

TASKING[®]

***Getting Started with the
LAPACK Performance
Libraries***

Copyright © 2017 TASKING BV.

All rights reserved. You are permitted to print this document provided that (1) the use of such is for personal use only and will not be copied or posted on any network computer or broadcast in any media, and (2) no modifications of the document is made. Unauthorized duplication, in whole or part, of this document by any means, mechanical or electronic, including translation into another language, except for brief excerpts in published reviews, is prohibited without the express written permission of TASKING BV. Unauthorized duplication of this work may also be prohibited by local statute. Violators may be subject to both criminal and civil penalties, including fines and/or imprisonment. Altium[®], TASKING[®], and their respective logos are registered trademarks of Altium Limited or its subsidiaries. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.

Table of Contents

1. Introduction to the LAPACK Performance Libraries	1
2. Preparing for First Use	3
2.1. Installing the Software	3
2.1.1. Installation for Windows	3
2.1.2. Licensing	4
2.2. Building the LAPACK Performance Libraries	8
2.3. How to Use the Documentation	10
3. Using the LAPACK Performance Libraries	11
4. Example Project	13

Chapter 1. Introduction to the LAPACK Performance Libraries

LAPACK (Linear Algebra PACKage) is a software library for numerical linear algebra. It provides routines for solving systems of linear equations and linear least squares, eigenvalue problems, and singular value decomposition. It also includes routines to implement the associated matrix factorizations such as LU, QR, Cholesky and Schur decomposition. LAPACK was written in Fortran 90 and the original routines handle both real and complex matrices in both single and double precision.

The LAPACK Performance Libraries are based on LAPACK 3.7.0 (see the Netlib site at <http://www.netlib.org/lapack>). The Fortran code is converted to C code and the provided functions handle real matrices and single precision only.

The LAPACK Performance Libraries are comprised of three separate physical libraries:

- LAPACK principal function library
- BLAS basic function library
- F2C support function library

LAPACK (Linear Algebra PACKage)

This is the main library for numerical linear algebra. LAPACK functions are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS).

BLAS (Basic Linear Algebra Subprograms)

This library contains functions that provide standard building blocks for performing basic vector and matrix operations. These functions form the low-level layer of the LAPACK functions. Machine-specific optimized BLAS libraries are available for a variety of computer architectures. The LAPACK Performance Libraries provide such an optimized BLAS implementation for Infineon TriCore.

F2C (Fortran to C)

The F2C library is a run-time library that is used by the LAPACK and BLAS library.

Chapter 2. Preparing for First Use

This chapter guides you through the installation process of the LAPACK Performance Libraries. It also describes which documentation is available and how you best can use it.

2.1. Installing the Software

This section describes the installation of the software for Windows. TASKING products are protected with TASKING license management software (TLM). To use a TASKING product, you must install that product and install a license.

2.1.1. Installation for Windows

System Requirements

Before installing, make sure the following minimum system requirements are met:

- Windows 7 or higher
- 2 GHz Pentium class processor
- 1 GB memory
- 500 MB free hard disk space

Installation

Do **not** install the product in an existing TASKING VX-toolset installation directory, since this may damage your TASKING VX-toolset license file.

1. If you received a download link, download the software and extract its contents.
- or -
If you received an USB flash drive, insert it into a free USB port on your computer.
2. Run the installation program (**setup.exe**).
The TASKING Setup dialog box appears.
3. Select a product and click on the **Install** button. If there is only one product, you can directly click on the **Install** button.
4. Follow the instructions that appear on your screen. During the installation you need to enter a license key, this is described in [Section 2.1.2, Licensing](#).

After you have installed the product, you can continue with [Section 2.2, Building the LAPACK Performance Libraries](#).

2.1.2. Licensing

TASKING products are protected with TASKING license management software (TLM). To use a TASKING product, you must install that product and install a license.

The following license types can be ordered from Altium.

Node-locked license

A node-locked license locks the software to one specific computer so you can use the product on that particular computer only.

For information about installing a node-locked license see [Section 2.1.2.3.2, *Installing Server Based Licenses \(Floating or Node-Locked\)*](#) and [Section 2.1.2.3.3, *Installing Client Based Licenses \(Node-Locked\)*](#).

Floating license

A floating license is a license located on a license server and can be used by multiple users on the network. Floating licenses allow you to share licenses among a group of users up to the number of users (seats) specified in the license.

For example, suppose 50 developers may use a client but only ten clients are running at any given time. In this scenario, you only require a ten seats floating license. When all ten licenses are in use, no other client instance can be used.

For information about installing a floating license see [Section 2.1.2.3.2, *Installing Server Based Licenses \(Floating or Node-Locked\)*](#).

