Embedding it better...

µTasker Document

- **CodeWarrior 10**

Table of Contents

1. Introduction

CodeWarrior 10 is the new IDE from Freescale based on Eclipse which replaces the earlier CodeWarrior IDE from Metroworks, which Motorola acquired in 1999, and further developed as Freescale's (semiconductor spinoff from Motorola in 2003) Developer Technology Organisation.

The Eclipse platform has become popular in recent years as open-source software development environment and adopted by many companies, including a number of semiconductor manufacturers for their development tools; *one notable exception however being ATMEL Corp. who moved away from their AVR32 Studio Eclipse based platform to their present AVR Studio 6 based on Microsoft Visual Studio (interestingly, the main rival to Eclipse when IBM first started working on the Eclipse software, which was then made open-source).*

Eclipse is Java based and supports many plug-ins. This means that it often runs quite slowly and its potentially huge range of capabilities may make it overwhelming when users just want to do some 'simple' embedded development. This is also something that Freescale is aware of and new CodeWarrior development improves the user experience by streamline its look and feel and accelerate such things that could be otherwise unnecessarily cumbersome[1].

Eclipse, and thus CodeWarrior 10, works with workspaces. Workspaces are a concept that can cause difficulties in embedded projects base on the concepts that traditional IDEs use. For example, it becomes difficult to share common parts of projects (such as libraries) because the IDE wants to import the source files into its own unique project directory. However this has been addressed in newer Eclipse releases and so the impacts are not as serious as they were in earlier versions.

Each company basing tools on Eclipse make specific adaptations which make their solution either worse of better than competitors. This document specifically looks at the CodeWarrior 10 solution with regards to using it together with the µTasker project. It discusses how to create a new application in the Kientis project environment (Coldfire projects are equivalent) and gives tips and tricks to work around remaining caveats.

[1] Freescale's Jim Trudeau discusses the CodeWarrior development in the following blog:

https://community.freescale.com/community/the-embedded-beat/blog/2012/09/25/codewarrior-ide-v103-a-whole-new-perspective

While the CodeWarrior 10.3 Beta is discussed there the version CodeWarrior 10.2 (the official release version at the time of writing) is used here.

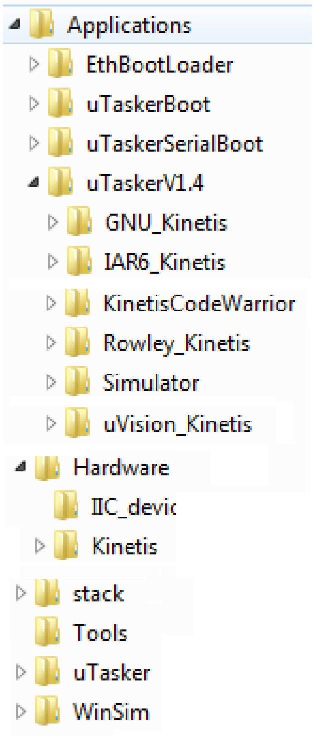
For a simple step-by-step guide to building the µTasker project in CodeWarrior see http://www.utasker.com/docs/KINETIS/uTaskerV1.4_Kinetis_demo.pdf

2. Importing the Project into a Workspace

When CodeWarrior starts it will ask for a workspace, unless the user has disabled the question, in which case it will work with the last workspace used.

The first time a new user sees this it may cause some panic because the first question is what is this workspace? Many IDEs use the same term to save a projects environment (where files are located, the compiler and debugger settings and user's editor preferences etc.) but one generally saves this as a project file or maybe as a workspace and doesn't need to specify it before actually doing anything. Many traditional IDEs' workspace is in fact simply a file that the IDE uses when starting to restore the user's or project's settings, but CodeWarrior's workspace is not a file but a location on the PC where it will be working and, optionally, keeping all of the project's files.

