '<tag>' in dependency file <file> [NEX#42]*** Malformed or missing parameter record in dependency file <file>. See error message for failure details. Attempt to fix with clean and complete rebuild.
43	***Can not compute communication port control byte [NEX#43]*** Cannot compute communication port control byte. Verify that your standard transceiver parameter database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
44	***Unknown reason [NEX#44]*** Failure to compute communication parameters, no reason given.
45	***Internal error [NEX#45]*** Internal error when calculating communication parameters.
46	***Invalid data [NEX#46]*** Invalid data caused error when calculating communication parameters. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
47	***Unable to compute CP configuration for transceiver <xcvr> [NEX#47]*** Unable to compute the CP (Communications Parameters) configuration. See error message for failure details. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
48	***Unrecognized encoded clock rate value <value> [NEX#48]*** Unrecognized clock ID. Currently supported encoded clock ID values range from 0 (625kHz) to 7 (40MHz).

NEX#	Description
49	***Unable to compute end-of packet wait time for single-ended or differential mode transceiver <xcvr> with hardware clock ID <id> [NEX#49]*** Unable to compute the end-of packet wait time for a single-ended or differential mode transceiver. This is most likely caused by a very fast-running Neuron chip, combined with a slow channel type. Reduce the clock speed and see if the problem persists.
50	***The transceiver's general purpose data record seems malformed: <gp data> [NEX#50]*** The transceiver's general-purpose data record seems malformed. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
51	***The device's clock rate (encoded value <id>) and the transceiver's communication rate (encoded value <id>) result in an invalid communication clock divider value. One of the two input rates might be invalid [NEX#51]*** The device's clock rate and the transceiver's communication rate result in an invalid communication clock divider value. One of the two input rates might be invalid. See the error message for failure details. Try increasing or lowering the Neuron clock speed.
52	***The transceiver requires a minimum clockrate which is higher than the device's configured input clock speed [NEX#52]*** The transceiver requires a minimum clock-rate that is higher than the device's configured input clock speed. This combination cannot be used; you must choose a higher clock rate or a different transceiver.
53	***The encoded value for the device's clock input of <id> is not within the supported range of <min> to <max> [NEX#53]*** The encoded value for the device's clock input is not within the supported range. See error message for failure details.
54	***The encoded value for the channel's minimum clock rate of <id> is not within the supported range of <min> to <max> [NEX#54]*** The encoded value for the channel's minimum clock rate of <id> is not within the supported range. See error message for failure details. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.

NEX#	Description
55	***Unable to determine preamble length using transceiver <xcvr> [NEX#55]*** Unable to determine the preamble length. See error message for failure details. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
56	***Unable to determine packet cycle duration using transceiver <xcvr> [NEX#56]*** Unable to determine the packet cycle duration. See error message for failure details. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
57	***Unable to determine beta-2 control value using transceiver <xcvr> [NEX#57]*** Unable to determine the beta-2 control value. See error message for failure details. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
58	***Unable to determine transmit interpacket padding using transceiver <xcvr> [NEX#58]*** Unable to determine the transmit interpacket padding. See error message for failure details. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
59	***Unable to determine receive interpacket padding using transceiver <xcvr> [NEX#59]*** Unable to determine the receive interpacket padding See error message for failure details. Verify that your standard transceiver database (**stdxcvr.xml**) has not been corrupted. An update might be available from the www.lonmark.org website.
60	***Unknown command ID <id> [NEX#60]*** This is an internal error condition.
61	***Cannot create file '<file>' (missing external utility <utility>) [NEX#61]*** A file <file> cannot be created because tool <utility> is missing, or cannot be found on the system search path.

NEX#	Description
62	***Cannot export 'clone domain' and 'no domain' (the two features are mutually exclusive) [NEX#62]*** When exporting an image as configured, this image can be exported with domain information, without domain information ('no domain'), or into the 'clone domain'. These three scenarios are mutually exclusive. See the Neuron chip or Smart Transceiver data book and the *IzoT NodeBuilder FX User's Guide* for details about exporting configured images.
63	***The configured image can not be exported authenticated in the chosen domain configuration [NEX#63]*** For an image to be exported authenticated, a domain ID (with a length of 1, 3, or 6 bytes), and a subnet/device id (different from 0/0) must be used unless the firmware supports open media authentication. See the Neuron Chip or Smart Transceiver data book and the *IzoT NodeBuilder FX User's Guide* for details about exporting configured images.
64	***An image can not be exported as configured, without domain, and with a 96 bit authentication key [NEX#64]*** Exporting configured and authenticated images requires the domain to be specified if a 96-bit authentication key is to be used. See the Neuron Chip or Smart Transceiver data book and the *IzoT NodeBuilder FX User's Guide* for details about exporting configured images.
65	***The currently chosen firmware does not support 96 bit authentication keys [NEX#65]*** You must choose a firmware version that supports this feature, or use a 48-bit authentication key instead.
66	***96 bit authentication keys requires two domain table entries [NEX#66]*** You cannot reduce the size of the domain table to one entry (by using the Neuron C compiler directive **#pragma num_domain_entries 1** and still export this device image using a 96-bit authentication key. You can use a 48-bit authentication key, or you must allow for both domain table entries to be created.
67	***At least one of domain ID, subnet ID, and device ID has not been specified but is required for this export configuration [NEX#67]*** The image cannot be exported as desired due to lack of data. For example, this would occur if an image should be exported in the configured state, and only the subnet and device ID, but no domain ID, was provided. Verify that to specify all data required, or to specify the correct state of the exported image.
68	***The firmware does not support 96 bit authentication keys [NEX#68]*** You must disable authentication, use a 48-bit authentication key, or use a firmware version that does provide support for 96-bit authentication keys.

NEX#	Description
71	***The chosen firmware does not support operation at 3.2768 or 6,5536MHz [NEX#71]*** The 6.5536 MHz clock rate is only supported in Version 14 firmware and later.
73	***The chosen combination of transceiver and Neuron model requires operation at 3.2768 or 6.5536 MHz [NEX#73]*** When using the PL31x0 chip, when combined with the PL-20A or PL-20A-LOW transceiver, you must use the 6.5536 clock rate. For other cases, please consult your Neuron Chip or Smart Transceiver data book.
74	***The <transceiver_name> transceiver requires a minimum clock rate of <clock_rate> MHz [NEX#74]*** The chosen transceiver requires the Neuron Chip or Smart Transceiver to operate at a certain minimum clock rate; please consult your transceiver or Smart Transceiver data book for details.
75	***The <transceiver_name> transceiver supports a maximum clock rate of <clock_rate> MHz[NEX#75]*** The chosen transceiver requires the Neuron Chip or Smart Transceiver to operate at a certain maximum clock rate; please consult your transceiver or Smart Transceiver data book for details.
76	***The <neuron_model> Neuron model requires a minimum clock rate of <clock_rate> MHz[NEX#76]*** The chosen Neuron or Smart Transceiver model cannot operate at the selected clock rate; instead the minimum clock rate indicated in the message is required.
77	***The <neuron_model> Neuron model supports a maximum clock rate of <clock_rate> MHz[NEX#77]*** The chosen Neuron or Smart Transceiver model cannot operate at the selected clock rate; the maximum clock rate is indicated in the message.
78	***Transceiver doesn't support attenuation measurements[NEX#78]*** Some powerline transceivers support an **–attenuation** command. This error is returned when this command is invoked on a transceiver that does not support it.
79	***I/O clock selection doesn't match the hardware clock*** This error occurs when the clock selection, defined with the pragma specify_io_clock compiler directive, does not match the hardware clock. This directive is generally not required for Series 5000 or 6000 chips, because the I/O clock is fixed for those chips.

