LANTRONIX®

# NTC-500 and NTC-550 Series Software Developer Guide

**Part Number PMD-00360**
**Revision A June 2026**

## Intellectual Property



*Lantronix* is a registered trademark of Lantronix, Inc. in the United States and other countries.

Patented: https://www.lantronix.com/legal/patents/. Additional patents pending.

## Warranty

For details on the Lantronix warranty policy, please go to our web site at https://www.lantronix.com/support/warranty/.

## Contacts

Lantronix, Inc.
48 Discovery, Suite 250
Irvine, CA 92618, USA
Toll Free: 800-526-8766
Phone: 949-453-3990
Fax: 949-453-3995

**Technical Support**

Online: www.lantronix.com/technical-support/

**Sales Offices**

For a current list of our domestic and international sales offices, go to the Lantronix website at https://www.lantronix.com/about-us/contact/.

## Disclaimer

All information contained herein is provided "AS IS." Lantronix undertakes no obligation to update the information in this publication. Lantronix does not make, and specifically disclaims, all warranties of any kind (express, implied or otherwise) regarding title, non-infringement, fitness, quality, accuracy, completeness, usefulness, suitability or performance of the information provided herein. Lantronix shall have no liability whatsoever to any user for any damages, losses and causes of action (whether in contract or in tort or otherwise) in connection with the user's access or usage of any of the information or content contained herein. The information and specifications contained in this document are subject to change without notice.

## Revision History

| Date | Rev. | Comments |
|---|---|---|
| June 2026 | A | Initial Document |

For the latest revision of this product document, please check our online documentation at https://www.lantronix.com/support/documentation/.

# Contents

# 1. Using the SDK

## Introduction

This document serves as a guide to those wishing to develop or customise software to run on a Lantronix Wireless product. The SDK itself is a relatively simple wrapper around the Lantronix Wireless toolchain. It is intended to simplify the learning curve when compared to the bare cross-toolchain.

As such, it includes the cross-toolchain (binutils, gcc, g++, glibc, etc.), selected sections of the source code, the pre-compiled components of a full system and a set of hierarchical make files.

The SDK includes the kernel source, to facilitate the compiling of additional driver modules. Other source code can be supplied on request - most of the system is licensed under BSD, Apache, GPL, LGPL and similar licenses.

## Offer to supply source code

The software included in this product contains copyrighted software that is licensed under the GPL. You may obtain the complete corresponding source code from us by contacting Lantronix via: www.lantronix.com/technical-support/.

This offer is valid to anyone in receipt of this information.

# 2. Prerequisites

## Linux PC Requirements

Before proceeding with the SDK setup, ensure that you have a Linux PC prepared for the task. The system should meet the following requirements:

- OS: A Linux-based operating system (e.g., Ubuntu, CentOS, Fedora, etc.)
- Disk Space: Ensure sufficient disk space for the SDK and all associated files (at least 10 GB recommended).
- Memory: A minimum of 8 GB RAM for smooth performance.
- Network Access: Ensure the machine has internet access to download the SDK and any dependencies.

## Software/Tools/Applications

### Docker

Docker is required to create the build environment for compiling the SDK.

To install Docker, you can run the following command:

```
sudo apt-get install docker.io
```

### Git (Optional)

Git is used for version control and accessing any necessary source code repositories.

To install Git, use the following command:

```
sudo apt-get install git
```

### SSH/SCP

These tools are required to copy SDK packages to target devices via secure shell or secure copy protocol.

To install SSH and SCP, use the following command:

```
sudo apt-get install openssh-client openssh-server
```

# 3. Installing and Using Docker

## Install Docker on Ubuntu/Debian

To install Docker on your Ubuntu or Debian-based system, run the following commands:

```
sudo apt update
sudo apt install -y docker.io
```

## Start Docker Service

Start the Docker service and enable it to start on boot:

```
sudo systemctl start docker
sudo systemctl enable docker
```

## Add User to Docker Group

If you want to run Docker commands without sudo, you can add your user to the Docker group. Run the following command:

```
sudo usermod -aG docker $USER
```

After running this command, log out and log back in to apply the changes.

