TASKING®

Using the TASKING Software Platform for AURIX

MA160-869 (v1.0rb3) June 19, 2015

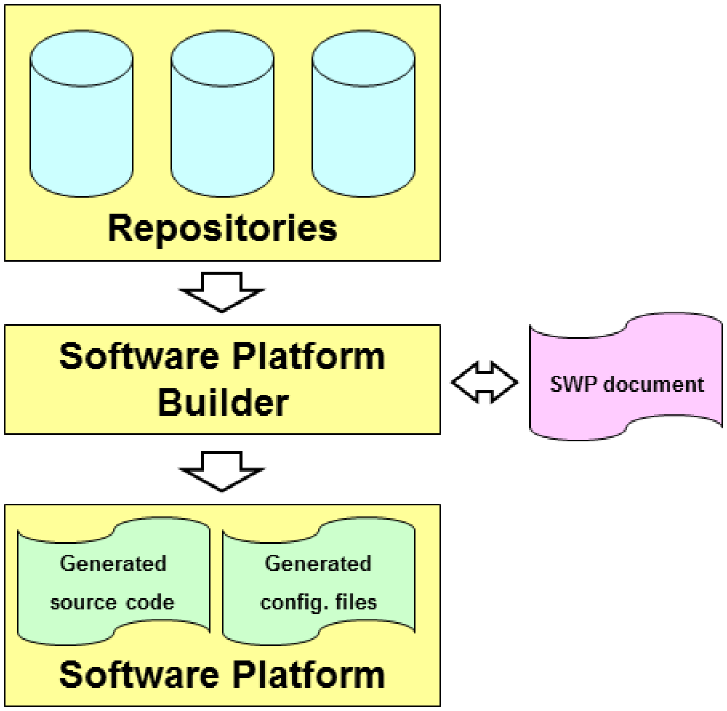
Table of Contents

Chapter 1. Introduction

With TASKING Software Platform you can quickly create full-featured applications. A Software Platform is made up of software blocks, pieces of functionality that you can use in your application, like RTOS facilities, peripheral access or software protocols. The exact contents of a Software Platform depend on the need of your application.

The Software Platform Builder is used to manage your Software Platform. It is both a graphical editor and a code generator. Collections of software modules are delivered as Software Platform repositories.

The following figure shows the process to create your own Software Platform:



A Software Platform repository may contain any kind of software, but typical modules include interrupt services, timers, peripherals (hardware wrappers), drivers, kernel services (such as POSIX multithreading), device I/O, file system (FatFs), networking (TCP/IP), graphical user interface, etc.

The main reasons for having software modules are:

- Integration. Modules do not live isolated. Quite the contrary, they need to relate and interact with other modules. For instance, they often define interfaces which are implemented by other modules.
- Software reuse. Software reuse means effectively to be capable of encapsulating all the information of a component (hardware and/or software) in a consistent manner. Once this information is made

available, higher abstraction layers can use it to customize or configure the system after little (or none) user intervention.

- Hide complexity. Writing software from scratch is a time-consuming activity. To write software that accesses or controls a peripheral, you need a thorough knowledge of how the peripheral works: which registers you need, which device specific commands to use, which communication protocols to use, and which interrupts to handle.

  The modules of the Software Platform repository take care of all these laborious lower level routines and instead provide you with an easy to use Application Programming Interface (API) for each peripheral that your application needs to control.

- Configurability. Embedded systems require very precise (compile-time) configuration mechanisms to manage their intrinsic complexity. A typical module may show different possible configurations that have direct impact on the final behavior of the software. The main idea behind compile-time configurations in embedded systems is to remove unnecessary functionality in order to enhance memory costs and real-time performance.

- Expandability. Altium, but also other parties, can adapt the modules and create their own modules.

For information how to use the Software Platform Builder, see Chapter 3, *Getting Started with the Software Platform Builder*.

Chapter 2. Organization of the Software Platform Repositories

Software Platform repositories can contain numerous software modules that take care of lower level software routines as well as modules that offer extra functionality by providing you with a convenient API.

The Software Platform consists of device stacks and software services. This chapter describes both parts and how they are related.

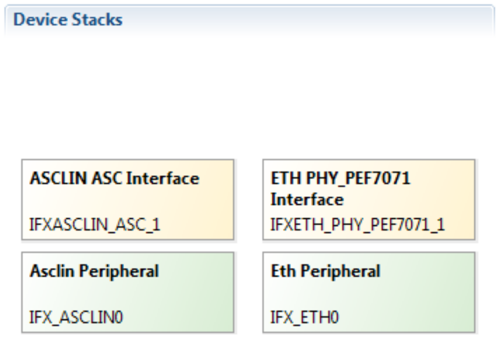
2.1. Device Stacks

Device stacks are all about making hardware peripherals available to application code through abstract and generic software interfaces. By placing more or less modules on a stack, you can choose the abstraction level you want to use in your application. The lowest level modules are specific for a particular hardware device. On top of that, you can stack higher level modules that provide more generic functionality to access the device. For example, at the higher, abstract level, you could choose to use a module to access a file system in your application. At the lower levels you still can select modules to decide which specific storage device you want to access (a hard drive, SD card, RAM drive, ...) Thus, the lower level modules are more specific for a particular peripheral while the higher level modules are less hardware specific and can even be used in combination with multiple peripheral devices.