NEX#	Description
80	***The chosen external clock cannot be used to drive the UART*** This error occurs when a clock selection, defined with the pragma specify_io_clock compiler directive, is not suitable to drive the on-chip UART. This directive is generally not required for Series 5000 or 6000 chips, because the I/O clock is fixed for those chips.
81	***The target hardware doesn't support the combination of external clock, clock multiplier and minor cycle divider*** This error occurs when the target hardware does not support the clock specified. This error usually indicates an incorrect build script or other form of direct calls to the Neuron Exporter.
82	***This version does not recognize size or format of this target's <system-structure>*** This error occurs as a result of a mismatch between the compiler, linker and exporter, and these tools' respective expectations of the target system. This error usually indicates an incorrect build script or other form of direct calls to the build tools.
83	***Merging the transient image into <target-file> requires an Intel-Hex file format*** This error occurs when a transient application image needs processing and a Motorola S-Record export file format has been requested. Some image files, including the transient image, do not support the Motorola S-Record file format natively. To generate application image files in this file format, export all build artifacts using the Intel-hex format (the default), and use third-party tools to convert the images which require the Motorola S-Record format.
84	***Application exceeds capacity by <num> bytes*** This error occurs when the application's transient image exceeds the maximum size allowed. Consult the list of supported serial flash memory parts in the series 6000 chip databook for alternative, larger, flash memory parts.
4001	***Inserting XIF appendix <filename> might have caused an unwanted duplication of records. Verify the XIF file for correctness [NEX#4001]*** Multiple XIF appendix files might have caused duplication of template and value file records. More than one of the following files contributed to the resulting XIF template and value file records: .BF2, .XF2, and a possible explicit XIF extension file (**--xifappendix** command). Verify the resulting XIF file, and remove explicit or implicit XIF appendix files if no longer needed.
4002	***Applicationless state disables some Reboot Options [NEX#4002]*** A device has been exported as applicationless. Some of the reboot options requested are meaningless for this configuration, and will be ignored.

NEX#	Description
4003	***Specified Reboot Options will prevent network loading [NEX#4003]*** The specified reboot options prevent loading of the application image over the network.
4004	***Old-style dependency file <filename>. Use .nxdep instead [NEX#4004]*** Old-style dependency file was specified. This file format is obsolete and should not be used any longer. Use **--nxdep**.
4005	***Ignoring superfluous .phd file (<filename>) [NEX#4005]*** Ignoring superfluous .phd file. Both a new and a legacy linker dependency file have been specified; the old format (.phd) is being ignored.
4006	***The selected feature requires exporting a configured image (feature ignored) [NEX#4006]*** A feature was requested that can only be used with a configured image; use the --state command to request exporting of a configured image.
4007	***Cannot update dependency file <file> (<reason>). This might cause the build status calculator to fail [NEX#4007]*** The Exporter's dependency file cannot be written due to the reason given in the message. The file should be made writable to prevent the build status calculator from failing.
4008	***No boot ID was specified. A random value (<value>) is being used instead [NEX#4008]*** A boot ID value was not explicitly specified. To prevent multiple versions of the same device to be exported with the same boot ID, the Exporter automatically allocates a boot ID randomly (value shown in the message). It is recommended to use the --bootid command to explicitly specify the desired boot ID value, or to use the Project Make Facilities' boot ID management feature.
4009	***The export configuration does not permit the authentication key, domain ID, subnet ID, or device ID to be set. These will be ignored and the image will be exported in the requested state [NEX#4009]*** The image cannot be exported in the desired state with all data provided. Superfluous data is being ignored, as the state has priority over the other parameters. For example, this could occur if an image is to be exported in a 'no domain' configuration although a domain ID has been specified.

NEX#	Description
4010	***The subnet/device ID has not been specified for the desired configuration. S/N = 1/1 is assumed [NEX#4010]*** This warning will occur whenever an image is to be exported into the cloned domain, without both subnet and device ID being given. A device that has the clone domain flag raised must still have a valid domain/subnet/device configuration. The warning indicates that the problem has been recognized, subnet/device ID 1/1 are being assumed, and the export continues.
4011	***No domain ID was specified for the desired configuration. A zero-length domain is assumed [NEX#4011]*** This warning will occur whenever an image is to be exported into the cloned domain, without a domain ID and/or domain ID length being given. A device that has the clone domain flag raised must still have a valid domain/subnet/device configuration. The warning indicates that the problem has been recognized, a zero-length domain is being assumed, and the export continues.
4012	***Both subnet and device ID must be zero for the desired configuration. The specified values are being ignored [NEX#4012]*** A non-zero value for the subnet ID and/or device ID has been specified, leading to an invalid configuration. These values are being ignored and the image is exported in the desired state. Verify that the image has been exported into the correct state, and/or specify the correct subnet/device ID values as needed.
4014	***The hardware semaphore and the__ lock{} construct are not operational at this clock setting [NEX#4014]*** The hardware semaphore (and the *__lock*{}construct) are only operational in clock configurations in which the interrupts are executed in a separate execution context from the application context. Requesting the semaphore (entering the *__lock* clause) will always succeed immediately for a non-operational semaphore.

9

Neuron Linker (NLD) and Neuron Librarian (NLIB) Errors

This chapter lists and describes the errors that can be reported by the Neuron Linker and Neuron Librarian.

Overview

Message codes shown as [NLD#<*number*>] in this chapter are shown as [NLIB#<*number*>] when they originate from the Neuron Librarian, but the numerical message identifier is always unique and unambiguous.

NLD and NLIB Errors

Table 11 lists the NLD and NLIB error codes.

Table 11. NLD and NLIB Error Codes

NLD#	Description
1	***System file limit exceeded [NLD#1]***
2	***Cannot access or read file 'neuron.typ' [NLD#2]***
3	***Neuron type (-t switch) must be set [NLD#3]***
4	***Custom image creation incompatible with some switches [NLD#4]***
5	***Custom image creation requires NEURON external memory [NLD#5]***
6	***Custom image creation requires base image name [NLD#6]***
7	***No object files specified [NLD#7]***
8	***The 'nv_in_addr' feature is not available with the selected firmware image [NLD#8]***
9	***The 'alias' feature is not available with the selected firmware image [NLD#9]***
10	***The 'preempt_safe' feature is not available with the selected firmware image [NLD#10]***
11	***The 'idempotent duplicate' feature is not available with the selected firmware image [NLD#11]***
12	***The new msgin fields feature is not available with the selected firmware image [NLD#12]***
13	***The 'transaction-by-address' feature is not available with the selected firmware image [NLD#13]***
14	***The '#pragma codegen nosiofar' feature is not available with the selected firmware image [NLD#14]***
15	***The '#pragma debug network_kernel' feature is not available with the selected firmware image [NLD#15]***

NLD#	Description
16	***The 'resp_in.addr' feature is not available with the selected firmware image [NLD#16]***
17	***The EEPROM lock feature is not available with the selected firmware image [NLD#17]***
18	***The 'msg_tag_index' and 'nv_in_index' feature is not available with the selected firmware image [NLD#18]***
19	***The 'read_only_data_struct' version is not available with the selected firmware image [NLD#19]***
20	***Write error on map file - disk full? [NLD#20]***
21	***Write error on sym file - disk full? [NLD#21]***
22	***Firmware symbol file did not specify system RAM usage [NLD#22]***
23	***Hardware or firmware selected does not support flash EEPROM [NLD#23]***
24	***Cannot create link-dependency file: <s> [NLD#24]***
25	***Write failed - disk full? [NLD#25]***
26	***Cannot find required symbols for Eval Kit special fixup [NLD#26]***
27	***Program error, code <ld> [NLD#31]***
32	***Error for seek operation on cached file [NLD#32]***
33	***Error reading cached file [NLD#33]***
34	***Error writing cached file [NLD#34]***
35	***File already open in CacheOpen [NLD#35]***
36	***Cannot open library file <s> [NLD#36]***
37	***Cannot open file <s> [NLD#37]***
38	***Cannot open the output file <s> [NLD#38]***
39	***Cannot open output debug file <s> [NLD#39]***
40	***Error closing input file [NLD#40]***
41	***Write error on output composite debug file - disk full? [NLD#41]***