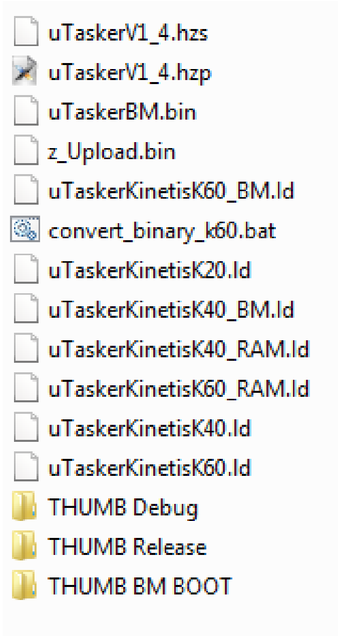
The µTasker project includes various IDE projects and has the following structure:



The "`Hardware`" directory contains the main hardware specific interface (allowing the more general project parts to operate specifically on the Kinetis), "`uTasker`" contains the operating system part, generic drivers, etc. "`stack`" the TCP/IP stack, "`WinSim`" the project's general simulation support and "Applications" a number of individual projects (such as `uTaskerBoot` and `uTaskerSerialBoot`).

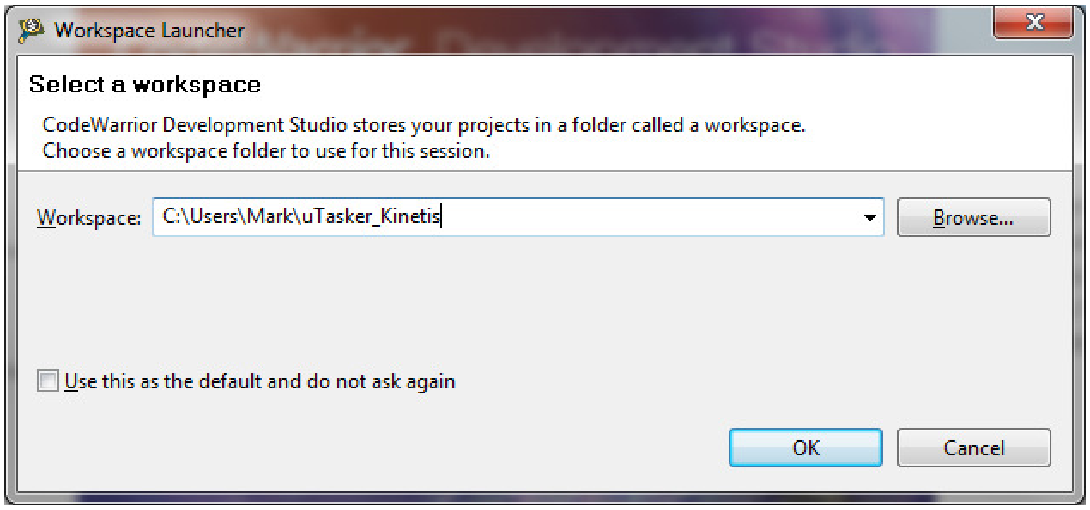
The "`uTaskerV1.4`" project is designed to be used as a basis for new projects by copying it to a new directory (with the project's) name and then adapting this to suit the specific requirements.

Each project then contains project setups for a number of popular IDEs and the "`KinetisCodeWarrior`" directory is associated with CodeWarrior10 use, although not in the normal sense, as will be seen later. As a 'normal' example one could take the "`Rowley_Kinetis`" project folder and see that it contains the following files and sub-directories:



This is very much a traditional project folder that most embedded developers will be very familiar with.

It contains the IDEs project file "`uTaskerV1.4.hzp`" which can be double clicked to open the IDE itself and load the specific project. It also contains some sub-directories containing object files that were created when the project was built in Debug or Release modes. Conveniently the resulting binary files and the used linker script files are also here so that the can be simply accessed when needed.