You can use the Software Platform Builder editor to build device stacks.

Example

The following figure contains an ASCLIN ASC device stack and an ETH device stack.

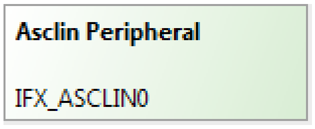


Each colored stack item represents a software module. In this example the ASCLIN stack consists of two modules, for each abstraction level one. In general, your application interfaces only to the highest level modules.

Device stacks may be composed of the following types of plug-ins: peripherals and drivers.

2.1.1. Peripherals

Peripherals (the green stack items) are the lowest level modules. They provide information for the higher layers of the stack to access the peripherals. Information such as the base address, interrupt assignment and any soft peripheral configuration is all stored in the peripheral.


Asclin Peripheral

IFX_ASCLIN0

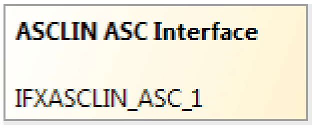
Peripheral modules are meant to be instantiated. Each instance then corresponds to one device of the given kind. Instances of peripherals do not normally require other instances.

In most situations, your application does not access the peripherals directly, because the application accesses them through the driver's interface on top of it.

For most peripherals you need to use the TASKING Pin Mapper to configure the connections between peripherals and port pins. For information about the TASKING Pin Mapper see *Using the TASKING Pin Mapper for AURIX*.

2.1.2. Drivers

Drivers (the yellow stack items) provide the next level of abstraction. They provide low-level access to specific hardware either via a peripheral or via another driver.


ASCLIN ASC Interface

IFXASCLIN_ASC_1

The difference between a driver and a peripheral is well defined. The peripheral only defines basic information about the hardware but provides no further functionality. This basic information can be used by a driver which you can place on top of the peripheral.

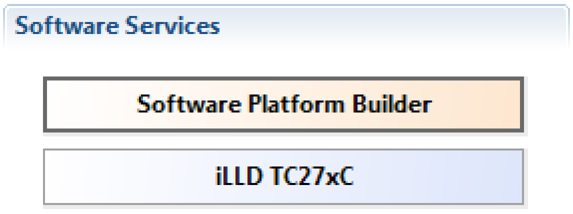
Driver modules are meant to be instantiated. Each instance then corresponds to one device of the given kind. A driver instance requires one instance of the corresponding peripheral.

Drivers are hardware specific and so are the interfaces they offer. Hence, if your application contains code that accesses a driver's API, your application will be hardware dependent. Drivers still operate at a low abstraction level, and using a device at the driver level requires knowledge of that particular driver's interface.

The driver provides the initialization for the peripheral and pins. See Section 3.3, *Configuring the Software Platform Plug-ins* how you can set some initialization options. To interface with the peripheral use the low-level driver routines from the iLLD API.

2.2. Software Services

Some services are static and not meant to be instantiated. They have no (direct) relationship with peripherals or other services. They facilitate common functionality, like iLLD TC27xC support. They are shown in your editor at the left side of the **Device Stacks**. Those services may also be added automatically if required by other services.



When you click on the iLLD software service you can set some options. After you have generated the Software Platform sources (Section 3.4, *Generating and Using the Source Code*) you can use the low-level driver routines from the iLLD API.

Chapter 3. Getting Started with the Software Platform Builder

The Software Platform Builder is the graphical interface to manage the plug-ins of the Software Platform repositories. The Software Platform Builder enables you to:

- add peripherals
- build device stacks
- add software services
- configure all software services and stack items
- generate code for your Software Platform

3.1. Creating a Software Platform Document

To use the Software Platform Builder you need a project. The Software Platform Builder becomes available when you add a special Software Platform document to your project.