License service types

The license service type specifies the process used to validate the license. The following types are possible:

- **Client based** (also known as 'standalone'). The license is serviced by the client. All information necessary to service the license is available on the computer that executes the TASKING product. This license service type is available for node-locked licenses only.
- **Server based** (also known as 'network based'). The license is serviced by a separate license server program that runs either on your companies' network or runs in the cloud. This license service type is available for both node-locked licenses and floating licenses.

Licenses can be serviced by a cloud based license server called "**Remote TASKING License Server**". This is a license server that is operated by TASKING. Alternatively, you can install a license server program on your local network. Such a server is called a "**Local TASKING License Server**". You have to configure such a license server yourself. The installation of a local TASKING license server is not part of this manual. You can order it as a separate product (SW000089).

The benefit of using the Remote TASKING License Server is that product installation and configuration is simplified.

Unless you have an IT department that is proficient with the setup and configuration of licensing systems we recommend to use the facilities offered by the Remote TASKING License Server.

2.1.2.1. Obtaining a License

You need a license key when you install a TASKING product on a computer. If you have not received such a license key follow the steps below to obtain one. Otherwise, you cannot install the software.

Obtaining a server based license (floating or node-locked)

- Order a TASKING product from Altium or one of its distributors.

A license key will be sent to you by email or on paper.

If your node-locked server based license is not yet bound to a specific computer ID, the license server binds the license to the computer that first uses the license.

Obtaining a client based license (node-locked)

To use a TASKING product on one particular computer with a license file, Altium needs to know the computer ID that uniquely identifies your computer. You can do this with the **getcid** program that is available on the TASKING website. The detailed steps are explained below.

1. Download the **getcid** program from <http://www.tasking.com/support/tlm/download.shtml>.
2. Execute the **getcid** program on the computer on which you want to use a TASKING product. The tool has no options. For example,

```
C:\Tasking\getcid  
Computer ID: 5Dzm-L9+Z-WFbO-aMkU-5Dzm-L9+Z-WFbO-aMkU-MDAy-Y2Zm
```

The computer ID is displayed on your screen.

3. Order a TASKING product from Altium or one of its distributors and supply the computer ID.

A license key and a license file will be sent to you by email or on paper.

When you have received your TASKING product, you are now ready to install it.

2.1.2.2. Frequently Asked Questions (FAQ)

If you have questions or encounter problems you can check the support page on the TASKING website.

<http://www.tasking.com/support/tlm/faq.shtml>

This page contains answers to questions for the TASKING license management system TLM.

If your question is not there, please contact your nearest Altium Sales & Support Center or Value Added Reseller.

2.1.2.3. Installing a License

The license setup procedure is done by the installation program.

If the installation program can access the internet then you only need the licence key. Given the license key the installation program retrieves all required information from the remote TASKING license server.

Getting Started with the LAPACK Performance Libraries

The install program sends the license key and the computer ID of the computer on which the installation program is running to the remote TASKING license server. No other data is transmitted.

If the installation program cannot access the internet the installation program asks you to enter the required information by hand. If you install a node-locked client based license you should have the license file at hand (see [Section 2.1.2.1, Obtaining a License](#)).

Floating licenses are always server based and node-locked licenses can be server based. All server based licenses are installed using the same procedure.

2.1.2.3.1. Configure the Firewall in your Network

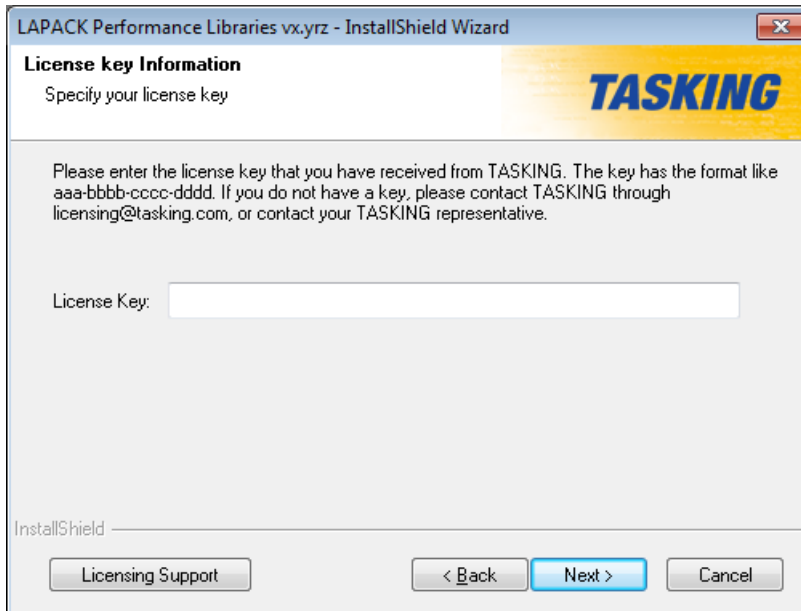
For using the TASKING license servers the TASKING license manager tries to connect to the Remote TASKING servers `lic1.tasking.com` .. `lic4.tasking.com` at the TCP ports 8080, 8936 or 80. Make sure that the firewall in your network has transparent access enabled for one of these ports.