## Verify Docker Installation

To verify that Docker was successfully installed, run the following command:

```
docker –version
```

# 4. Loading the Cassini Docker Build Image

The SDK uses a prebuilt Docker image that contains the required cross-toolchain and build dependencies.

## Download the Docker Image

Download the Docker image archive from the following locations:

- https://www.lantronix.com/products/ntc-500-series/#product-resources
- https://www.lantronix.com/products/ntc-550-series/#product-resources

Downloaded example file:

docker-cassini-1.11.tar.xz

## Extract the Docker Image Archive

To extract the Docker image archive, use the following command:

```
tar -xf docker-cassini-1.11.tar.xz
```

## Load the Docker Image

Once the archive is extracted, load the Docker image into your local Docker repository with the following command:

```
docker load -i docker-cassini-1.11.tar
```

## Verify the Image is Loaded

To verify that the Docker image has been successfully loaded, run the following command:

```
docker images
```

Expected:

docker-registry:5000/cassini:1.11

# 5. Obtain the SDK folder for the Desired Release

Download the SDK from Lantronix site at https://www.lantronix.com/products/ntc-500-series/#product-resources or https://www.lantronix.com/products/ntc-550-series/#product-resources.

# 6. Copy the Downloaded SDK File to the Linux PC

Transfer the downloaded SDK zip file to the Linux PC where you intend to compile the SDK.

Example SDK File:

Cassini_ntc_552_3.3.1.0R9_others.tar.gz

# 7. Extract the SDK Files on the Linux PC

Use the following command to extract the example tar file:

```
tar -xvf  SDK_Cassini_ntc_552_3.3.1.0R9.tar.bz2
```

After extraction you will have SDK_Cassini_ntc_552_3.3.1.0R9 directory

Go to SDK_Cassini_ntc_552_3.3.1.0R9 directory using the command:

```
cd SDK_Cassini_ntc_552_3.3.1.0R9
```

It contains the following files:

- dependencies.sh
- examples
- hosttools
- Makefile
- Projects
- README.md
- staging_l
- staging_l_openwrt
- staging_l_qcle

# 8. SDK Testing Methods

This chapter provides different SDK testing methods with descriptions and instructions to execute them.

## Method 1: Copy an example project to projects directory and compile available example project

### Test Objectives

To understand the basic SDK compilation flow by copying an already available example project into the projects directory and compiling it without modifying the source code or Makefile. This method helps verify that the SDK setup, build environment, common Makefile rules, and IPK generation process are working correctly.

### Steps

1. Go to root directory i.e. SDK_Cassini_ntc_552_3.3.1.0R9 for example.

```
cd SDK_Cassini_ntc_552_3.3.1.0R9
```

2. Run make clean
3. Copy any example project for e.g. helloworld into projects/myapp directory

```
cp -r examples/helloworld projects/myapp
```

4. Check the contents of projetcs/myapp

```
ls projects/myapp

helloworld.c
Makefile
postinst
postrm
preinst
```

5. Execute make command and wait till compilation completes.

```
make
```

6. Check for the ipk file under image directory

```
ls images/

helloworld_1.0_ntc_552.ipk
```

7. Sign the generated .ipk files using the appropriate signing key as required by the platform (refer to Section 10).
8. Install the signed .ipk file on the device (refer to Section 11).

## Method 2: Create our own source file and edit Makefile

### Test Objectives

To create a custom C source file inside an existing example project structure and modify the Makefile so that the SDK builds the new source file instead of the default example file. This method helps understand how PKG_NAME, PROJECT_TYPE, and SOURCES are used to generate a custom binary IPK package.