Add a Software Platform document to an existing project

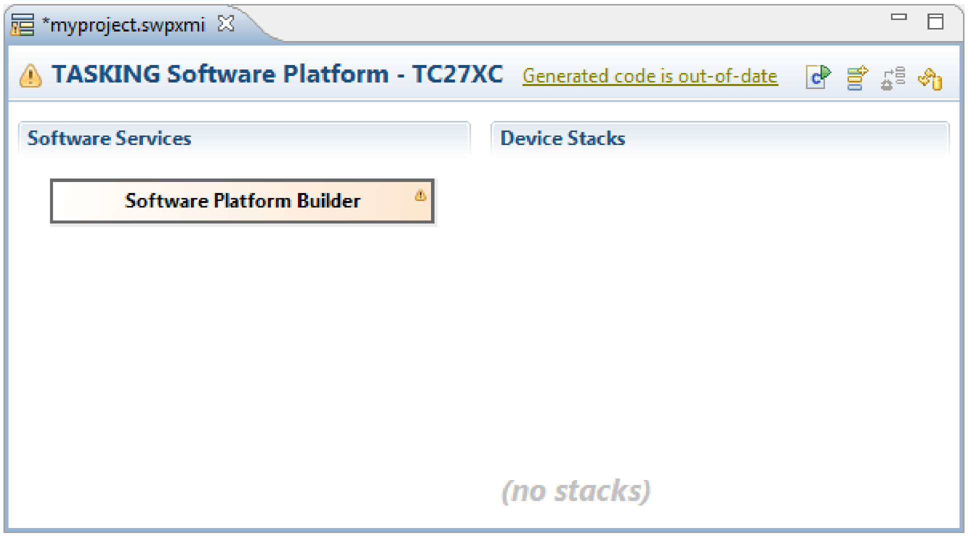
1. First make sure you have an existing project. This is explained in the *Getting Started* manual of the toolset. In this example we assume you have a project called `myproject`.
2. From the **File** menu, select **New » TASKING Software Platform Document**.

   The New Software Platform Document wizard appears.
3. Select the **Project** folder for the Software Platform document: type the name of your project (`myproject`) or click the **Browse** button to select a project.
4. In the **File name** field, enter a name for the Software Platform document, for example `myproject.swpxmi` and click **Finish**.

   A new Software Platform document is added to the project with extension `.swpxmi`.

TASKING Software Platform Builder editor

Double-click on the Software Platform document to open it. The editor consists of two sections: the **Software Services** and the **Device Stacks**.


*myproject.swpxmi
TASKING Software Platform - TC27XC
Generated code is out-of-date
Software Services
Device Stacks
Software Platform Builder
(no stacks)

The following toolbar icons are available:

Icon	Action	Description
	Generate Code	Generates the source code and adds it to your project.
	Add	Adds software services and/or device stacks.
	Synchronize with Pin Mapper output	Imports Pin Mapper settings. If you already have configured peripherals in the Pin Mapper, the settings are synchronized with the Software Platform.
	Reload Repository	Reloads the repository. This can be necessary when you have updated the repository or when you changed the contents of a local repository.

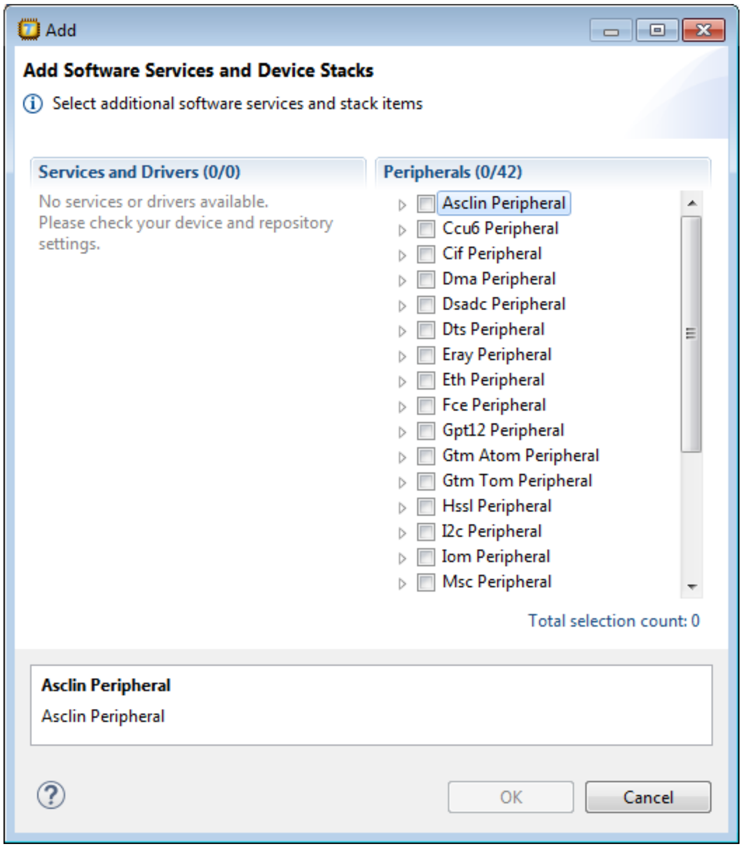
3.2. Working with Device Stacks

In the **Device Stacks** section you can build device stacks by adding modules from one or more Software Platform repositories. To build a device stack, you can start bottom-up by choosing a peripheral or you can start top-down by choosing a high-level stack service.

Add a software service or device stack

1. From the **Software Platform** menu, select **Add...** or click .

   The Add Software Services and Device Stacks dialog appears. The left side shows the stack services, software services and drivers. For AURIX no services or drivers are available. The right side shows all peripherals for the selected device.



Add Software Services and Device Stacks

Select additional software services and stack items

Services and Drivers (0/0)

No services or drivers available.
Please check your device and repository settings.