2.1.2.3.2. Installing Server Based Licenses (Floating or Node-Locked)

If you do not have received your license key, read [Section 2.1.2.1, Obtaining a License](#) before you continue.

1. If you want to use a local license server, first install and run the local license server before you continue with step 2. You can order a local license server as a separate product (product code SW000089).
2. Install the TASKING product and follow the instruction that appear on your screen.

The installation program asks you to enter the license information.



The screenshot shows a window titled "LAPACK Performance Libraries vx.yrz - InstallShield Wizard". The window has a yellow header with the "TASKING" logo. Below the header, the text reads "License key Information" and "Specify your license key". A paragraph of instructions follows: "Please enter the license key that you have received from TASKING. The key has the format like aaa-bbbb-cccc-dddd. If you do not have a key, please contact TASKING through licensing@tasking.com, or contact your TASKING representative." Below this text is a text input field labeled "License Key:". At the bottom of the window, there are four buttons: "Licensing Support", "< Back", "Next >", and "Cancel". The "Next >" button is highlighted in blue.

3. In the **License key** field enter the license key you have received from Altium and click **Next** to continue.

*The installation program tries to retrieve the license information from a remote TASKING license server. Wait until the license information is retrieved. If the license information is retrieved successfully subsequent dialogs are already filled-in and you only have to confirm the contents of the dialogs by clicking the **Next** button. If the license information is not retrieved successfully you have to enter the information by hand.*

4. Select your **License type** and click **Next** to continue.

You can find the license type in the email or paper that contains the license key.

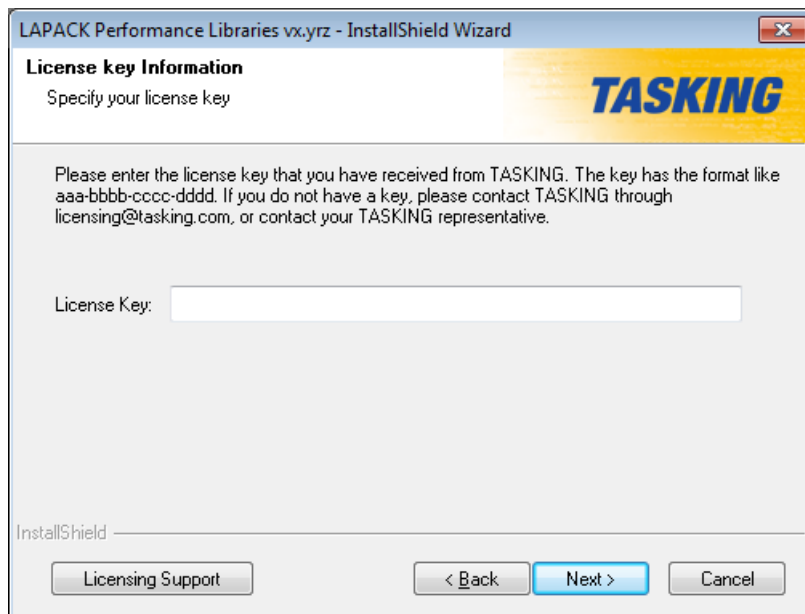
5. Select **Remote TASKING license server** to use one of the remote TASKING license servers, or select **Local TASKING license server** for a local license server. The latter requires optional software.
6. (For local license server only) specify the **Server name** and **Port number** of the local license server.
7. Click **Finish** to complete the installation.

2.1.2.3.3. Installing Client Based Licenses (Node-Locked)

If you do not have received your license key and license file, read [Section 2.1.2.1, Obtaining a License](#) before continuing.

1. Install the TASKING product and follow the instruction that appear on your screen.

The installation program asks you to enter the license information.



Getting Started with the LAPACK Performance Libraries

2. In the **License key** field enter the license key you have received from Altium and click **Next** to continue.

*The installation program tries to retrieve the license information from a remote TASKING license server. Wait until the license information is retrieved. If the license information is retrieved successfully subsequent dialogs are already filled-in and you only have to confirm the contents of the dialogs by clicking the **Next** button. If the license information is not retrieved successfully you have to enter the information by hand.*

3. Select **Node-locked client based license** and click **Next** to continue.
4. In the **License file content** field enter the contents of the license file you have received from Altium.

