

## Objectives

- Using and customizing Yocto
- Creating Yocto-based Embedded Linux platforms
- Using Yocto to develop components
- Customizing the BSP
- Building out of tree modules
- Setup Source cache

*Labs are conducted QEMU ARM-based board  
We use a recent version of Yocto*

## Prerequisite

- Good C programming skills (see our [oL2 - C Language for Embedded MCU](#) course)
- Knowledge of Linux Embedded systems (see our [oD1 - Embedded Linux](#) course)
- Preferably knowledge of Linux user programming (see our [oD0 - Linux User Mode Programming](#) course)

## Course Environment

- Theoretical course
  - PDF course material (in English).
  - Course dispensed using the Teams video-conferencing system.
  - The trainer answers trainees' questions during the training and provide technical and pedagogical assistance through the Teams video-conferencing system.
- Practical activities
  - Practical activities represent from 40% to 50% of course duration.
  - Code examples, exercises and solutions
  - One Online Linux PC per trainee for the practical activities.
  - The trainer has access to trainees' Online PCs for technical and pedagogical assistance.
  - Eclipse environment and GCC compiler.
  - QEMU Emulated board or physical board connected to the online PC (depending on the course).
  - Some Labs may be completed between sessions and are checked by the trainer on the next session.
- Downloadable preconfigured virtual machine for post-course practical activities
- At the start of each session the trainer will interact with the trainees to ensure the course fits their expectations and correct if needed

## Duration

- Total: 30 hours
- 5 sessions, 6 hours +/-30 min each (excluding break time)
- From 40% to 50% of training time is devoted to practical activities
- Some Labs may be completed between sessions and are checked by the trainer on the next session

## Target Audience

- Any embedded systems engineer or technician with the above prerequisites.

## Evaluation modalities

- The prerequisites indicated above are assessed before the training by the technical supervision of the trainee in his company, or by the trainee himself in the exceptional case of an individual trainee.
- Trainee progress is assessed in two different ways, depending on the course:
  - For courses lending themselves to practical exercises, the results of the exercises are checked by the trainer while, if necessary, helping trainees to carry them out by providing additional details.
  - Quizzes are offered at the end of sections that do not include practical exercises to verify that the trainees have assimilated the points presented
- At the end of the training, each trainee receives a certificate attesting that they have successfully completed the course.
  - In the event of a problem, discovered during the course, due to a lack of prerequisites by the trainee a different or additional training is offered to them, generally to reinforce their prerequisites, in agreement with their company manager if applicable.

## Plan

### Session 1

#### **Introduction to Yocto**

- Overview of Yocto
  - History
  - Yocto, Open Embedded and Poky
  - Purpose of the Yocto project
  - The main projects
- Yocto architecture
  - Overview
  - Recipes and classes
  - Tasks

#### **The Yocto build system**

- Build system objectives
  - Building deployable images
  - Layers and layer priorities
  - Directory layout
  - Configuration files (local, machine and distribution)
  - The bitbake tool
- Using Yocto
  - Building a package
  - Building an image (root file system + u-boot + kernel)
- Miscellaneous tools around Yocto
  - Yocto SDK
  - Extensible SDK

*Exercise: Building a root file system using Yocto*

*Exercise: Use bitbake commands to build package & images*

*Exercise: Building a root file system using Yocto*

*Exercise: Build an extensible SDK for the generated image*

*Exercise: Deploy the generated image*

#### **Yocto package recipes structure**

- Recipe architecture
  - Tasks
  - Task dependencies
  - Recipe dependencies
- The bitbake language
  - Standard variables and functions
  - Classes and recipes
  - The base Yocto classes

- Main bitbake commands
- Adding a new layer
  - Layer structure
  - Various kinds of layers