Peripherals (0/42)

- Asclin Peripheral
- Ccu6 Peripheral
- Cif Peripheral
- Dma Peripheral
- Dsadc Peripheral
- Dts Peripheral
- Eray Peripheral
- Eth Peripheral
- Fce Peripheral
- Gpt12 Peripheral
- Gtm Atom Peripheral
- Gtm Tom Peripheral
- Hssl Peripheral
- I2c Peripheral
- Iom Peripheral
- Msc Peripheral

Total selection count: 0

Asclin Peripheral

Asclin Peripheral

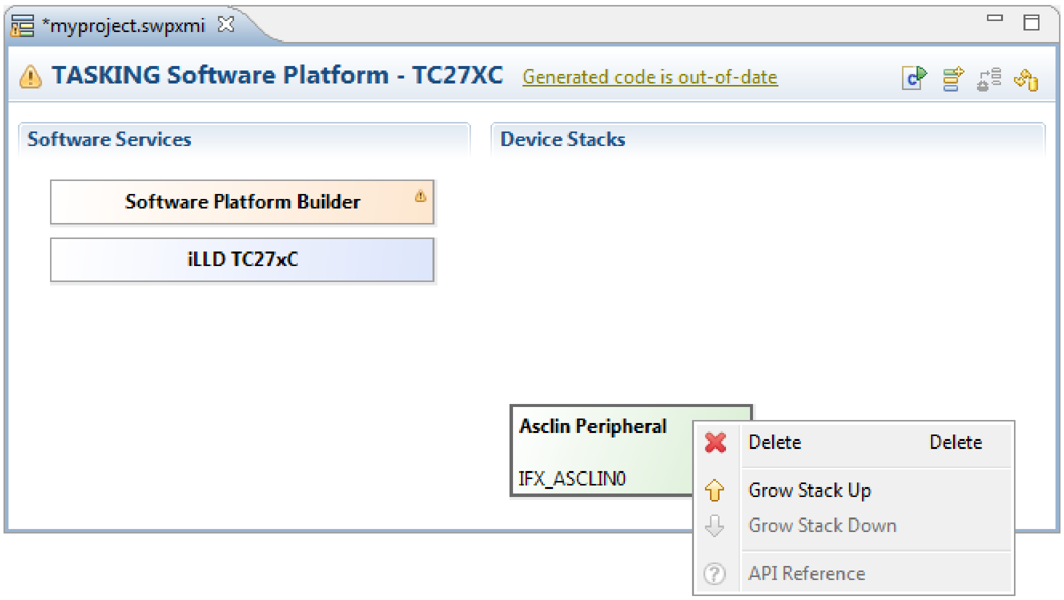
2. Select the peripherals you want and click **OK**.

   The new device stacks and software services appear in the TASKING Software Platform editor. Some software services appear automatically because they are required by other services or stack items.

If you do not see any peripheral, you probably selected an unsupported device or a generic architecture.

Growing stacks

Once you have added one or more stack items, you can use **Grow Stack Up** or **Grow Stack Down** from the context menu to complete your device stack.

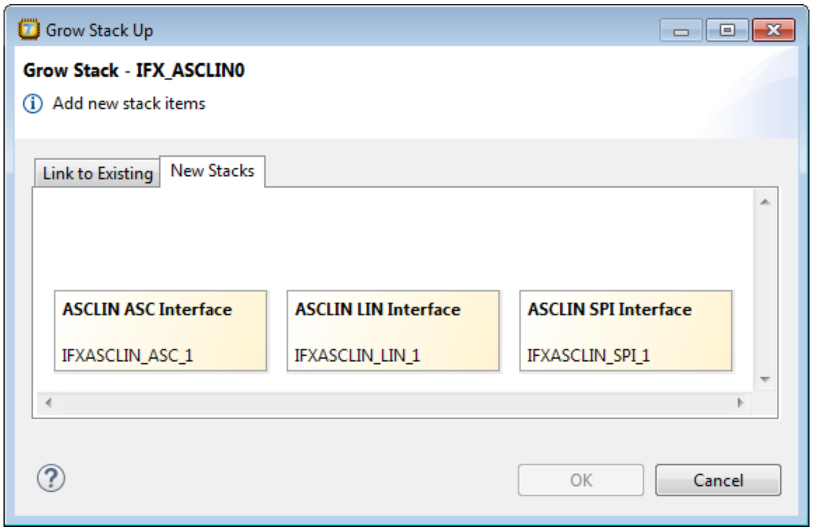


The **Grow Stack Up** and **Grow Stack Down** menu items only become available if possibilities exist to extend the stack from the point of the selected stack item. Once you have partial device stacks on the document, you can extend them with other stack items to grow them to the abstraction level you wish to use in your application. For an interface description of a module select **API Reference**.

To grow a stack up:

1. Right-click on a plug-in of the existing stack and select **Grow Stack Up**.

   The Grow Stack dialog appears. The dialog shows all possible combinations of sub-stacks that you can place on top of the selected module.