The license data is stored in the file licfile.txt in the etc directory of the product.

5. Click **Finish** to complete the installation.

2.2. Building the LAPACK Performance Libraries

The LAPACK Performance Libraries are shipped as a set of source files and makefiles. In order to use the libraries, you need to extract the source files and build the libraries first.

Open Command Prompt with Administrator rights

For Windows, the LAPACK Performance Libraries will be installed in the 'Program Files (x86)' folder by default. This folder has only limited write access, so in order to be able to unpack and build the libraries from within this location you need Windows Administrator rights. To be able to continue you have to start a Command Prompt as administrator. You can achieve this by following these steps:

1. From the Windows **Start** button, select **All Programs » Accessories**.
2. Right-click **Command Prompt**, and then click **Run as administrator**.
3. If the **User Account Control** dialog box appears, confirm that the action it displays is what you want, and then click **Yes**.

If you do not have administrator rights on your PC, then install the LAPACK Performance Libraries outside the 'Program Files (x86)' folder, by selecting a different location when the installation program asks you to provide an installation folder.

Unpack, build and copy

For the next instruction it is assumed that the product is installed in the default location and that you have opened a Command Prompt with administrator rights.

In directory `etc\lib\src.lapack` of the LAPACK Performance Libraries product a makefile is available that takes care of unpacking, building and copying the LAPACK Performance Libraries. It is advised that you use this makefile. The makefile executes the following commands for you:

1. It unpacks the libraries by running the self extracting ZIP files: **unpack-tc-libblas-sources.exe**, **unpack-tc-libf2c-sources.exe** and **unpack-tc-liblapack-sources.exe**. Note that it is required for you to have a valid license key for unpacking the libraries. Refer to [Section 2.1.2, Licensing](#) for more information.
2. As a next step the LAPACK, BLAS and F2C libraries are built with the TriCore VX-toolset. There are two sets of makefiles, one for AURIX (located in subdirectory `tc16x`) and one for AURIX 2G (located in subdirectory `tc162`). In total there are six sub makefiles available:

```
<installation-dir>\ctc\lib\src.lapack\
|
| - blas\tc16x\libblas_fpu\makefile
| - blas\tc162\libblas_fpu\makefile
| - f2c\tc16x\libf2c_fpu\makefile
| - f2c\tc162\libf2c_fpu\makefile
| - lapack\tc16x\liblapack_fpu\makefile
| - lapack\tc162\liblapack_fpu\makefile
```

3. As a last step, the libraries are copied to the proper location. The AURIX libraries are copied to directory `ctc\lib\tc16x`, while the AURIX 2G libraries are copied to directory `ctc\lib\tc162` of the LAPACK Performance Libraries product.

Suppose you have purchased TriCore VX-toolset version v6.2r1 and it is installed in the default location: `C:\Program Files (x86)\TASKING\TriCore v6.2r1`. If you want to build the LAPACK Performance Libraries for AURIX and AURIX 2G using this version, then go to the location of the main makefile and run the make utility **amk** by specifying its complete path. You can do this by invoking the following commands in a Command Prompt (replace `vx.yrz` with the version of the LAPACK Performance Libraries product):

```
C:
cd "\Program Files (x86)\TASKING\LAPACK vx.yrz\ctc\lib\src.lapack"
"C:\Program Files (x86)\TASKING\TriCore v6.2r1\ctc\bin\amk"
```

As a result of this make action, the libraries are unpacked, they are built and the AURIX versions are copied to the `ctc\lib\tc16x` directory of the LAPACK Performance Libraries product. The AURIX 2G versions of the libraries are copied to the `ctc\lib\tc162` directory.

It is also possible to build the LAPACK Performance Libraries with an older version of the tools, i.e. TriCore VX-toolset versions v4.x or v5.x. However these older versions of the tools do not support AURIX 2G derivatives. In case you want to build the LAPACK Performance Libraries with an older version, you need to run the make utility **amk** with argument **aurix** to avoid build errors:

```
C:
cd "\Program Files (x86)\TASKING\LAPACK vx.yrz\ctc\lib\src.lapack"
"C:\Program Files (x86)\TASKING\TriCore v5.0r2\ctc\bin\amk" aurix
```

As a result of this make action, only the libraries for AURIX are unpacked, they are built and copied to the `ctc\lib\tc16x` directory of the LAPACK Performance Libraries product.

This completes the preparation of the libraries.

2.3. How to Use the Documentation

The documentation for the LAPACK Performance Libraries consists of:

- this getting started manual
- the official LAPACK User's Guide
- API documentation

It is strongly recommended to read the documentation in this order.