### Steps

1. Go to root directory i.e. SDK_Cassini_ntc_552_3.3.1.0R9

```
cd SDK_Cassini_ntc_552_3.3.1.0R9
```

2. Run make clean

```
make clean
```

3. Copy any example project for e.g. helloworld into projects/myapp directory

```
cp -r examples/helloworld projects/myapp
```

4. Check the contents of projetcs/myapp

```
ls projects/myapp

helloworld.c
Makefile
postinst
postrm
preinst
```

5. Create your own source file

```
vi mysrcfile.c
```

6. Write a simple C program for demo

```
e.g. #include <stdio.h>
int main(int argc, char *argv[])
{
    printf("My simple C program for demo!\n");
    return 0;
}
```

7. Edit Makefile using following commands:

```
cat > Makefile << 'EOF'
PKG_NAME := mysrcfile
PROJECT_TYPE := BINARY
SOURCES := mysrcfile.c
include ../common.mk
EOF
```

8. Execute make command and wait till compilation completes

```
make
```

9. check for the ipk file under image directory

```
ls images/

mysrcfile_1.0_ntc_552.ipk
```

10. Sign the generated .ipk files using the appropriate signing key as required by the platform (refer to Section 10)
11. Install the signed .ipk file on the device (refer to Section 11)

## Method 3: Create from scratch

### Test Objectives

To create a completely new project from scratch inside the projects directory by adding a minimal Makefile and source file. This method helps to understand the minimum files and configuration required to build a new application using the SDK common build system.

### Steps

1. Go to root directory i.e. SDK_Cassini_ntc_552_3.3.1.0R9

```
cd SDK_Cassini_ntc_552_3.3.1.0R9
```

2. Run make clean

```
make clean
```

3. Create new project directory.

```
mkdir -p projects/myproject
```

4. Go to projects/myproject directory

```
cd projects/myproject
```

5. Create minimal Makefile

```
cat > Makefile << 'EOF'
PKG_NAME := myproject
PROJECT_TYPE := BINARY
SOURCES := myproject.c
include ../common.mk
EOF
```

6. Create a new C code file

```
vi myproject.c
```

7. Write a small code

```
e.g. #include <stdio.h>
int main ()
{
  printf("my project");
}
```

8. Go to root directory i.e. SDK_Cassini_ntc_552_3.3.1.0R9

```
cd ../..
```

9. Execute make command and wait till compilation completes.

```
make
```

10. Check for the ipk file under image directory.

```
ls images/

myproject_1.0_ntc_552.ipk
```

11. Sign the generated .ipk files using the appropriate signing key as required by the platform (refer to Section 10)
12. Install the signed .ipk file on the device (refer to Section 11).

## Method 4: Add any custom CFLAGS, LDFLAGS in the project Makefile

### Test Objectives

To add custom compiler flags and linker flags in the project Makefile using CFLAGS and LDFLAGS.

This example uses the standard C math library, libm, to verify that:

- CFLAGS can be used to pass compile-time macros
- LDFLAGS can be used to link an external library

Example: Using Math Library with CFLAGS and LDFLAGS

In this example, the application uses sqrt() from the math library. The application must be linked with libm by adding -lm to LDFLAGS.

## Steps

1. Go to root directory i.e. SDK_Cassini_ntc_552_3.3.1.0R9

```
cd SDK_Cassini_ntc_552_3.3.1.0R9
```

2. Run make clean.

```
make clean
```

3. Create new project directory.

```
mkdir -p projects/myproject
```

4. Go to projects/myproject directory

```
cd projects/myproject
```

5. Create a new C code file

```
vi main.c
```

6. Write a small code.

```
#include<stdio.h>
#include<math.h>
#ifndef INPUT_VALUE
#define INPUT_VALUE 0.0
#endif

int main(void)
{
    double result = sqrt(INPUT_VALUE);

    printf("%s: sqrt(%.2f) = %.2f\n", APP_NAME, INPUT_VALUE, result);

    return 0;
}
```