Grow Stack Up

Grow Stack - IFX_ASCLIN0

Add new stack items

ASCLIN ASC Interface	ASCLIN LIN Interface	ASCLIN SPI Interface
IFXASCLIN_ASC_1	IFXASCLIN_LIN_1	IFXASCLIN_SPI_1

2. Select any module in any of the possible sub-stacks to extend the selected stack upwards and click **OK**.

To grow a stack down:

1. Right-click on a plug-in of the existing stack and select **Grow Stack Down**.

   The Grow Stack dialog appears. The dialog shows all possible combinations of sub-stacks that you can place below the selected module.

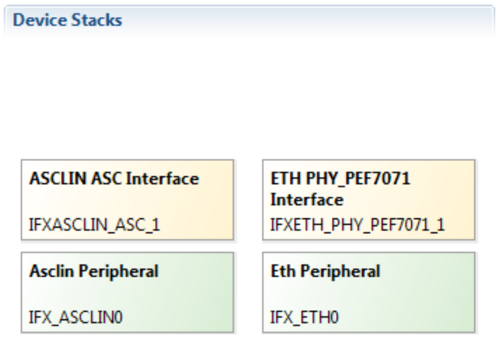
2. Select any module in any of the possible sub-stacks to extend the selected stack downwards and click **OK**.

Note that when you click on a **Missing Requirement**, the Grow Stack dialog appears automatically.

The Grow Stack dialog contains two tabs, one to form a new stack and one to link two existing stacks. If your document contains partial stacks which potentially can be combined into a single stack, it will be visible in the **Link to Existing** tab. You can link them together to form a complete stack. Otherwise select the **New Stacks** tab.

Example

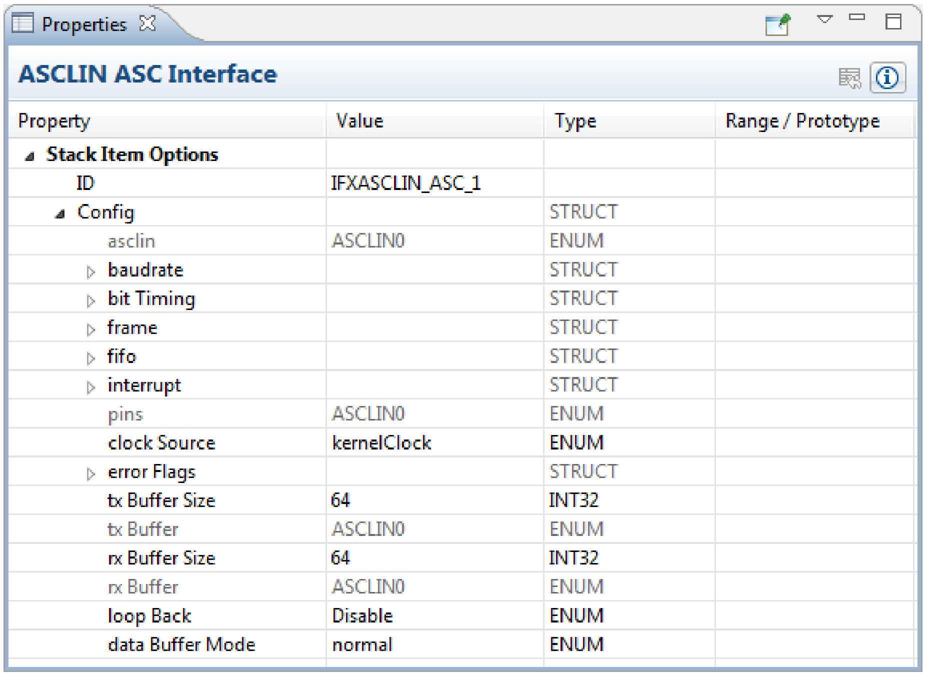
The following figure shows some device stacks.



3.3. Configuring the Software Platform Plug-ins

You can configure peripherals, drivers, stack services and software services. Stack items can have individual options and/or global options. Individual options are only valid for a specific stack item, global options are valid for all stack items of the same type. When you click on an item in the Software Platform Builder, configuration options appear in the Properties view.

For example, when you click **ASCLIN ASC Interface**, a view similar to the following appears:



ASCLIN ASC Interface

Property	Value	Type	Range / Prototype
Stack Item Options			
ID	IFXASCLIN_ASC_1		
Config		STRUCT	
asclin	ASCLIN0	ENUM	
baudrate		STRUCT	
bit Timing		STRUCT	
frame		STRUCT	
fifo		STRUCT	
interrupt		STRUCT	
pins	ASCLIN0	ENUM	
clock Source	kernelClock	ENUM	
error Flags		STRUCT	
tx Buffer Size	64	INT32	
tx Buffer	ASCLIN0	ENUM	
rx Buffer Size	64	INT32	
rx Buffer	ASCLIN0	ENUM	
loop Back	Disable	ENUM	
data Buffer Mode	normal	ENUM	

Here you can set the initialization options that are needed for your application.

A plug-in option may automatically set other plug-in options.

Connecting peripherals to pins

For most peripherals you need to specify which pins are used. You can do this with the TASKING Pin Mapper.