Getting Started with the LAPACK Performance Libraries (this manual)

The getting started manual describes how to install the product, and build the LAPACK Performance Libraries.

The next chapter of this manual explains how to use the LAPACK Performance Libraries in a TASKING VX-toolset for TriCore.

The official LAPACK User's Guide

You can find the official LAPACK User's Guide on the Netlib site at <http://www.netlib.org/lapack/lug>.

LAPACK API Documentation

In the `ctc\doc` directory of the product you can find the API documentation. The API documentation is in HTML format and provided to you as a ZIP file. If you want to use the documentation then unzip the file named `lapack_apidoc.zip` with any unzip program. Note that if you have installed the LAPACK Performance Libraries in the 'Program Files (x86)' folder, you have to run the unzip tool as administrator. By default the HTML files are extracted to directory `ctc\doc\lapack-apidoc`. Open the file `index.html` that is present in this directory with your favorite browser to start navigate the documentation.

Note that the LAPACK Performance Libraries are based on Fortran code that is converted to C code. The API documentation is partially derived from the Fortran code and partially derived from the C code and therefore may not be ideal. You will find function signatures in C notation, where the arguments are denoted in lowercase and in many cases the arguments are pointers. The described parameters on the other hand are in Fortran notation, i.e. parameters are denoted in uppercase and all parameters are pointers implicitly.

Chapter 3. Using the LAPACK Performance Libraries

Once the LAPACK Performance Libraries have been built (see [Section 2.2, Building the LAPACK Performance Libraries](#)), an application can use the LAPACK functions. By default, the LAPACK libraries are configured to assume 4-byte alignment for single-word objects (`int`, `long`, `float`). An application that uses these libraries must be built with the same compiler option (`--eabi=H`) to prevent alignment mismatches. The only thing that is further required is that the tools know where to find the libraries.

Using the command prompt

When you invoke the compiler from the command line, specify the location of the LAPACK header file by using the option `--include-directory` or `-I`.

For example:

```
ctc ... -I"C:\Program Files (x86)\TASKING\LAPACK vx.yrz\ctc\include.lapack"
      --eabi=H myapp.c ...
```

When you invoke the linker, specify the location of the libraries by using the option `--library-directory` or `-L` and the libraries by using the option `--library` or `-l`.

For example:

```
ltc ... myapp.o -L"C:\Program Files (x86)\TASKING\LAPACK vx.yrz\ctc\lib\tc16x"
      -llapack_fpu -lblas_fpu -lf2c_fpu ...
```

Using the TriCore Eclipse IDE

Specify the location of the LAPACK header file in the C/C++ Compiler Options on the Settings properties page, and specify the libraries in the Linker Options on the Settings properties page:

1. From the **Project** menu, select **Properties for**.

The Properties dialog appears.

2. In the left pane, expand **C/C++ Build** and select **Settings**.

In the right pane the Settings page appears.

3. On the Tool Settings tab, select **C/C++ Compiler » Include Paths**.

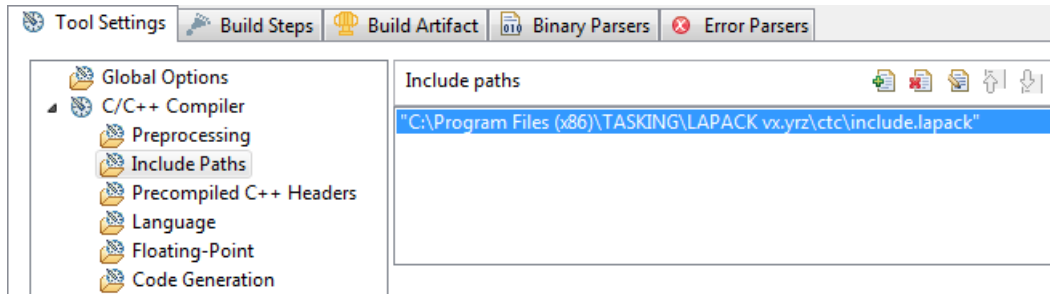
The Include paths box shows the directories that are added to the search path for include files.