NLD#	Description
42	***Debug output file write failed - is disk full? [NLD#42]***
43	***Farmalloc not allowed, s <lu> c <lu>\n [NLD#43]***
44	***Alloc size <ld> exceeds 64K for single element [NLD#44]***
45	***Farmalloc called: size=<ld>, count=<ld> [NLD#45]***
46	***Alloc size <ld> exceeds 64K [NLD#46]***
47	***IBits not allocated? [NLD#47]***
48	***IBits(<ld>) not allocated? [NLD#48]***
49	***Cannot create IBits file: <s> [NLD#49]***
50	***Write error to IBits file: <s> -- disk full? [NLD#50]***
51	***Cannot open image IB file <s> [NLD#51]***
52	***Bad record in file <s> [NLD#52]***
53	***Linker symbol table is too big [NLD#53]***
54	***Invalid symbol file format (line <ld>) in <s> [NLD#54]***
55	***Redefinition of <s> in file <s> [NLD#55]***
56	***File <s> is corrupt [NLD#56]***
57	***Symbol file write failed - disk full ? [NLD#57]***
58	***Init segment named <s> in file <s> is invalid [NLD#58]***
59	***Insufficient RAM for system requirements [NLD#59]***
60	***Absolute address in RAMFAR conflicts with system [NLD#60]***
61	***Cannot locate a RAM buffer for flash of size <ld> [NLD#61]*** On Neuron 3150 chips and 3150 Smart Transceivers that have off-chip memory, the system requires a RAM buffer to construct a complete flash page prior to writing. Failure to allocate this buffer is fatal; you must provide more RAM or change declarations in your application so that your application consumes less RAM, thus leaving more for the system. Changing the buffers in size of number also can reclaim the missing amount of RAM. See Chapter 8 of the *Neuron C Programmer's Guide* for more information on managing memory resources.

NLD#	Description
62	***Writable data areas placed in Flash memory by linker [NLD#62]***
63	***Writing data to Flash memory can cause delays, potentially leading to missed packets [NLD#63]***
64	***Writable data in Flash should be updated only rarely [NLD#64]*** These three diagnostics refer to writing to flash memory during regular application operation. When writing a flash page, the system must ignore incoming network traffic. In applications with a lot of incoming network traffic and small buffer counts, this might cause loss of an incoming packet. Write operations to flash memory should be kept to a minimum where possible.
65	***Cannot locate a buffer for debug kernel of size 13 [NLD#65]***
66	***Cannot relocate segment in file <s> [NLD#66]*** The NLD#66 error occurs when the linker has used up all available memory, and fails to allocate memory for the current segment. On a Series 5000 or Series 6000 Chip, where the memory assignment between RAM and non-volatile memory areas is controlled by software, you might be able to improve the memory map to meet your application's (and the linker's) requirements. In most cases, however, the memory map defines the physically available resources, which cannot be increased by redefining the hardware template. See Chapter 8 of the *Neuron C Programmer's Guide* for managing memory resources on a Neuron Chip.
67	***Symbol <s> is undefined [NLD#67]*** NLD#67 indicates that references to symbol <s> cannot be satisfied. Typically, this symbol would be referred in your application using an 'extern' specification. Inspect the set of function libraries that you offer in the link, or review the spelling of the symbol name in the 'extern' specification.
68	***Cannot find address of APINIT/APINITV [NLD#68]***
69	***APINIT/APINITV symbol is not defined [NLD#69]***
70	***APINIT/APINITV address vector not correct [NLD#70]***
71	***Image and/or library inits cannot be done [NLD#71]***
72	***Page number must be one or two hex digits only [NLD#72]***
73	***ROM size is fixed - cannot use '-z' switch with this Neuron model [NLD#73]***
74	***Page number for '-z' switch is too small for this Neuron model [NLD#74]***
75	***Offchip memory not permitted on <s> device [NLD#75]***

NLD#	Description
76	***Use of EEPROM for sys image requires 0 write time [NLD#76]***
77	***Off-chip ROM end must be at least page 0x<02X> [NLD#77]***
78	***Erroneous memory map settings [NLD#78]***
79	***Invalid memory map settings [NLD#79]***
80	***Cannot reserve onchip system RAM [NLD#80]***
81	***System RAM requirement exceeds available onchip RAM [NLD#81]***
82	***Unable to reserve space in EENEAR for system image [NLD#82]***
83	***Onchip EEPROM configuration memory requirement is excessive [NLD#83]***
84	***Consider reducing the number of alias entries [NLD#84]***
85	***Out of room in on-chip EEPROM data area [NLD#85]***
86	***Onchip EE Data may not exceed <ld> bytes for this firmware [NLD#86]***
87	***Cannot find address of EENEARBYTES [NLD#87]***
88	***No room in on-chip EEPROM for application checksum of <ld> byte<s> [NLD#88]***
89	***No room in on-chip EEPROM for memory map of <ld> bytes [NLD#89]***
90	***Application too large for on-chip EEPROM [NLD#90]***
91	***Cannot find address of BUFCOUNTS [NLD#91]***
92	***No Area in RelocSegment, segType=<c> [NLD#92]***
93	***Custom images may not use <s> area [NLD#93]***
94	***No more memory in offchip <s> area [NLD#94]***
95	***No more memory in onchip <s> area [NLD#95]***
96	***No more memory in <s> area [NLD#96]***
97	***Cannot have absolute addresses in CODE area [NLD#97]***
98	***Cannot have absolute addresses in EENEAR area [NLD#98]***
99	***Cannot have absolute addresses in RAMNEAR area [NLD#99]***

NLD#	Description
100	*Custom images can have absolute addresses only in ROM [NLD#100]*
101	*Absolute address <X> conflict in <s> area [NLD#101]*
102	*Absolute block from <X> to <X> not available [NLD#102]*
103	*Absolute block does not fit in <s> area [NLD#103]*
104	*Uninitialized data areas in Flash memory -- Check for interleave [NLD#104]*
105	*Too many search paths [NLD#105]*
106	*File <s> is not an object file [NLD#106]*
107	*File <s> is from an incompatible assembler [NLD#107]*
108	*Cannot link programs from Evaluation version LB [NLD#108]*
109	*The 'all_bufs_offchip' pragma requires Neuron with offchip RAM [NLD#109]*
110	*Program being linked must be from Evaluation version LB [NLD#110]*
111	*No file name for output - use '-o' switch [NLD#111]*
112	*Cannot write symbol file [NLD#112]*
113	*Cannot open file <s> for writing [NLD#113]*
114	*Cannot open image NX file <s> [NLD#114]*
115	*Cannot get Neuron memory to read base image [NLD#115]*
116	*Bad hexfile format in <s> [NLD#116]*
117	*Bad checksum in <s> [NLD#117]*
118	*Unknown record format in <s> [NLD#118]*
119	*The file <s> is not a valid library file [NLD#119]*
120	*Constrained segment cannot exceed 256 bytes [NLD#120]*
121	*Segment cannot be both onchip and offchip [NLD#121]*
122	*Overflow in fixupCmdBuffer [NLD#122]*
123	*Overflow in fixupValueBuffer [NLD#123]*

NLD#	Description
124	*Short branch offset <ld> is out of 0..15 range [NLD#124]*
125	*Offset <ld> is out of 8..23 range [NLD#125]*
126	*CALL to <lX> is larger than 0x1FFF [NLD#126]*
127	*Could use relative branch [NLD#127]*
128	*Could use 'CALLR' [NLD#128]*
129	*Could use 'CALL' [NLD#129]*
130	*Could use smaller sequence [NLD#130]*
131	*Byte value <ld> is out of 0..255 range [NLD#131]*
132	*Could use 'PUSHS' [NLD#132]*
133	*Byte offset <ld> is out of -128..127 range [NLD#133]*
134	*Could use small branch [NLD#134]*
135	*DATA.B byte <lX> is larger than 255 [NLD#135]*
136	*Bad fixup code <c> [NLD#136]*
138	*Invalid NEAR reference [NLD#138]*
139	*Bad fixup expression from <s>(<ld>) [NLD#139]*
326	*Option switches specified are not compatible [NLD#326]*
327	*List of module names ignored - no action options [NLD#327]*
328	*Version <ld> is not supported [NLD#328]*
329	*No module names to add [NLD#329]*
330	*Module name <s> is too long [NLD#330]*
331	*Cannot add module <s> - already in library [NLD#331]*
332	*Cannot add module <s> debug info - version 1 lib [NLD#332]*
333	*Module <s> debug info already exists in library [NLD#333]*
334	*Cannot add debug info - module <s> not in library [NLD#334]*
335	*Unknown file type <s> [NLD#335]*