Workspace Launcher

Select a workspace

CodeWarrior Development Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

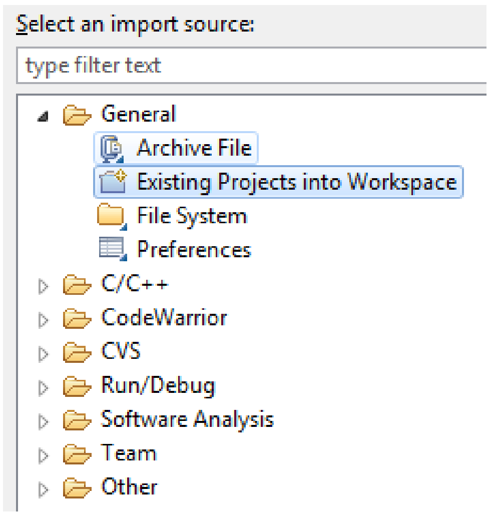
Workspace: C:\Users\Mark\uTasker_Kinetis

This shows the selection of the workspace, which can be located anywhere on the PC (not directly related to the location of the project files) and it is worthwhile keeping a track of where different projects' work spaces are located in case several independent projects are worked on.

When CodeWarrior 10 has started it will display a welcome window (this window can be returned to later by selecting "`Welcome`" in the "`Help`" menu.

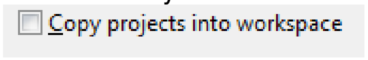
Choose "`Go to Workbench`" to go to the C/C++ perspective.

Normally the "`CodeWarrior Projects`" view is displayed on the left hand side of the perspective and this is where projects can be imported to by using the mouse right click to display its context menu and choosing "import".



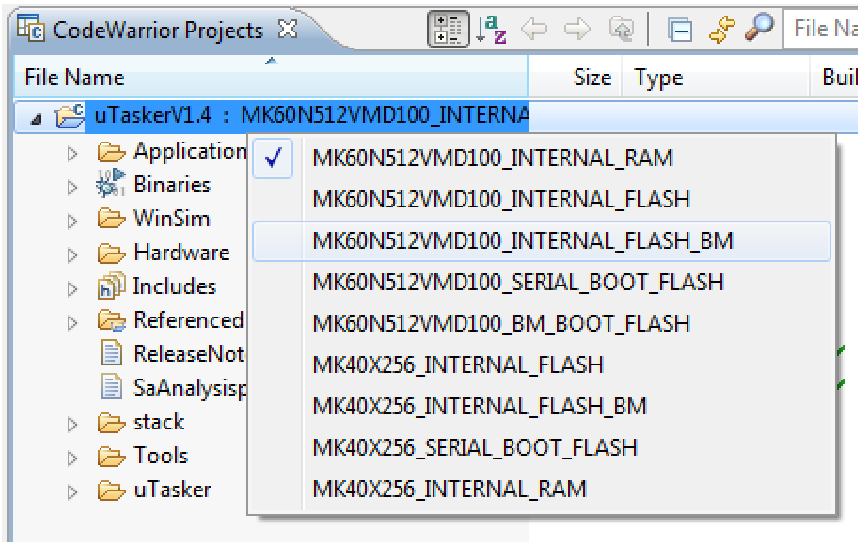
When the Import dialog appears select "`General | Existing Projects into Workspace`" and then browser to the location of the µTasker project on the PC. Select the project's main director (the one containing uTasker, Applications, etc.). This will then show up with its path name and a check box to show that it was recognised as a valid project location.

Finally click on "`Finish`" to perform the import, whereby it is to be noted that there is a check box which is off by default:



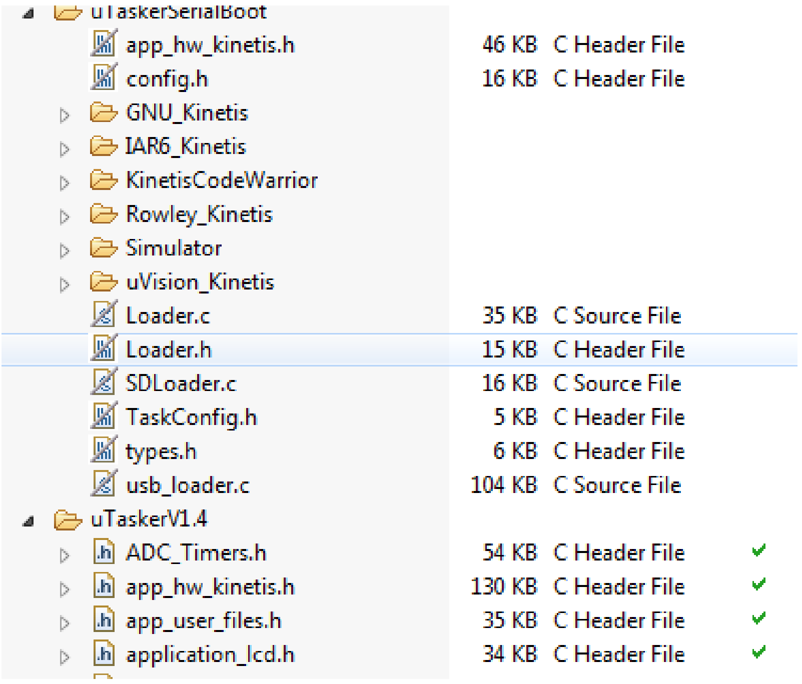
If this were to be checked the import woudl involve a copy of the content of the directory into the workspace which probably doesn't have any advantages in general use. When the check box is left in its default unchecked state the source files will be accessed by the CodeWarrior 10 project at their original location.

Once the import has completed the CodeWarrior Projects view will show the uTaskerV1.4 project and it is possible to open a drop down list of its targets as show in the following screen shot:


CodeWarrior Projects
File Name
Size
Type
Buil
uTaskerV1.4 : MK60N512VMD100_INTERNA
Application
Binaries
WinSim
Hardware
Includes
Referenced
ReleaseNot
SaAnalysisp
stack
Tools
uTasker
MK60N512VMD100_INTERNAL_RAM
MK60N512VMD100_INTERNAL_FLASH
MK60N512VMD100_INTERNAL_FLASH_BM
MK60N512VMD100_SERIAL_BOOT_FLASH
MK60N512VMD100_BM_BOOT_FLASH
MK40X256_INTERNAL_FLASH
MK40X256_INTERNAL_FLASH_BM
MK40X256_SERIAL_BOOT_FLASH
MK40X256_INTERNAL_RAM

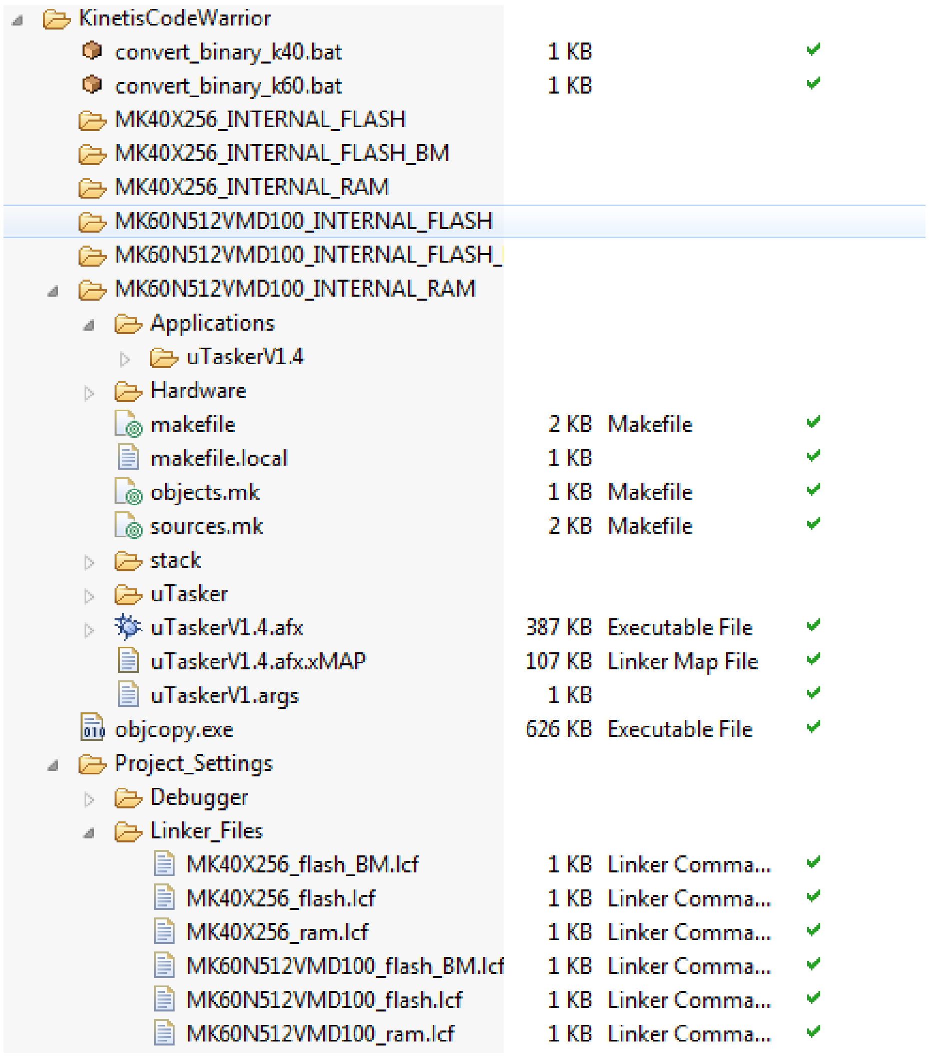
This shows the first difference compared to traditional projects. Rather than having one project for each application (`uTaskerV1.4`, `uTaskerBootLoad`er, etc.) there is only one single project (uTaskerV1.4) which includes the different applications as targets to it (note that, in this example, the target `MK60N512VMD100_BM_BOOT_FLASH` is used to build the uTaskerBootLoader project for a K60 and `MK60N512VMD100_INTERNAL_FLASH` is used to build the uTaskerV1.4 project for the K60). *The target to be used can be set as active target in the drop-down list.*