7. Create the Makefile with the smart template version and add the required CFLAGS and LDFLAGS.

```
vi Makefile

PKG_NAME := myproject
PROJECT_TYPE := BINARY
CFLAGS += -DAPP_NAME=\"MathLibTest\"
CFLAGS += -DINPUT_VALUE=25.0
LDFLAGS += -lm
include ../common.mk

#CFLAGS += -DAPP_NAME=\"MathLibTest\"
#Defines the application name macro at compile time.

#CFLAGS += -DINPUT_VALUE=25.0
#Defines the input value used by the application.

#LDFLAGS += -lm
#Links the application with the standard math library, libm, which is required for sqrt().

#Always use += when adding custom CFLAGS or LDFLAGS.
#Do not use := because it will overwrite the default flags required for successful compilation.
```

8. Go back to the SDK root directory

```
cd ../..
```

9. Execute make command and wait till compilation completes

```
make
```

10. Verify that the .ipk package is generated under the images directory

```
ls images/

myproject_1.0_ntc_552.ipk
```

11. Sign the generated .ipk files using the appropriate signing key as required by the platform (refer to Section 10)
12. Install the signed .ipk file on the device (refer to Section 11).
13. Run the application on the device.
    Expected output:

```
MathLibTest: sqrt(25.00) = 5.00
```

If the expected output is displayed, it confirms that:

- APP_NAME and INPUT_VALUE were passed through CFLAGS.
- sqrt() was linked from the math library using LDFLAGS += -lm.

## Method 5: Rewrite custom targets like install, clean in Makefile

### Test Objectives

To understand how to define custom Makefile targets such as build_app, install_app, and install while still using common SDK targets like install_control, install_files, and gen_ipk. This method is useful when the default build or install behavior needs to be customized for a specific project.

### Steps

1. Go to root directory i.e. SDK_Cassini_ntc_552_3.3.1.0R9

```
cd SDK_Cassini_ntc_552_3.3.1.0R9
```

2. Run make clean

```
make clean
```

3. Copy any example project for e.g. helloworld into /projects directory

```
cp -r examples/helloworld projects
```

4. Replace Makefile with custom targets

```
# ---- Required ----
PKG_NAME     := custom-targets
PROJECT_TYPE := BINARY
# ---- Optional Metadata ----
PKG_VERSION  := 1.0
TITLE        := custom-targets example
DESCRIPTION  := Simple custom-targets application for demo
include ../../projects/common.mk
.PHONY: install build_app install_app
install: build_app install_app install_control install_files maybe_install_dev gen_ipk
build_app:
@# Your recipe for building
install_app:
@# Your recipe for installing
EOF
```

***Note:*** *install_files, install_control, maybe_install_dev, and gen_ipk are shipped/common targets coming from: include ../../projects/common.mk. So, you normally do not define them in your project Makefile. Do not add this unless you want to override common behaviour. Because then your local install files will replace/override the one from common.mk.*

5. Give make install command and wait till compilation completes

```
make install
```

6. After system finishes compilation process check the image directory

```
ls images/

custom-targets _1.0_ntc_552.ipk
```

7. Sign the generated .ipk files using the appropriate signing key as required by the platform (refer to Section 10)
8. Install the signed .ipk file on the device (refer to Section 11)

## Method 6: Overwrite library name

### Test Objectives

To build a project as a library instead of a binary and customize the generated library name using LIB_NAME. This method is useful when creating shared library packages and when development files also need to be installed using INSTALL_DEV.