NLD#	Description
336	*Too many object files (reading <s>) [NLD#336]*
337	*No module names to delete [NLD#337]*
338	*Did not find module <s>, cannot delete it [NLD#338]*
339	*Library does not contain debug info for <s> [NLD#339]*
340	*Cannot get file status for object file <s> [NLD#340]*
341	*Library symbol limit exceeded [NLD#341]*
342	*Library symbol character limit exceeded [NLD#342]*
345	*Cannot create a temporary file in PackLibrary [NLD#345]*
346	*Error writing temporary file in PackLibrary [NLD#346]*
347	*Error reading temporary file in PackLibrary [NLD#347]*
348	*Error in seek operation on library file [NLD#348]*
349	*Error reading library file [NLD#349]*
350	*Error writing library file - is disk full? [NLD#350]*
351	*Cannot open <s> file <s>: file missing or disk full [NLD#351]*
352	*Out of memory [NLD#352]*
353	*Out of memory - Use Extended Memory Option [NLD#353]*
354	*You must set the outfile name with the '-o' switch [NLD#354]*
355	*The '-o' switch has no effect with '-d' or '-s' switch [NLD#355]*
356	*Invalid input in switch file <s> [NLD#356]*
357	*Cannot nest switch files [NLD#357]*
358	*Cannot open switch file <s> [NLD#358]*
359	*You may only use -o once [NLD#359]*
360	*You MUST set the extension of the output file explicitly [NLD#360]*
361	*Did not find debug info for module <s> in library <s> [NLD#361]*
362	*Library <s> is an unsupported version [NLD#362]*

NLD#	Description
363	*Revision levels of files must match [NLD#363]*
364	*No control information available [NLD#364]*
365	*The '.dbt' extension can only be used for linked files [NLD#365]*
366	*Extended linked format requested, extended information not available [NLD#366]*
367	*The linked info in the input files is not used for '.dbg' [NLD#367]*
368	*The extension <s> is not valid for a debug file [NLD#368]*
369	*Program error - Found an enum tag that is multiply resolved [NLD#369]*
370	*No enum tag reference [NLD#370]*
371	*Program error - Found a struct/union tag that is multiply resolved [NLD#371]*
372	*No struct/union tag reference [NLD#372]*
373	*Tag <s> never got resolved [NLD#373]*
374	*Fixup list out of order [NLD#374]*
375	*Input file <s> is not a debug file [NLD#375]*
376	*Revision level <ld> of input file <s> is not supported [NLD#376]*
377	*The header in <s> conflicts with previous files [NLD#377]*
378	*The linked and header information will not be used [NLD#378]*
379	*The file <s> is not linked, unlike some previous files [NLD#379]*
380	*The linked information will not be used [NLD#380]*
381	*The file <s> is linked, but some previous files were not [NLD#381]*
382	*The extended header in <s> conflicts with previous files [NLD#382]*
383	*The extended header information will not be used [NLD#383]*
384	*File <s> has no name alpha table. [NLD#384]*
385	*File <s> has no name table. [NLD#385]*
386	*File <s> has no file name table. [NLD#386]*

NLD#	Description
387	*File <s> has no string table. [NLD#387]*
388	*File <s> string table size exceeds maximum of <lu> chars. [NLD#388]*
389	*File <s> has no scope table. [NLD#389]*
390	*Multiple files contain 'when' clauses [NLD#390]*
391	*Too many name table entries [NLD#391]*
392	*Too many file name table entries [NLD#392]*
393	*New string pool is full [NLD#393]*
394	*Duplicate tag def for <lu>:<lX> [NLD#394]*
395	*Name strlen <lu> (index <lu>) > 100 [NLD#395]*
396	*Strings at index <lu> and <lu> are dup [NLD#396]*
397	*Name alpha table bad order: pos <lu> (index <lu>) [NLD#397]*
398	*The symbol type is undefined [NLD#398]*
399	*Too many functions [NLD#399]*
400	*Segment is too large - exceeds 65,000 bytes [NLD#400]*
401	*The highest image address this linker supports has been exceeded [NLD#401]*
402	*The 'refresh_memory' feature is not available with the selected firmware image [NLD#402]*
406	*Cannot access or read file 'neuron.xml' [NLD#406]*
463	*Unspecified error in option processing [NLD#463]*
464	*Unspecified error in execution of librarian [NLD#464]*
465	*Unspecified error in execution of linker [NLD#465]*
466	*Insufficient space in EEPROM area for SIDATA [NLD#466]*
467	*Flash sector size invalid (must be 64 or 128) [NLD#467]*
468	*Image base name should not have an extension [NLD#468]*
469	*Too many characters in M switch param [NLD#469]*

NLD#	Description
470	*Invalid NEURON CHIP model name (-t), <s> [NLD#470]*
471	*Invalid number of banks, <d>; must be 2..<d> [NLD#471]*
472	*EE Write time (-y) must be 0..255 [NLD#472]*
473	*Invalid custom version, <d>; must be 128..254 [NLD#473]*
474	*The change-nv_len feature (set_nv_length function) is not available with the selected firmware image [NLD#474]*
475	*The system image extension 'proxy' is not available with the selected firmware image [NLD#475]*
476	*The system image extension 'phase' (or 'inverted_phase') is not available with the selected firmware image [NLD#476]*
477	*The system image extension 'nv_length_override' is not available with the selected firmware image [NLD#477]*
478	*The system image extension 'future' is not available with the selected firmware image [NLD#478]*
479	*The selected firmware image does not support the SCI or SPI I/O objects [NLD#479]*
480	*The selected firmware image does not support the IO_11 extended I/O pin [NLD#480]*
481	*The selected firmware image does not support the proxy route table [NLD#481]*
482	*The custom MAC feature was not linked from the libraries [NLD#482]*
483	*The selected firmware image does not support the RAM Test on Power-up only option [NLD#483]*
486	*The selected chip does not have extended RAM [NLD#486]*
489	*Can't allocate a RAM buffer for the ISR for SCI or SPI I/O device [NLD#489]*
494	*The set_lvi() function is not available with the selected firmware image [NLD#494]*
500	*The memory forwarding feature is not available with the selected firmware image [NLD#500]*
502	*Too many nested macros in library path: <s> [NLD#502]*

NLD#	Description
503	***Unauthorized attempt to use restricted feature. [NLD#503]***
504	***Resource <d> is not recognized and will be ignored [NLD#504]***
505	***The target chip does not support machine instruction set version <d> [NLD#505]***
506	***The target chip does not support all semaphores or interrupt sources required by this application [NLD#506]***
507	***The 'enable_io_pullups' directive is ineffective; there are no I/O pullups on this chip [NLD#507]***
508	***The stretched triac output model is not available with the selected firmware image [NLD#508]***
509	***The target hardware UART does not support all features required by a SCI or SPI I/O object declared in this application [NLD#509]***
511	***Too many network variables declared. Please check the maximum number of NVs allowed for the firmware version being used [NLD#511]*** If your device is using ver 16 or higher firmware, you can declare a maximum of 254 network variables and 127 aliases. If your device is using a firmware version lower than ver 16, you can declare a maximum of 62 network variables.
512	***Too many aliases declared. Please check the maximum number of aliases allowed for the firmware version being used [NLD#512]*** If your device is using ver 16 or higher firmware, you can declare a maximum of 127 aliases. If your device is using a firmware version lower than ver 16, you can declare a maximum of 62 aliases.
513	***The onchip service pin pull-up resistor cannot be disabled with this chip [NLD#513]***
514	***Flash driver symbol <s> not found [NLD#514]***
515	***Statement expansion must be turned on for this target in order to allow debugging [NLD#515]*** For Series 3100 Chips, you must enable the **Expand Statements** option in the **Compiler** tab of the **NodeBuilder Device Template Target Properties** dialog to debug this target. To open this dialog, right click the target device, click **Settings** on the shortcut menu, then select the **Compiler** tab.