When the “`Applications`” directories are expanded it will be seen that some files are marked with a green tick and some are displayed as crossed-out:


uTaskerSerialBoot
app_hw_kinetis.h 46 KB C Header File
config.h 16 KB C Header File
GNU_Kinetis
IAR6_Kinetis
KinetisCodeWarrior
Rowley_Kinetis
Simulator
uVision_Kinetis
Loader.c 35 KB C Source File
Loader.h 15 KB C Header File
SDLoader.c 16 KB C Source File
TaskConfig.h 5 KB C Header File
types.h 6 KB C Header File
usb_loader.c 104 KB C Source File
uTaskerV1.4
ADC_Timers.h 54 KB C Header File
app_hw_kinetis.h 130 KB C Header File
app_user_files.h 35 KB C Header File
application_lcd.h 34 KB C Header File

The target selected controls which of the files and folders in the (referenced) workspace actually belong to the build and which don’t.

Looking down inside the KinetisCodeWarrior sub-directory shows that it may not be used to hold the individual projects themselves but is still being used to keep the linker script files and also the generated object files and make files (which CodeWarrior automatically generates based on the target's files) plus the resulting output elf file `uTaskerV1.4.afx`.



```
KinetisCodeWarrior
  convert_binary_k40.bat                1 KB                    ✓
  convert_binary_k60.bat                1 KB                    ✓
  MK40X256_INTERNAL_FLASH
  MK40X256_INTERNAL_FLASH_BM
  MK40X256_INTERNAL_RAM
  MK60N512VMD100_INTERNAL_FLASH
  MK60N512VMD100_INTERNAL_FLASH_
  MK60N512VMD100_INTERNAL_RAM
    Applications
      uTaskerV1.4
    Hardware
    makefile                            2 KB  Makefile          ✓
    makefile.local                      1 KB                    ✓
    objects.mk                          1 KB  Makefile          ✓
    sources.mk                          2 KB  Makefile          ✓
    stack
    uTasker
    uTaskerV1.4.afx                   387 KB  Executable File   ✓
    uTaskerV1.4.afx.xMAP              107 KB  Linker Map File   ✓
    uTaskerV1.args                      1 KB                    ✓
  objcopy.exe                         626 KB  Executable File   ✓
  Project_Settings
    Debugger
    Linker_Files
      MK40X256_flash_BM.lcf             1 KB  Linker Comma...   ✓
      MK40X256_flash.lcf                1 KB  Linker Comma...   ✓
      MK40X256_ram.lcf                  1 KB  Linker Comma...   ✓
      MK60N512VMD100_flash_BM.lcf       1 KB  Linker Comma...   ✓
      MK60N512VMD100_flash.lcf          1 KB  Linker Comma...   ✓
      MK60N512VMD100_ram.lcf            1 KB  Linker Comma...   ✓
```

Although not 100% compatible with the more traditional project management layout the result is close. It is now possible to share general files between multiple projects and the setup files and resulting output files are located in equivalent sub-directories to the other IDEs.