*Exercise: Adding a new layer*

## Session 2

### **Writing package recipes for Yocto**

- Various kind of recipes and classes
  - Bare program
  - Makefile-based package
  - autotools-based package
  - u-boot
  - kernel
  - Out-of-tree module
- Recipe creation strategies
  - From scratch
  - Using devtool
  - Using recipetool
  - From an existing, similar, recipe
- Debugging recipes
  - Debugging recipe selection
  - Debugging dependencies
  - Debugging tasks
- Defining packaging
  - Package splitting
- Automatically starting a program

*Exercise: Writing a recipe for a local user-maintained package*

*Exercise: Writing and debugging a package recipe for an autotools-based package*

*Exercise: Starting a program at boot (systemd)*

### **Modifying recipes**

- Customizing an existing package recipe (.bbappend)
- Recipe dependencies
- Creating and adding patches
  - Creating a patch for a community-provided component
  - Creating a patch for an user-maintained component
- Defining new tasks
  - Task declaration
  - Coding tasks

*Exercise: Adding patches and dependencies to a community package*

*Exercise: Adding a rootfsinstall task to directly copy the output of a user package in the rootfs image*

## Session 3

### **Creating new kinds of recipes**

- Creating classes
  - Creating new independent classes
  - Inheriting from an existing class

*Exercise: Create a class to generalize the “rootfsinstall” task*

### **Creating a root file system**

- Building a root file system with Yocto

- Creating a custom root file system
- Writing an image recipe
  - Selecting the packages to build
  - Selecting the file system types
  - The various kinds of images
- Inheriting and customizing images
  - Customizing system configuration files (network, mount points, ...)
- Users and groups management
- Package management
  - rpm
  - opkg

*Exercise: Writing and building an image recipe*

*Exercise: Add new users to the image*

*Exercise: Create an image with package support for OTA deployment*

*Exercise: Test OTA update on the generated image*

## Session 4

### **Development process using the extensible SDK and devtool**

- Using devtool to create a package and its recipe
- Using devtool to modify an existing package and recipe
- Using devtool to update a recipe to build a new version of a package

*Exercise: Create, test and modify a recipe for an existing package using devtool*

### **Develop and debug applications using SDK and eclipse**

- Adding eclipse remote debug packages
- Configuring eclipse

*Exercise: Create remote debugging session using eclipse*

### **Writing tasks in python**

- Introduction to python
- Using python in Yocto
  - The main bitbake classes
  - Defining variable values in Python
  - Writing tasks in Python

*Exercise: Writing a task and customizing a recipe in Python*

### **Porting Yocto**

- Porting Yocto to a new board
- BSP architecture
  - Selecting and configuring u-boot recipe
  - Selecting and configuring kernel recipe
- Adding a new BSP layer (yocto-bsp create)

*Exercise: Creating a new BSP layer*

## Session 5

### **BSP Development**

- Adding a custom u-boot to Yocto
- Customizing the Yocto kernel recipe
  - Setting the default configuration
  - Adding patches
  - Specifying the kernel sources

- Configuring Linux Kernel
  - Using menuconfig
  - Using patches
  - Creating Configuration Fragments
  - Validating Configuration
- Kernel device tree

*Exercise: Create u-boot and kernel recipes to use custom versions, test the result*

*Exercise: Patch kernel and activate new options using a fragment*

*Exercise: Create and use a new device tree*

## Out-of-Tree Modules

- Adding modules to image
- Creating an out-of-tree module
- Kernel modules with eSDK

*Exercise: Build and test modules*

## Tailoring the build system

- Setting up a Yocto source cache
  - Local, per system, cache setup
  - Setting up a global, network wide, cache
- Customizing the build system
  - Using a prebuilt toolchain
  - Using a pre-compiled kernel
- Optimizing Yocto build times
  - Using prebuilt, binary, packages
  - Using shared compilation caches

*Exercise: Setting up a global source cache*

*Exercise: Setting up an optimized build environment and rebuilding an image*

## Renseignements pratiques

**Inquiry : 30 hours**