NLD#	Description
516	***Statement expansion is not required for this target [NLD#516]*** For Series 5000 and Series 6000 Chips, you can reduce the size of your code by clearing the **Expand Statements** option in the **Compiler** tab of the **NodeBuilder Device Template Target Properties** dialog. To open this dialog, right click the target device, click **Settings** on the shortcut menu, then select the **Compiler** tab. When the **Expansion Statements** option is enabled, all Neuron C statements in your code are expanded to at least 2 bytes of machine code.
517	***The 'disable_journal_reset' pragma requires a system firmware that uses journaling [NLD#517]*** Consult the Neuron C Reference Guide for more details about this pragma directive.
518-519	***Compilation and link target chip mismatch [NLD#518]*** ***Compilation and link target system firmware version mismatch [NLD#519]*** Neuron C Compiler and Neuron Linker must be used with the same target specification. When necessary, recompile for each target.
523	***Transient segments must be relocatable [NLD#523]*** The Neuron Linker cannot link absolute transient segments. This error is probably caused by an incorrectly declared transient segment in Neuron assembly source. The Neuron C Compiler only generates relocatable transient segments.
526	***Cannot access or write transient segment image file <name> [NLD#526]*** This error occurs when the Neuron Linker cannot write the transient segment image file, and intermediate output used by the Neuron Exporter. Check that the file with the name indicated with this diagnostic is not write-protected and try to link again.
531-532	***Transient application symbol <name> is undefined [NLD#531]*** ***Transient system symbol <name> is undefined [NLD#532]***

NLD#	Description
536-541	***<name> is declared as a transient application function but implemented resident [NLD#536]*** ***<name> is declared as a transient application function but implemented as a transient system function [NLD#537]*** ***<name> is declared as a transient system function but implemented resident [NLD#538]*** ***<name> is declared as a transient system funciton but implemented as a transient application function [NLD#539]*** ***<name> is declared resident but implemented as a transient application function [NLD#540]*** ***<name> is declared resident but implemented as a transient system function [NLD#541]*** Various linker diagnostics referring to a mismatch between declaration and implementation of symbol <name>. The Neuron Linker attempts to resolve the conflict (a warning is issued) or aborts with an error.
542	***Generating I-Bits files (*.IB) is no longer supported [NLD#542]*** The Neuron Linker's *-i (--ibits)* option is no longer supported and ignored.
5433	***One or more memory boundaries have been specified but are ignored, because the target chip supports auto-tuning memory maps [NLD#543]*** Linking applications for a series 6000 Neuron Chip or Smart Transceiver or compatible targets invokes the *auto-tuning link algorithm*, which automatically allocates different memory types as required. The warning is given because memory boundaries were specified in the Neuron Linker invocation for such a target.
544	***Unknown link algorithm <id> [NLD#544]*** This error occurs because the firmware requires a link algorithm not recognized by this version of the Neuron Linker.
545	***The system firmware does not support offchip persistent data storage [NLD#545]*** The error occurs when *far eeprom* variables were declared in the Neuron C application (or EEFAR segments were used in Neuron assembly), linked with targets without support for this
546	***The application exceeds the supported offchip persistent data storage by <number> bytes [NLD#546]*** This error occurs when the amount of *far eeprom* variables exceeds the maximum supported by the firmware.

NLD#	Description
550	***Only <n> bytes remain for the heap. A minimum of <m> bytes is required [NLD#550]*** The heap is an area of memory within the Neuron Chip or Smart Transceiver which is used for dynamic memory allocation, primarily used for transient functions. The heap is automatically sized as the largest contiguous memory area after constant data, resident code and all *eeprom* and *RAM* segments are relocated. This error occurs because the resulting heap is insuffient. To increase the heap or reduce the heap size requirement, you can reduce the amount of resident code and data (for example by executing more code as transient functions), or you can break up large transient functions into multiple smaller functions.
553	***Link algorithm <name> not recognized. Try 'auto' (or 2) or 'traditional' (1) [NLD#553]*** This error occurs when the Neuron Linker is invoked with an invalid link algorithm identifier.

10

Project Make Errors (PMK)

This chapter documents and explains the warning and error messages reported by the Project Make component of the IzoT NodeBuilder software.

PMK Errors

Table 12 lists the PMK error codes.

Table 12. PMK Error Codes

PMK#	Description
100 **101** **102**	***Build failure [PMK#100]*** ***Build failure [PMK#101]*** ***Build failure [PMK#102]*** These error codes are generic error codes, which occur as a natural result of a build failure. The reason for the build failure will appear in one or more error messages that precede the generic message (typically the most recent message(s) before this message). After a build failure that was determined by some other service used by the make facility (such as the compiler, the assembler, the linker, or the exporter), the "target service" should have already issued a more helpful error message. Make aborts the build attempt and must return an error indication. The Make facility error indication will be a PMK#100, PMK#101, or PMK#102 message. In case this message occurs without any other error given,
104	***Can't find target '<target name>' in template <templatefile>. [PMK#104]*** Build target invalid, probably caused by invalid target name (should be "Development", "Release", and so on)
105	***Can't load the device template file <filename>[PMK#105]*** NodeBuilder device template file is invalid, does not exist, or is corrupt.
107	***Don't know what to do. No action specified.[PMK#107]*** No action given. Actions are "Build", "clean", and so on. See command line usage of the PMK project make facility, or type 'PMK -?' to obtain on-screen usage hints.
108	***Don't know what to build. Target missing.[PMK#108]*** No build target has been specified, but a build target is required for the requested action. The tool cannot operate without a build target being specified; use the -t (--target) command to specify the desired target. Targets are "Release", "Development", and so on.
110	***<Dependency message>[PMK#110]*** A dependency file cannot be read, or the file is corrupt. To fix, attempt the "clean" and "build unconditionally" commands. PMK combines error messages caused by dependency file access routines into this PMK#110 error message, but provides the full detail of the failure cause in the error message.

PMK#	Description
111	***Failure when launching LONUCL32.DLL[PMK#111]*** Failure attaching to the LONUCL32.DLL - make sure LONUCL32.DLL exists in the current Windows search path, and make sure no other application is attempting to build a target at the same time.
112	***Failure initializing <service>, code <code>.[PMK#112]*** Failure initializing the internal service <service> with failure code <code>. Verify that the target service exists in the current Windows search path (for example, LONNCC32.DLL, LONNAS32.DLL, LONNEX32.DLL, LONNLD32.DLL, and so on).
113	***Unknown <command-type> command (<numerical command id>)=<parameter>, or parameter is invalid or malformed. [PMK#113]*** Perform a "clean" operation and attempt to re-build.
114	***Cannot read hardware template file <file>.[PMK#114]*** The hardware template file is either missing or corrupt. Verify that the hardware template file <file> is present. Attempt to fix a possible corrupted hardware template file using NodeBuilder's hardware template editor.
115	***Cannot read project file <file>[PMK#115]*** Project file <file> is missing or corrupt.
116	***Cannot read standard Neuron type file <file>. [PMK#116]*** Neuron chip database file <file> cannot be found, is missing or is corrupt. The default name for this file is 'neuron.xml', and the default location is \LonWorks\Types (on whichever drive your LonWorks folder resides. Attempt to correct by re-installing the IzoT NodeBuilder software.
117	***Hardware template includes invalid Neuron key <key>. [PMK#117]*** The Neuron model indicated by the hardware template file seems invalid, or the neuron.xml database is corrupt. Also see discussion on PMK#116 above.
118	***Cannot create folder <folder>: <failure reason> [PMK#118]*** A folder <folder> needs to be created as part of the build process. This operation fails with the reason given in the error message.