4. To define a new directory for the search path, click on the **Add** button in the **Include paths** box.

5. Type or select a path. For example:

```
"C:\Program Files (x86)\TASKING\LAPACK vx.yrz\ctc\include.lapack"
```

Getting Started with the LAPACK Performance Libraries



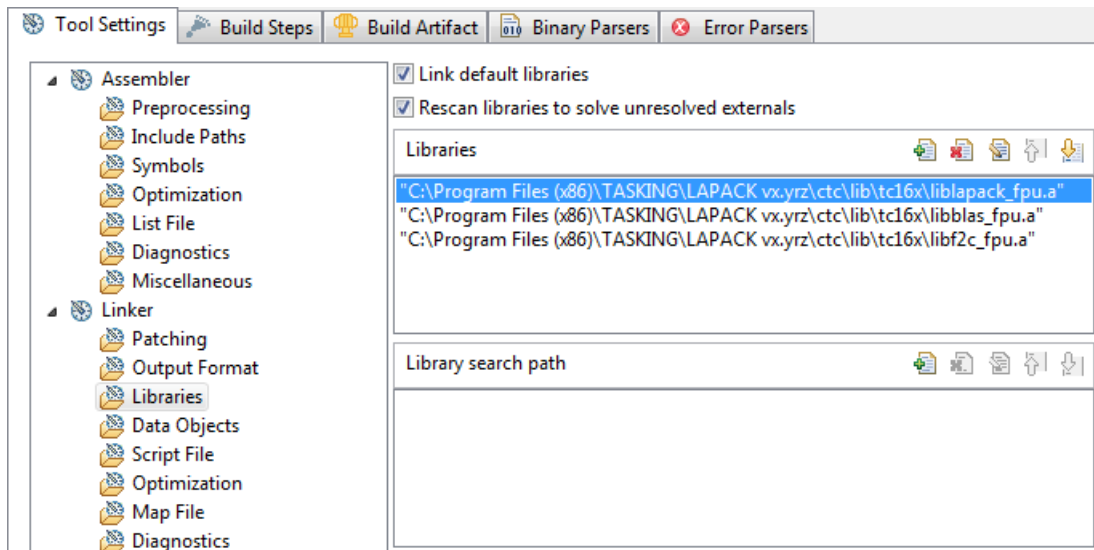
- On the Tool Settings tab, select **C/C++ Compiler** » **Miscellaneous**.
- In the **Additional options** field, enter `--eabi=H`
- On the Tool Settings tab, select **Linker** » **Libraries**.

The Libraries box shows the list of libraries that are linked with the project.

- To add a library, click on the **Add** button in the **Libraries** box.
- Type or select the full path to the LAPACK library:

`"C:\Program Files (x86)\TASKING\LAPACK vx.yrz\ctc\lib\tc16x\liblapack_fpu.a"`

- Repeat steps 7. and 8. for libraries `libblas_fpu.a` and `libf2c_fpu.a`.



Chapter 4. Example Project

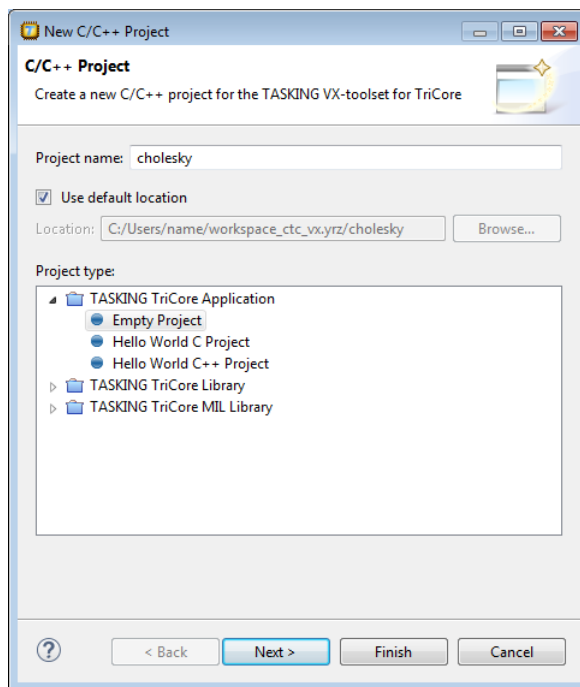
This chapter demonstrates how to use the Cholesky example that is delivered with the product.

The following steps assume that you have installed and built the LAPACK Performance Libraries and that you have installed a TASKING VX-toolset for TriCore.

Create an empty TriCore project with the New C/C++ Project wizard

1. Start the TriCore Eclipse IDE.
2. From the **File** menu, select **New » TASKING TriCore C/C++ Project**

The New C/C++ Project wizard appears.