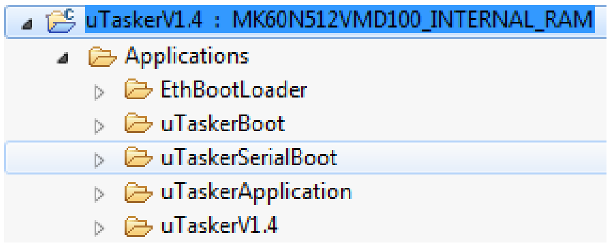
There is a complication in that the default output location for files is at the same directory level as the "`Applications`" directory. How this is manipulated is described in the following section which handles creating a new project in this imported project environment.

3. Creating a New Project in the Environment

The µTasker project is designed to enable its use in new projects, rather than having to start these with a `main()` routine and build up the code from various (possibly existing) sources. The standard technique is to make a copy of the uTaskerV1.4 project in the "`Applications`" directory, rename it and start adapting iots configuration to suit the specific project's goals, removing or adding source as fit.

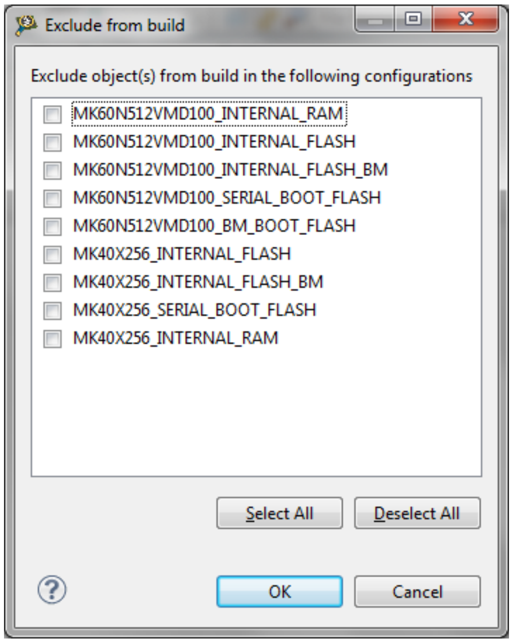
Since the CodeWarrior 10 project has been imported but its source files are referenced (not actually copied and used in the IDE's workspace) a new project is started by copying the uTaskerV1.4 (or another existing project directory in the project structure) and renaming the directory to suit.

The following shows the result after the copied directory has been renamed to `uTaskerApplication` (the name of the new project in this example):



When the CodeWarrior Projects view is refreshed (F5 can be used for this) the new directory also appears in the imported project.

If an existing project is now built there will be errors since the new directory will belong to all targets, therefore the first action is to exclude the new directory from existing target builds. This is fortunately simple to do by clicking on the directory, selecting the context menu (right mouse click) command "**`Resource Configurations | Exclude from Build…`**" and then choosing "**`Select All`**":