PMK#	Description
119	***Cannot determine default firmware version for image <image>: <failure reason> [PMK#119]*** Failure to determine the firmware version to use as a default. The file 'default.ver', normally contained in the \Lonworks\images folder (on whichever drive your LonWorks folder resides) could be missing or could be corrupt. As a workaround, you can explicitly specify the firmware version in the target device preferences, do not use 'Default' as a firmware version.
120	***Cannot determine set of standard libraries from <liblistfile>: <failure reason> [PMK#120]*** Failure to determine standard neuron libraries from <liblistfile>. Verify that <liblistfile> (a text file), which defaults to \Lonworks\images\Stdlibs.lst, is present and is not corrupt.
121	***Cannot determine transceiver type. Verify preferences in device template and project. [PMK#121]*** Failure to determine the transceiver type. The project file and/or device template file might be corrupt. Edit these files and correct the transceiver preferences, specifying an explicit transceiver type (other than 'Default'), if needed.
122	***Cannot calculate signature, <detail> [PMK#122]*** The external interface signature cannot be calculated. This indicates missing or corrupt intermediate files (.BIF and .BF2 extensions), or missing or corrupt explicit XIF appendix files (XF2 extension). Verify the project and device preferences, and attempt to build unconditionally.
124	***Unknown macro switch record <record> received from <source> (try performing unconditional build). [PMK#124]*** Internal error: switch uses invalid macro value. To fix, attempt the "clean" and "build unconditionally" commands and
125	***Cannot compute dependency code for DRF catalog <catalog> and program ID <id> (LDRF error <code>). Verify that your LDRF catalog is valid and up-to-date. [PMK#125]*** The LDRF catalog cannot be accessed. Use the Resource Editor utility to make sure the catalog file is present and intact. The catalog file defaults to LonWorks\Types\ldrf.cat (on whichever drive your LonWorks folder resides). If the catalog file is missing, attempt to re-create one by using the MKCAT.EXE utility, that is available for download from the www.lonmark.org website.
126	***Malformed min/max model number specified in <file>. Correct preferences or disable automatic program ID management. [PMK#126]*** The program ID data given in the NB device template file <file> appears malformed. Edit and correct the preferences using NodeBuilder's device template file editor, or disable automatic program ID management.

PMK#	Description
127	***The specified program ID <id> appears malformed and invalid. [PMK#127]*** Use a simple ASCII string, or a byte-array format, using a single colon as a separator between each byte (for example, 94:56:78:9A:0B:0C:0D:0F).
130	***Missing hardware template. You must assign a hardware template to device <device>, build target <target>, first. [PMK#130]*** A build was attempted on a build target <target>, that belongs to the NodeBuilder device template <device>, where the hardware template has not yet been specified. Use NodeBuilder's device template editor to specify the hardware template, and re-attempt the build.
132	***Cannot read the hardware platforms database <file>. [PMK#132]*** Verify that the platform's database file <file> exists. The default location for this database file is \LonWorks\NodeBuilder\Templates\Hardware\nbplatforms.xml
133	***Library <lib> is required but cannot be found [PMK#133]*** The Project Make Facility has aborted the build process because a library <lib> is required, but cannot be found.
4001	***An empty program ID has been specified. It is recommended to change the program ID, using the device template editor [PMK#4001]*** An empty program ID, that is, a program ID without a value, has been specified. It is recommended to change the program ID, using the device template editor. This might result in a compilation failure. This message should not occur when using a machine-generated device template file.
4002	***The configured program ID results in an unstable build status, and may not suitable for automatic management. It is recommended to reconsider the specified program ID, or to disable program ID management. This might lead to a failure in the remaining build process [PMK#4002]*** This message indicates that program ID management causes an unstable build status. This loop condition has been detected after three fix-up compilation attempts, and the fix-up attempts have been aborted to prevent an endless loop condition. A fix-up compilation might occur as a result of automatic program ID management, where the PID manager detects an interface change after the initial compilation, where the interface change impacts the compiler's DRF lookup. Thus, a re-compilation is attempted with the new program ID, and in this particular case, the problem persists. This effect might be caused by the combination of automatic program ID management and a DRF scope value 6. This combination is not recommended for use in NodeBuilder 3, because program ID management modifies the model field, and the scope 6 resource file expects the model field not to change.

PMK#	Description
4003	***Can't write file <make dependency file>. This might cause the build status calculator to malfunction, but does not impact the build results (system error code <code>) [PMK#4003]*** Failure to write a .nkdep dependency file. Insufficient write-permission, or a write-protected media could cause this effect. This failure does not harm the build itself, but causes the build status calculator not being able to determine the build status correctly. Solution: make .nkdep file writable, or disable automatic program ID management.
4004	***Can't delete intermediate folder <folder>: <reason> (system error code <code>)[PMK#4004]*** Failure to complete a "clean" command. Write-protected files or folders in that area might cause this, or it could be caused by user-defined data in the intermediate folder(s) ("IM" folder(s)) or in the target folders ("Development" or "Release" folder(s)). The IzoT NodeBuilder tool attempts only to "clean" files produced by an IzoT NodeBuilder.
4005	***Can't delete intermediate file <file>: <reason> [PMK#1005]*** See PMK#4004, but for an individual file. Failure reasons include the file being locked by another process, the file being write-protected, and so on. Failure details are given in <reason>, part of the actual message being displayed.
4006	***The channel type number noted in the program ID <id> is <cid1>, whereas the transceiver is designed for channel type <cid2> [PMK#4006]*** The channel type ID field in the standard program ID describes a channel type that is different from the one referred to by the transceiver used by the current hardware template. This does not affect the build, but might result in an improperly implemented LONMARK device. This warning may only occur if a standard program ID is used (0x8* or 0x9* format).
4007	***The program ID model number field has reached the configured maximum of <maximum>. The previous program ID <previous_pid> will be changed to use the configured minimum model number <minumum>. Note that this might cause a failure when importing the external interface template into LNS [PMK#4007]*** Automatic program ID management detected a required program ID change and reached the end of the configured range. Program ID management proceeds by re-starting at the beginning of the specified range. This is likely to cause a problem when importing the external interface into LNS, you will have to delete the relevant LNS device template objects and reattempt the build.

PMK#	Description
4008	***The file <file> (<full path>) was previously required for a build, but can not be found. This might cause a build failure [PMK#4008]*** The build status calculator recognizes a file that was required for a previous build does not exist any more. This can possibly cause a build failure, and it could be a normal situation after purposeful removal of the file in question. For example, if the file in question were a .h file which is no longer included by the NC code, then this would be normal. If the **#include** statement still exists, then this might result in a compilation failure.
4009	***Unknown file type for <file> (<full path>) [PMK#4009]*** The build status calculator was unable to determine the type of a file. This indicates a corrupted dependency file. Proceed with build and attempt a re-build.
4010	***Can't open or write to build log file <file>. An existing build log file in this location might be out-of-date as a result of this failure [PMK#4010]*** Can't open or write to build log file <file>. An existing build log file in this location might be out-of-date as a result of this failure. Verify that the existing build log file is not write-protected.
4011	***Cannot produce link summary. The linker mapfile <mapfilename> is malformed. [PMK#4011]*** Attempt to correct the problem with a "clean", followed by a re-build. If PMK#4011 and PMK#4012 both appear together, attempt to address the condition causing the PMK#4011 before attempting to address the PMK#4012.
4012	***Cannot produce link summary. The linker mapfile <mapfilename> is missing [PMK#4012]*** Attempt to correct the problem with a "clean", followed by a re-build. This message might be a consequence of PMK#4011. If PMK#4011 and PMK#4012 both appear together, attempt to address the condition causing the PMK#4011 before attempting to address the PMK#4012.
4013	***The requested action <action> overrides a previous choice. [PMK#4013]*** This warning indicates that more than one, mutually exclusive, actions have been requested on the command line. Actions are -build, -clean, -compile, and -query. One and only one action must be given; a failure occurs if none is given (PMK#107). The most recent action is performed if more than one is given, noted with warning PMK#4013).