1. Add a TASKING Pin Mapper Document (**File » New » TASKING Pin Mapper Document**) and configure the pins as explained in *Using the TASKING Pin Mapper for AURIX*.

   This results in a file called `myproject.pincfg`.

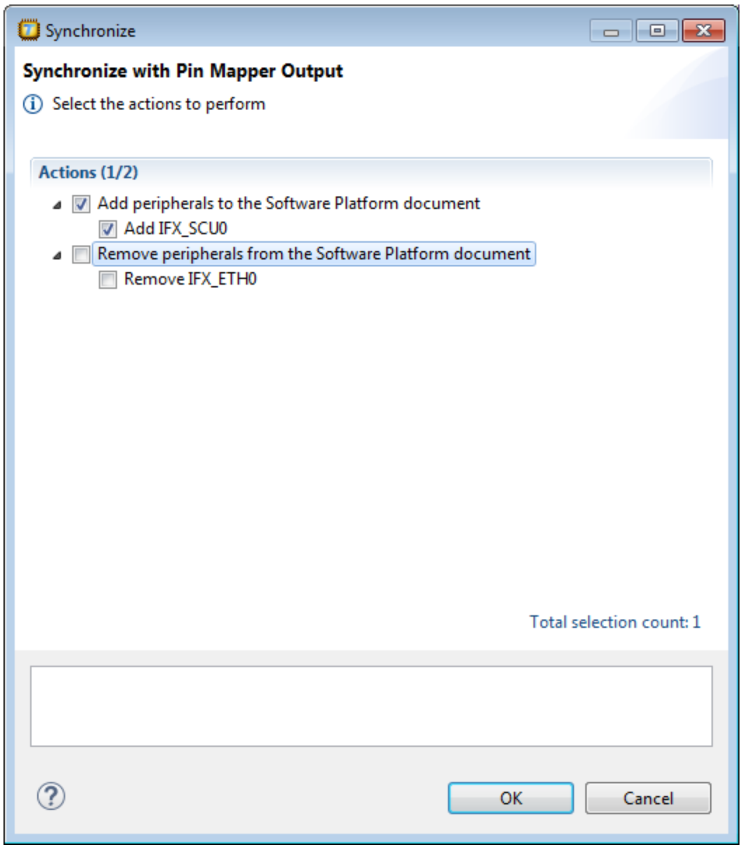
2. From the **Pin Mapper** menu, select **Generate Code** or click .

   The Pin Mapper sources are generated and are added to your project in the folder `PinMapper`.

3. Go back to the TASKING Software Platform document.

4. From the **Software Platform** menu, select **Synchronize with Pin Mapper Output** or click .

   The Synchronize dialog appears.



Synchronize

Synchronize with Pin Mapper Output

Select the actions to perform

Actions (1/2)

- [✓] Add peripherals to the Software Platform document
  - [✓] Add IFX_SCU0
- [ ] Remove peripherals from the Software Platform document
  - [ ] Remove IFX_ETH0

Total selection count: 1

5. Select the peripherals you want to add to or remove from the Software Platform document and click **OK**.

The Software Platform uses the output of the TASKING Pin Mapper to configure the peripherals. The pin configurations are used in the initialization routines. See Section 3.4, *Generating and Using the Source Code* for more information.

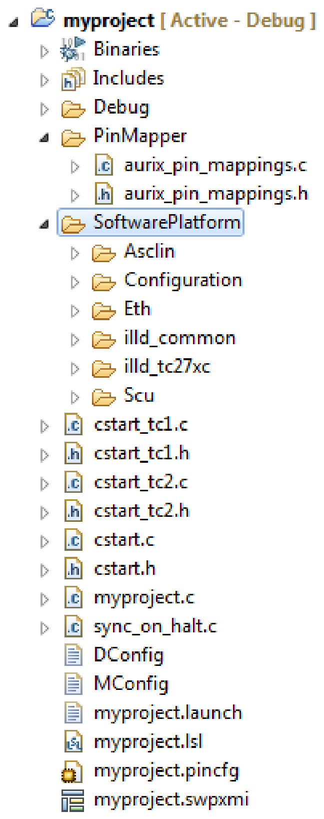
3.4. Generating and Using the Source Code

When you have added, configured and/or updated the Software Platform plug-ins you are ready to add the Software Platform sources to your project.

Generate source code

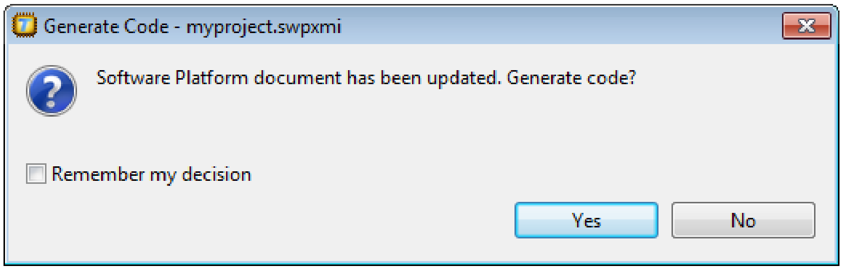
- From the **Software Platform** menu, select **Generate Code** or click .