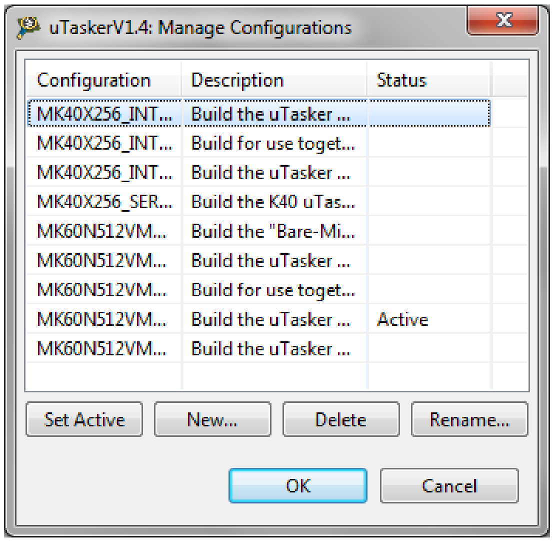

Exclude from build
Exclude object(s) from build in the following configurations
MK60N512VMD100_INTERNAL_RAM
MK60N512VMD100_INTERNAL_FLASH
MK60N512VMD100_INTERNAL_FLASH_BM
MK60N512VMD100_SERIAL_BOOT_FLASH
MK60N512VMD100_BM_BOOT_FLASH
MK40X256_INTERNAL_FLASH
MK40X256_INTERNAL_FLASH_BM
MK40X256_SERIAL_BOOT_FLASH
MK40X256_INTERNAL_RAM
Select All
Deselect All
OK
Cancel

Note that some Eclipse based IDEs don't include this practical command and the user needs to manually exclude every individual file in the directory and its sub-directories which can be a reason to already give up with the tool…!

4. Creating a New Target

Normally a new target would be created for a different processor type or to distinguish between whether the build is to run ion Flash or SRAM, but we use the target as project in order to allow all projects to co-exist in a single environment and share general code sources. Creating a new target is very simple too:

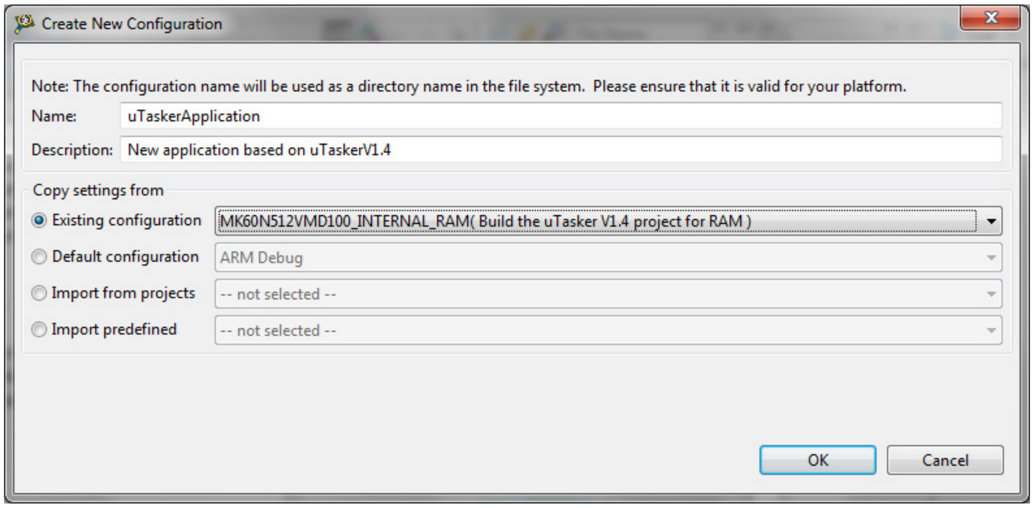
Select the project (the top level folder "`uTaskerV1.4`") in the CodeWarrior Projects view and then use the menu "**`Projects | Build Configuration | Manage…`**" to open up the following dialogue:



uTaskerV1.4: Manage Configurations

Configuration	Description	Status
MK40X256_INT...	Build the uTasker ...	
MK40X256_INT...	Build for use toget...	
MK40X256_INT...	Build the uTasker ...	
MK40X256_SER...	Build the K40 uTas...	
MK60N512VM...	Build the "Bare-Mi...	
MK60N512VM...	Build the uTasker ...	
MK60N512VM...	Build for use toget...	
MK60N512VM...	Build the uTasker ...	Active
MK60N512VM...	Build the uTasker ...	

Select "**`New…`**" and enter a name and description for the new target, basing it on the original one that it is a copy of:



Create New Configuration

Note: The configuration name will be used as a directory name in the file system. Please ensure that it is valid for your platform.