PMK#	Description
4014	***The device template file <filename_here> could not be updated; the file might be locked or write-protected. This does not impact the current build, but the build status calculator might determine an incorrect build status afterwards. It is recommended the file is made writeable, or only unconditional builds are performed. [PMK#4014]*** Write failure when updating the device template. This could be caused by a write-protected or otherwise not writable NodeBuilder device template file (.nbdt extension). The update failure causes a possible, subsequent, failure in automatic boot ID management and automatic program ID management. It is recommended to make the NodeBuilder device template file writable, or to disable boot ID management and program ID management.
4015	***The device template file <name> might be corrupted: property <propertyname> can not be read correctly. A default value will be used instead [PMK#4015]*** A possible file corruption occurred in the NodeBuilder device template file (.nbdt extension), a property cannot be read correctly. See warning message for details. Attempt to correct the problem by opening the device template file in a device template editor and save it again.
4016	***Possible data corruption in device template, field <fieldname> [PMK#4016]*** See PMK#4015.
4017	***Cannot copy file <sourcefile> to <destination>: <reason> [PMK#4017]*** A file cannot be copied for reasons given in the actual message. Files are only copied for convenience; copied files include the linker map or the application symbol table.
4018	***More than one XIF appendix file has been found in the source folder, although only one can be added to the XIF file. The superfluous file <filename> will be ignored [PMK#4018]*** See documentation for more details about XIF appendix files and recommended procedures when maintaining legacy applications.
4019	***Library <lib> might be required but cannot be found [PMK#4019]*** Library <lib> is recommended but may not be required. The build proceeds. If the build fails with linker errors, reporting unresolved references, this warning about the missing library file could be related to the linker failure. Verify that the required library is present in the expected folder.

11

Common Command Line Errors (UCL)

This chapter lists and describes errors that can be reported by the common command line system. The common command line system is used in the commands “ncc”, “nas”, “nld”, “nex”, “nlib”, “pmk”, and others.

UCL Errors

Table 13 lists the UCL error codes.

Table 13. UCL Error Codes

UCL#	Description
1	***service is locked [UCL#1]*** The UCL engine is locked. The UCL engine LONUCL32 cannot reference itself. Verify that to specify the correct target UCL service other than LONUCL32;
2	***service not found. [UCL#2]*** The targeted UCL service cannot be found. Verify that the service DLL is in a folder contained within the current user's search path, and make sure the DLL exists.
3	***target service locked. [UCL#3]*** The target UCL service is already in use, and does not support multiple concurrent clients. Wait for the first client to detach, and attempt to re-attach thereafter.
4	***target service fails to initialize [UCL#4]*** The target service cannot be initialized correctly. Such failure could be caused by a required but missing DLL, or some other service-dependent initialization failure.
5	***invalid command, or malformed parameter [UCL#5]*** Invalid option. An unknown command was sent to the target UCL service. Check the service documentation for supported commands and options.
6	***invalid value [UCL#6]*** An invalid parameter value was provided. Check the service documentation for supported commands and their parameters.
7	***the requested feature is not available [UCL#7]*** The targeted service does not support the feature required to fulfill the request. For example, the targeted service might not provide help strings for on-screen usage hints, or it might not support parameters with a particular data type.
8	***service not initialized. [UCL#8]*** An attempt has been made to use an uninitialized UCL service. This is an internal error condition.

UCL#	Description
9	***wrong context. [UCL#9]*** A UCL server operation has been requested out of context. This is an internal error condition.
10	***Command not understood: <cmd> [UCL#10]*** Failure in command parser – see error message for details. Commands on the command line or in a command file cannot be understood due to a syntax error. See on-screen usage hints or printed documentation for a listing of supported command line parameters.
11	***Bad command set [UCL#11]*** Some commands are missing (but required), non-accumulative commands have been given more than once, and so on. Check the target service documentation for the supported and required commands. This is an internal error condition,
12	***Can't attach to command file <file> [UCL#12]*** ***Service <service> has invalid VERSIONINFO resource [UCL#13]*** An operating system error occurred when accessing a command file or a UCL service. Check the filenames, and make sure that no other process holds exclusive access rights to those files at the same time.
13	***Fail to unload service DLL <service>: <reason> [UCL#13]*** UCL failed to unload the UCL service named in the message, for reasons detailed in the message.
100-999	These numbers are reserved for service-specific error codes, check the documentation of the specific service for details.
4001	***Can't open or write to build log file <filename>. An existing build log file in this location might be out-of-date as a result of this failure [UCL#4001]*** Cannot create build log file. See warning message for detailed error description. This message is benign as it doesn't affect the service's operation at all, but it might lead to an incorrect or outdated log file.
4002	***The service <servicename> was run on a possibly incorrect build script <scriptfilename> [UCL#4002]*** The automatically generated script might be based on a bad build. This relates to a command script file that was produced with the --mkscript command, and the build or build attempt which produced that command script file did not complete without error. Warning UCL#4002 is given when this command script file is used in an attempt to perform a script-driven build (or whatever action the script requests), suggesting the script might be bad.

UCL#	Description
4003	***Skipping <file> to avoid endless recursion [UCL#4003]*** Command file recursion. A loop condition was detected when parsing command files, and the file named in the warning message was excluded from repeated processing to prevent an endless command file loop to occur.
4004	The content of this warning is user-defined. A warning message has been specified by the UCL client, using the --warning command. The message content is controlled by the caller (a command script, for example), and not under control of UCL.

12

Neuron Firmware Error Codes

This chapter lists and describes the Neuron Chip firmware system error messages. These error messages do not have a three-letter code associated with them.

Overview

Every application reserves one byte of on-chip EEPROM memory space to hold the error log. If the firmware posts an error, it is usually due to a severe problem. A network diagnostics tool could periodically collect the error log and device statistics to monitor the health of the system.

An application can also post errors, using the **error_log( )** function. Only the last error posted is saved. Users are allocated error numbers 127 and below.

You can use the LonMaker Device Manager in the IzoT Commissioning tool to fetch the error log and other statistics from a device. To do this, right-click the device, click **Manage** on the shortcut menu, and then click **Test** in the **LonMaker Device Manager** dialog. For more information on using the LonMaker Device Manager, see Chapter 8 of the *IzoT Commissioning Tool User's Guide*.

The NodeBuilder Debugger shows any posted errors in the **Events Log** tab on the Results pane.

Neuron Firmware Errors

Table 14 lists the Neuron Firmware error codes.

Table 14. Firmware Error Codes

Code	Description
45	***io_access() violation*** A call to **io_access()** has occurred when the ISR context is not scheduled (as would be the case for the two lowest clock rate settings). Executing the **io_access()** functionality could deadlock the system, so it is skipped and the error is logged.
46	***Stack Collision/Underflow/Overflow*** The data stack and return stack have collided, underflowed, or overflowed. The system resets and post-reset this error is logged.
47	***Illegal opcode executed.*** Most likely a program crash. The system resets and post-reset this error is logged.
48	***ISR resource violation.*** The following operations are forbidden from an ISR: Allocation of an outgoing message buffer. The allocation fails. Network variable updates or polls. The setting of the network variable "update bit" is denied. Any writes to non-volatile memory. The write operation is skipped.

Code	Description
49	***Breakpoint in an ISR.*** The error is logged, and the ISR context resets the device. Post-reset the error is detected and the device's state is changed to hard-offline. This is to prevent a possible endless repetition of this condition.
50	***System Image Write Protect.*** Writes to the system image are trapped, the error is logged, and the device resets.
129	***Bad event.*** This run-time error is checked only in the development environment. This error could occur if the network configuration is invalid, if the network management tool is malfunctioning, or if the Neuron Chip firmware image is corrupted. If this error occurs, try reloading the device.
130	***NV length mismatch.*** This error may occur rarely due to network transmission problems. The length of the data in a network variable update message is inconsistent with the length expected by the device. Rebuild and reload the images if this error continues to occur. Note this error does not get set in conjunction with the change of a network variable type for a NV of changeable type. However, if such type change is performed in an inappropriate manner, this error might be logged upon the first arrival of the incorrectly sized network variable data.
131	***NV message too short.*** This error may occur rarely due to network transmission problems. This error could occur if a network message is corrupted.
132	***EEPROM write failure.*** This error may occur rarely due to Neuron Chip, transceiver, or application failure. This error occurs if too many erase/write cycles have been performed on the EEPROM or flash memory. Up to 10,000 erase/write cycles per byte can be performed in the on-chip EEPROM. This error will also be logged if an EEPROM write is attempted when a device is online and has EEPROM locking enabled.
133	***Bad address type.*** This error could occur if the network configuration is invalid, if the network management tool is malfunctioning, or if the Neuron Chip firmware image is corrupted. If this error occurs, try reloading the device.