  The Software Platform sources are retrieved from the repositories and are added to your project in the folder `SoftwarePlatform`. Tool options such as defines and include paths are added to the Generated options field in your project properties. Options can also be set on the `SoftwarePlatform` folder and/or sub-folders.



```
myproject [ Active - Debug ]
  Binaries
  Includes
  Debug
  PinMapper
    aurix_pin_mappings.c
    aurix_pin_mappings.h
  SoftwarePlatform
    Asclin
    Configuration
    Eth
    illd_common
    illd_tc27xc
    Scu
  cstart_tc1.c
  cstart_tc1.h
  cstart_tc2.c
  cstart_tc2.h
  cstart.c
  cstart.h
  myproject.c
  sync_on_halt.c
  DConfig
  MConfig
  myproject.launch
  myproject.lsl
  myproject.pincfg
  myproject.swpxmi
```

Also, every time you save the changes you have made to your Software Platform document, a dialog appears asking if you want to generate the code. You can change this behavior in the Preferences dialog as explained in Section 3.7, *Software Platform Builder Preferences*.



Software Platform document has been updated. Generate code?

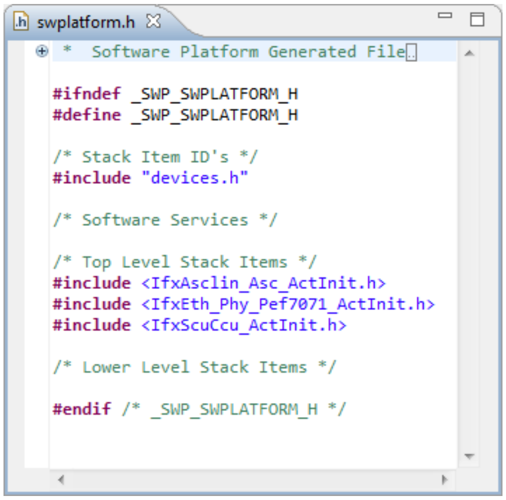
Using the Software Platform sources

A plug-in defines a set of functionality - that is, types, defines, structures and functions. This functionality is accessible from your application. You can find the generated files in the `SoftwarePlatform` folder of your project. To use the generated files, you must add a `#include` statement into your top-level source file (typically `main.c`):

```
#include "swplatform.h"
```

All other required header files needed to access the devices in the stack are included (made available) as a result of including this file.

The following is an example of `swplatform.h` generated as a 'wrapper' around the active project's Software Platform:



```
 *  Software Platform Generated File

#ifndef _SWP_SWPLATFORM_H
#define _SWP_SWPLATFORM_H

/* Stack Item ID's */
#include "devices.h"

/* Software Services */

/* Top Level Stack Items */
#include <IfxAsclin_Asc_ActInit.h>
#include <IfxEth_Phy_Pef7071_ActInit.h>
#include <IfxScuCcu_ActInit.h>

/* Lower Level Stack Items */

#endif /* _SWP_SWPLATFORM_H */
```

If you have set pin configurations in a Pin Mapper document, those settings are used in the initialization routines. For example, the file `IfxAsclin_Asc_ActInit.h` contains, amongst others, the line:

```
#include <PinMapper/aurix_pin_mappings.h>
```

You can now use the functions of the Software Platform in your sources. For example,

```
IfxAsclin_Asc *asclin;

int main( void )
{

    /* Initialize the Asclin module and initialize
       the pins as specified in the Pin Mapper document */

    asclin = IfxAsclin_Asc_ActInit(IFXASCLIN_ASC_1);
    ...
    return 0;
}
```

3.5. Using the Sample Projects

A Software Platform repository may contain a number of sample projects. These examples can be a good starting point for your own project.

You can import Software Platform repository examples via the Import wizard in Eclipse.