Name: uTaskerApplication

Description: New application based on uTaskerV1.4

Copy settings from

[✓ Existing configuration] MK60N512VMD100_INTERNAL_RAM(Build the uTasker V1.4 project for RAM)

Default configuration ARM Debug

Import from projects -- not selected --

Import predefined -- not selected --

Here it can be seen that a target for operation in internal SRAM has been used as base for the new configuration. The procedure needs to be repeated for each such 'real' target the new project needs to use.

5. Configuring a New Target

If the new target is now built it will be found that it is still building the original uTaskerV1.4 project. This is due to the fact that it has inherited the original target's settings, including the fact that it is using the content of `Applications/uTaskerV1.4` instead of `Applications/uTaskerApplication`. To rectify this the command "**`Resource Configurations | Exclude from Build…`**" can be used again on the project's directory and this time allow its content to be used by it and exclude the original uTaskerV1.4 folder.

In addition it is necessary to exclude the sub-directory "`Simulator`", although it is just as well to exclude all sub-directories in `Applications/uTaskerApplication`.

At this point it will be possible to build the target (project) [ ] but there are some problems with the target settings and generated target locations that will need to be solved before being finished. For example, the linker script file being used is still from the `uTaskerV1.4` folder and the projects make file and generated output is being located in a directory that the environment created called `uTaskerApplication` but at the same directory level as the directory Applications.

Up until now things have been quite simple but to fix but the last little quirks are a bit more complicated since we need now to get involved with paths and build settings. The following step by step guide shows how it is done – it is not particularly pleasant but, if followed exactly, results in the desired effects.

To enter the project configuration select the project (the top level folder "`uTaskerV1.4`") in the CodeWarrior Projects view and then use the menu "**`Projects | Properties`**" to open up the properties pane:

1 – Create a new path variable for the application in "**`Resource | Linked Resources | Path`**" Variables:
Using the "**`New…`**" button enter a path variable for the new project (for example `UTASKER_APPLICATION_LOC`) and set it to `PROJECT_LOC\Applications\uTaskerApplication\KinetisCodeWarrior`. Note that `PROJECT_LOC` is automatically pointing to the source file locations.
This new path variable will be helpful for the next steps.
It is a good idea to save this setting before continuing with the following so that the path variable is already known.

2 – Correct the output directory for generated files in "**`C/C++ Build | Builder Settings`**" `${ProjDirPath}/uTaskerApplication` (this is the default that was created for the target) to `${UTASKER_APPLICATION_LOC}/MK60N512VMD100_INTERNAL_RAM` (the path variable is important and the target directory can be chosen to suit)

3 – Correct the linker script path in "**`C/C++ Build | Settings | Tool Settings | ARM Linker | Input`**". For the "**`Linker Command File`**" simply replace `${APPLICATION_LOC}` by `${ UTASKER_APPLICATION_LOC}`

4 – Correct the include file search path in **`C/C++ Build | Settings | Tool Settings | ARM Compiler | Input`**. For the "**`Include User Serach Paths`**" simply replace the path by `"${UTASKER_APPLICATION_LOC}\.."` – be careful here since the `\..` at the end is important, as are the "" !

5 – Finally change the output file in "**`C/C++ Build | Settings | Build Artifact`**" from `uTasker1.4` to `utaskerApplication`.

If this if followed carefully and typing mistakes avoided it should now build correctly, using the headers from the new target directory and the linker script from the new target settings directory and generate the output file to the appropriate location.

In case of problems it may help to close CodeWarrior 10 once and start again to ensure that all new paths are being used.

Once successful the project development itself can be continued using CodeWarrior as editor, build and debug environment.

6. Conclusion

This document explains in detail one method of creating new µTasker projects in the µTasker project environment using CodeWarrior 10 as IDE.

The method explained uses targets as projects to get around some restrictions imposed by the Eclipse workspace operation. These methods work well but may not be the best. Furthermore they may be better methods that could be used in the present version of CodeWarrior 10 or may become possible in future versions as Eclipse and CodeWarrior itself are optimised.

Modifications:
- V0.0 13.12.2012 First version