3. Enter a name for your project, for example `cholesky`.

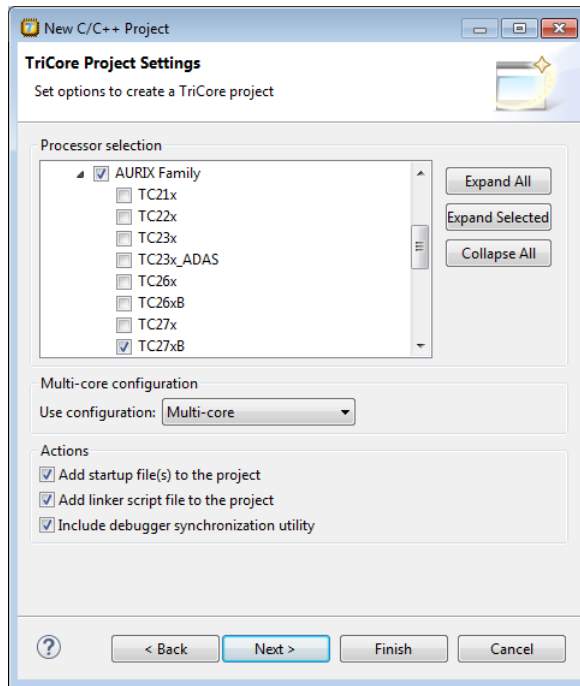
*In the **Location** field you will see the location where the new project will be stored. To change the default location, you can uncheck the **Use default location** check box and browse for an alternative location. However, use the default location for now.*

4. In the **Project type** box you can select whether to create an application or a library.
 - Expand **TASKING TriCore Application** and select **Empty Project**. This creates a project without a C source file containing the function `main()`.

Getting Started with the LAPACK Performance Libraries

- Click **Next** to continue.

The *TriCore Project Settings* page appears.



5. Select the target processor for which you want to build the application. For example TC27xB (under AURIX Family).
6. You can choose to add C startup code to your project and/or add a linker script file to your project.
 - Enable **Add startup file(s) to the project**. This adds the files `cstart.c` and `cstart.h` to your project, which are necessary for building C programs. These files are copies of `lib/src/cstart.c` and `include/cstart.h`. If you do not add the startup file(s) here, you can always add them later with **File » New » Startup Files**.
 - Enable **Add linker script file to the project**. This adds the file `cholesky.lsl` to your project which can be edited to customize linking and locating. If you do not add the linker script file here, you can always add it later with **File » New » Linker Script File (LSL)**.

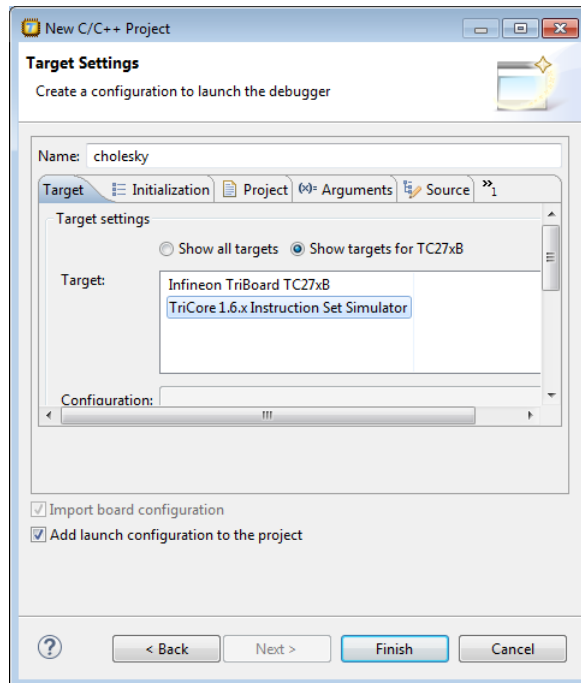
For details on changing the C startup code and linker script file refer to the *TASKING VX-toolset for TriCore User Guide*.

7. When the debugger stops the TriCore, this does not automatically flush all the states in the CPU's pipeline and caches. In order to be able to correctly show the program state, the debugger therefore needs to execute special flushing code every time the CPU halts. When you enable the option **Include debugger synchronization utility** this causes the file `ctc/lib/src/sync_on_halt.c` to be

copied to the project. If you are not going to use the TASKING debugger, you can save a few tens of bytes by disabling this option.

8. Click **Next**.

The Target Settings page appears.



9. In order to debug your project you need to create a debug configuration.
 - Select a target. You can select a target board or a simulator. For this example we select the **TriCore 1.6.x Instruction Set Simulator**.
 - (Optional) If you selected a target board, specify the **Configuration** and **Connection settings**. For the simulator you can skip this.
 - (Optional) If you selected a target board, enable **Import board configuration**. Your project settings, such as memory, flash settings and LSL file (`cholesky.lsl`) and startup code (`cstart.h`) will be adjusted to the selected board configuration for you to build your application.
 - Enable **Add launch configuration to the project**. This allows you to debug your project.
 - Leave the other tabs as is.
10. Click **Finish** to finish the wizard and to create the project.