### Steps

1. Go to root directory i.e. SDK_Cassini_ntc_552_3.3.1.0R9

```
cd SDK_Cassini_ntc_552_3.3.1.0R9
```

2. Run make clean

```
make clean
```

3. Copy any example project for e.g. helloworld into /projects directory

```
cp -r examples/helloworld   projects
```

4. Replace library name

```
# ---- Required ----
PKG_NAME := hellolib
PROJECT_TYPE := STATIC_LIBRARY

# ---- Library-Specific ----
INSTALL_DEV := y
LIB_NAME := libcustom.so

include ../../projects/common.mk
EOF
```

5. Give make command and wait till compilation completes

```
make
```

6. After system finishes compilation process check the image directory

```
ls images/

hellolib_1.0_ntc_552.ipk
```

7. Sign the generated .ipk files using the appropriate signing key as required by the platform (refer to Section 10)
8. Install the signed .ipk file on the device (refer to Section 11)

# Method 7: Override Markers

## Test Objectives

To understand how override markers such as CUSTOM_BUILD, CUSTOM_INSTALL, CUSTOM_CLEAN, and CUSTOM_ALL can be used to replace the default rules from common.mk. This method is useful when the project requires complete control over build, install, clean, and test operations.

## Steps

1. Go to root directory i.e. SDK_Cassini_ntc_552_3.3.1.0R9

```
cd SDK_Cassini_ntc_552_3.3.1.0R9
```

2. Run make clean

```
make clean
```

3. Copy any example project for e.g. helloworld into /projects directory

```
cp -r examples/helloworld   projects
```

4. Edit Makefile

```
PKG_NAME     := helloworld
PROJECT_TYPE := BINARY
PKG_VERSION  := 1.0
TITLE        := HelloWorld example
DESCRIPTION  := Simple hello world application for demo
# Override auto-generated rules
CUSTOM_BUILD := y
CUSTOM_INSTALL := y
CUSTOM_CLEAN := y
# or for all above actions define single flag below
# CUSTOM_ALL := y
include ../../projects/common.mk
# this should be the final makefile for helloworld if you are using custom_flags
.PHONY: build install clean

build:
  @echo "CUSTOM_BUILD works"
  $(CC) $(CFLAGS) $(LDFLAGS) -o $(PKG_NAME) $(SOURCES) $(LIBS)

install_local: build
  @mkdir -p $(INSTALLDIR)/usr/bin
  cp $(PKG_NAME) $(INSTALLDIR)/usr/bin/

install: install_local install_control install_files gen_ipk
  @echo "CUSTOM_INSTALL works"
```

5. clean:

   @echo "CUSTOM_CLEAN works"

   rm -f $(PKG_NAME) $(OBJS)Test below commands:

```
make clean
make
```

# 9. Testing Examples

Each example can be tested standalone.

## Testing helloworld

To test helloworld:

```
cd <root folder>
cp -r examples/helloworld projects
make helloworld clean
```

Output:

```
helloworld*.ipk in staging_p are generated or go to images folder.
```

## Testing shared library

To test shared library:

```
cd <root folder>
make shared_library clean
make shared_library
```

Output:

```
libsample*.ipk in staging_p are generated or go to images folder.
```

## Testing plugin

To test plugin:

```
cd <root folder>
make plugin clean
make plugin
```

Output:

```
pxplugin-template*.ipk in staging_p are generated or go to images folder.
```

## Testing all examples

```
cd <root folder>
cp -r examples projects
make clean
make
```

Output:

```
all *.ipk in staging_p are generated or go to images folder.
ipk name will start with the PKG_NAME provided in the makefile of the package.
```

# 10. Signing ipk Packages

After building packages, sign the generated .ipk files using the appropriate signing key as required by the platform.

# 11. Managing Packages on Target Device

You can install packages using the ipktool utility available on the device.

To install packages:

```
scp sampleapp_1.0_ntc_552.ipk root@device:/tmp
ipktool --install /tmp/sampleapp_1.0_ntc_552.ipk 10.3
```

To Install with reboot:

```
ipktool --install /tmp/sampleapp_1.0_ntc_552.ipk –reboot
```

To remove a package:

```
ipktool --remove sampleapp
```

To list installed packages:

```
ipktool --list
```

To perform a Dry Run Test:

```
ipktool --install /tmp/sampleapp_1.0_ntc_552.ipk --test
```