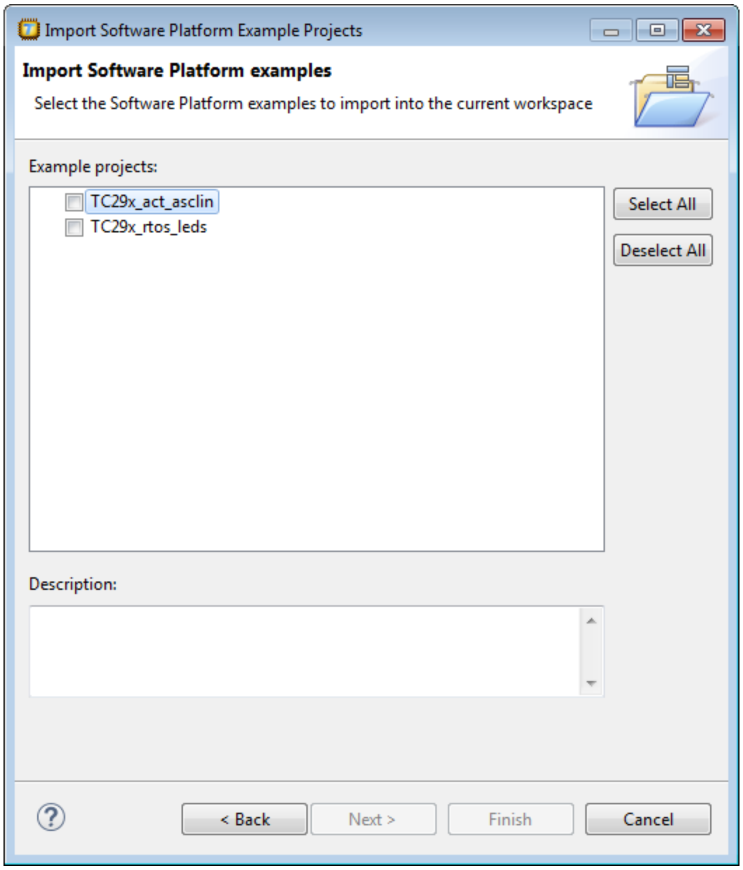
Import an existing Software Platform project

1. From the **File** menu, select **Import**.

   The Import dialog appears.

2. Select **TASKING Software Platform » Example Projects** and click **Next**.

   The Import Software Platform Examples dialog appears.



Import Software Platform Example Projects

Import Software Platform examples

Select the Software Platform examples to import into the current workspace

Example projects:

- [ ] TC29x_act_asclin
- [ ] TC29x_rtos_leds

Description:

3. Select the Software Platform examples you want to import into the current workspace and click **Finish**.

   The original examples are copied into the current workspace and the Software Platform sources and Pin Mapper sources are generated.

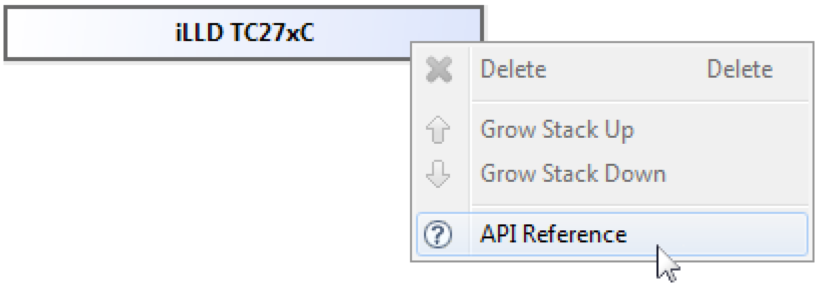
Once the examples are in your workspace you can build the projects.

3.6. API Reference

Each repository contains detailed descriptions of its contents. You can access this documentation from the Help menu (**Help » Help Contents » Software Platform Repository Reference**).

Accessing help on individual drivers or services

- Right-click on a stack item or a software service and select **API Reference**.



Help appears on the individual stack item.

3.7. Software Platform Builder Preferences

You can use the Preferences dialog in Eclipse to specify how the Software Platform Builder should operate.

To set preferences

1. From the **Window** menu, select **Preferences**.

   The Preferences dialog appears.

2. Select **TASKING » Software Platform Builder**.

   The Software Platform Builder page appears.

3. Set your preferences and click **OK**.

You can set the following preferences:

Generate code on save

By default the Software Platform Builders asks if you want to generate code when you save a document (**Prompt**). You can choose to do this automatically (**Always**) or **Never**.

Software Platform Editor

- **Show adaptors and internal items**. By default, internal items are not visible in the device stacks. This option can be useful when you need support from Altium or when you develop your own plug-ins. Normally, you do not need this option.

Properties View

- **Show resource info**. By default, information about the resource is not visible in the Properties view. You can use the **Show Resource Info** button (ⓘ) in the Properties view to toggle the resource information on or off, or you can use this preference option to turn resource information on by default.

- **Show hidden options**. By default, several plug-in options are not visible in the Properties view. This option can be useful when you need support from Altium or when you develop your own plug-ins. Normally, you do not need this option.

Chapter 4. Glossary

The following terminology is used to describe the TASKING Software Platform concepts (in alphabetical order).

Software Platform

The source code generated by the Software Platform Builder. The source code becomes part of your project. A graphical representation of the Software Platform is present in the Software Platform Builder.

Software Platform Builder

The graphical editor where you can edit a Software Platform document and where you can generate the source code for your Software Platform. The editor shows a graphical representation of your Software Platform.

Software Platform document

A file with extension `.swpxmi` which is part of your project. It contains the configuration of your Software Platform and is managed by the Software Platform Builder.

Software Platform plug-in or module

Software Platform definition files, source code and other resources. Plug-ins are bundled in a Software Platform repository.

Software Platform repository

The collection of Software Platform plug-ins. It provides the contents for a generated Software Platform.

TASKING Software Platform

The main software framework delivered by Altium, consisting of Software Platform repositories and the Software Platform Builder.