**Ethernet Example Tutorial**

# Introduction

This document explains the operation of the Ethernet threads in the ExampleProject project. This example and tutorial is meant to provide the developer with a quick-start guide to working with Ethernet communications and using the UDP protocol with a Primary/Secondary device setup. The Primary/Secondary setup in this case includes an echo step in the communications process which guarantees packet delivery from Primary to Secondary. The purpose of this is to ensure that there is no loss of any critical data transfer. An example might be turning on an end mill in a CNC machine. If for any reason this packet is corrupted or dropped, the machine can suffer costly damage. It’s up to the developer whether they need this, or some even more complete reliability measures. The reader may also notice that this reliability scheme involves a complete packet echo, as opposed to something less costly, like an ACK packet. This was chosen in order to minimize the probability of data corruption when used with critical systems. This example should provide enough information for future developers to create their own system with more or less overhead and reliability as their application demands.

This document assumes the reader has already cloned the Tutorials repository and ExampleProject project into their e2 Studio environment, and is familiar with the process of creating their own unique project based on this. For more information on that, please refer to the Project Creation Tutorial located in the Documents folder of the Tutorials Repo. To reach the Tutorials repo, please go to “<https://github.com/NYUAD-LabOps/Tutorials>”.

# Overview

The Ethernet example consists of two threads, EthernetPrimary and EthernetSecondary, that communicate with each other via the loopback address “127.0.0.1”. These threads simulate the actions of two real-world devices. In this case, the Primary sends a string to the Secondary every two seconds. This string contains an ASCII representation of an integer which begins at 1 and is incremented with each transmission. The Secondary receives this transmission and echoes it back to the Primary. Both threads print out statements to the console for each step in the interaction, so the developer can confirm that the application is working.

Prior to any sending operations, both threads wait for the Ethernet/IP stack to be initialized, then go through a series of API calls which initialize a socket, bind it to a port number, and tie this port to a receive callback. Initialization of the Ethernet/IP stack itself is handled automatically. This auto-init behavior is a feature enabled in the configuration.xml settings.

# **EthernetPrimary Thread**