Code	Description
134	***Preemption mode timeout.*** This system error is logged by the Neuron Chip firmware. The program ran out of buffers and the system gave up trying to get them. Increase the device timeout if this message occurs often. This error causes a reset.
135	***Already preempted.*** This system error is logged by the Neuron Chip firmware. If a program is already in preemption mode and tries to initiate another message, this error is generated. This error causes a Neuron Chip reset.
136	***Synchronous NV update lost.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. A synchronous network variable update was lost because the device was already in preemption mode.
137	***Invalid response allocation.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. This error occurs if an application program tries to allocate (build) a response when it hasn't received a request.
138	***Invalid domain.*** This error could occur if the network configuration is invalid, if the network management tool is malfunctioning, or if the Neuron Chip firmware image is corrupted. If this error occurs, try reloading the device.
139	***Read past end of message.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. This error occurs if an application program tried to read beyond the specified length of the message.
140	***Write past end of message.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. This error occurs if an application program tried to write past the specified end of the message.
141	***Invalid address table index.*** This error could occur if the network configuration is invalid, if the network management tool is malfunctioning, or if the Neuron Chip firmware image is corrupted. If this error occurs, try reloading the device.

Code	Description
142	***Incomplete message.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. This error occurs if an application program tries to send a message without first setting the code or data fields of the **msg_out** structure.
143	***NV update received for output network variable.*** This error may occur rarely due to network transmission problems. Another device tried to update an output network variable.
144	***No message available.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. This error occurs if an application program tries to reference the **msg_in** message object when no **msg_arrives** event has occurred.
145	***Illegal send.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. This error occurs if an application program tries to send a response or a message without first building one.
146	***Unknown PDU.*** This error may occur rarely due to network transmission problems. This run-time error is only checked in the development environment. This error could occur if a packet was corrupted on the network, but the CRC was valid.
147	***Invalid NV index.*** This run-time error is checked only in the development environment. This error could occur if the network configuration is invalid, if the network management tool is malfunctioning, or if the Neuron Chip firmware image is corrupted. If this error occurs, try reloading the device.
148	***Divide by zero error.*** This system error is logged by the Neuron Chip firmware. The application program executed a division by zero. This error is not reported by the floating point or 32-bit extended arithmetic library functions
149	***Invalid application error.*** This system error is logged by the Neuron Chip firmware. This error occurs if an application program tries to log an application error with an error out of range. The legal range is 1 to 127.

Code	Description
151	***Write past end of network buffer.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. The outgoing application message could not fit into the outgoing network buffer. The maximum length is 255 bytes.
152	***Checksum error over application program.*** This error may occur rarely due to Neuron Chip, transceiver, or application failure.
153	***Checksum error over configuration data.*** This error may occur rarely due to Neuron Chip, transceiver, or application failure. The Neuron Chip retains a checksum of the application program and of the configuration data. If it is not the correct value, an error is logged, and the device goes into a blank or unconfigured state. This is usually a hardware problem, although it could be caused by the application writing over itself. See the section *Defining Reboot and Integrity Options* in Chapter 7 of the *LonBuilder User's Guide* for a discussion of checksums and other integrity features.
154	***Transceiver register address out of range.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. The valid range for transceiver status information is 1 through 7.
155	***Transceiver register operation timeout occurred. ** This error may occur rarely due to Neuron Chip, transceiver, or application failure. A transceiver hardware failure occurred and the transceiver could not be configured.
156	***Application buffer too small.*** This system error is logged by the Neuron Chip firmware. A message was received into a network buffer but it could not fit into an application buffer. May need to increase the buffer size with the **#pragma app_buf_in_size** directive (see the *Compiler Directives* chapter in the *Neuron C Reference Guide*).
157	***io_in or io_out not ready.*** This system error is logged by the Neuron Chip firmware. This run-time error is checked only in the development environment. Function **io_in()** or **io_out()** invoked for a parallel I/O object when not in the proper state for input or output, respectively.

Code	Description
158	***Self test failed.*** This error may occur rarely due to Neuron Chip, transceiver, or application failure. The Neuron failed its self test. The self test includes tests of RAM and internal timer and counter logic. Note that this error code does not get set as a result of the RQ_SELF_TEST command being sent to the device object of an interoperable device.
160	***Authentication mismatch.*** A network variable message or network management message was rejected because of an authentication failure. Could be due to an authentication key mismatch or a lack of an authentication indicator in the original message. This error could indicate attempts of an intruder to "break in" to the network. This error could also occur if the network configuration is invalid, if the network management tool is malfunctioning, or if the Neuron Chip firmware image is corrupted. If this error occurs, try reloading the device.
161	***Self-installation semaphore.*** This value appears temporarily in the error log as part of normal Neuron Chip operation and it should not be construed as an error. Can appear during invocation of self-installation functions.
162	***Read write semaphore.*** This value appears temporarily in the error log as part of normal Neuron Chip operation and it should not be construed as an error. Can appear during reload of an application.
163	***Application image inconsistency.*** This system error is logged by the Neuron Chip firmware. This error occurs if the application code in ROM is inconsistent with the code in EEPROM. This is most likely due to an attempt to load a new application over the network without first reprogramming the EEPROM.
164	***Conflicting router S/W versions.*** This system error is logged by the Neuron Chip firmware. This error occurs when one attempts to connect two router halves with different software versions. This error only occurs in routers.
166	***EEPROM recovery occurred.*** This error may occur rarely due to Neuron Chip, transceiver, or application failure. This error is logged when the on-chip EEPROM is reloaded following a system error as defined by the EEPROM reboot word.

Code	Description
167	***Triac clockedge +- not supported.*** This system error is logged by the Neuron Chip firmware. This error is logged when an application using the **triac clockedge** plus/minus feature is loaded into a 3150, which does not support this feature.
168	***Checksum error over system image.*** This error may occur rarely due to Neuron Chip, transceiver, or application failure. This error is logged when the device goes application-less following a checksum error in the system image.
169	***Invalid proxy routing table index.*** This system error is logged by the Neuron Chip firmware.
170	***Invalid version.*** This system error is logged by the Neuron Chip firmware. Application was linked for a different version of firmware than is in the device.
173	***io_access() violation.*** A call to **io_access()** occurred when the interrupt service routine (ISR) context was not scheduled (that is, for the two lowest clock rate settings). Completing the **io_access()** call could deadlock the system, so it is ignored and the error is logged.
174	***Stack Collision/Underflow/Overflow.*** The data stack and return stack have collided, underflowed, or overflowed. The system resets, and logs this error after the reset.
175	***Illegal opcode executed.*** Likely caused by a program crash. The system resets, and logs this error after the reset.
176	***ISR resource violation.*** The following operations are not allowed within an interrupt service routine (ISR): • Allocation of an outgoing message buffer. The allocation fails. • NV updates or polls. The setting of the NV "update bit" is denied. • Any writes to non-volatile memory (NVM). The write operation is ignored.
177	***Breakpoint in an ISR.*** The error is logged, and the interrupt service routine (ISR) context resets the device. After the reset, the error is detected and the device's state is changed to hard-offline. The state change prevents a possible endless repetition of this condition.

Code	Description
178	***System Image Write Protect.*** Writes to the system image are trapped, the error is logged, and the device resets.
192-223	***State byte semaphore.*** This value appears temporarily in the error log as part of normal Neuron Chip operation and it should not be construed as an error. This appears when a device's state byte is being modified.