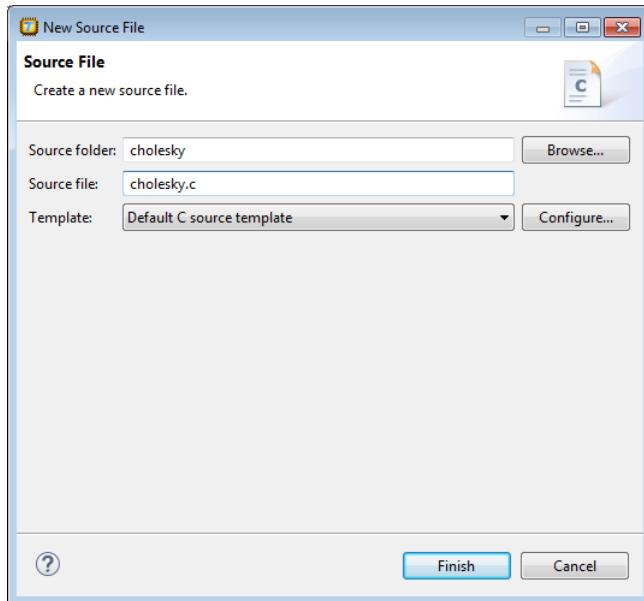
The project has now been created and is the active project.

Add a new file to your project

To add a new, empty file:

1. In the C/C++ Projects view, right-click on the name of the project, `cholesky`, and select **New » Source File**.

The New Source File dialog appears.



2. Specify a source folder and a name for the new file. By default, the new file will be stored in the project folder (in this case: `cholesky`). If your projects contains multiple folders, you can browse for an alternative source folder to store the new file in.

- In the **Source folder** field, make sure it refers to `cholesky`.
- In the **Source file** field, type the name of the new file, for example `cholesky.c`. Note that for C files you must specify the extension `.c`! For C++ files use the extension `.cc` or `.cpp`.
- In the **Template** field, select a code template for your source file, for example `Default C source template` or select `<None>` if you want to start with an empty file. Note that you can configure your own templates if you click on the **Configure...** button.

3. Click **Finish** to continue.

The new file `cholesky.c` is created and ready for editing in the editor view.

4. Enter the following code in the editor view:

```
#include <stdio.h>

#include "lapack.h"

int main(int argc, char ** argv)
{
    float A[3][3] = {{25.f, 15.f, -5.f}, {15.f, 18.f, 0.f}, {-5.f, 0.f, 11.f}};
    int n = 3;
    int lda = 3;
    int info;

    int i;

    /* SPOTRF2 computes the Cholesky factorization of a real symmetric
       positive definite matrix A using the recursive algorithm. */
    (void)spotrf2("L", &n, (float *)A, &lda, &info);

    printf("info=%d\n", info);
    for (i = 0; i < 3; ++i)
    {
        printf ("%f %f %f\n", A[0][i], A[1][i], A[2][i]);
    }

    return 0;
}
```

5. From the **File** menu, select **Save** (Ctrl+S). The project will be saved.

Add LAPACK include directory and libraries and specify C compiler option --eabi=H

- Follow the steps for TriCore Eclipse IDE in [Chapter 3, Using the LAPACK Performance Libraries](#).

Build the project

- From the **Project** menu, select **Build cholesky**.

Debug/run the example project

1. From the **Debug** menu select **Debug project**.

Alternatively you can click the  button in the main toolbar.

The TASKING Debug perspective is associated with the TASKING C/C++ Debugger. Because the TASKING C/C++ perspective is still active, Eclipse asks to open the TASKING Debug perspective.

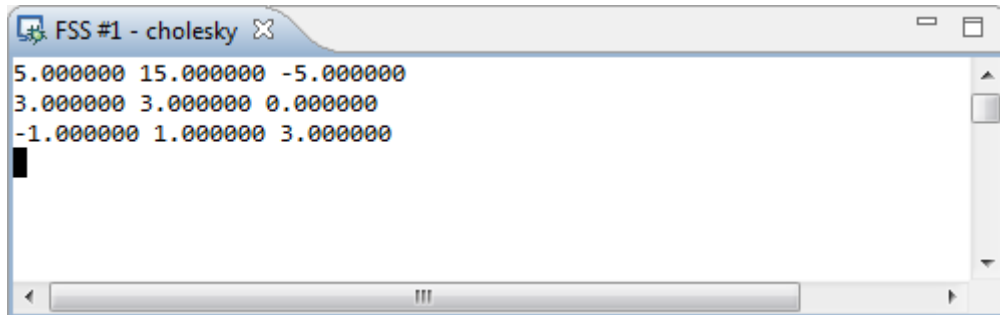
2. Optionally, enable the option **Remember my decision** and click **Yes**.

Getting Started with the LAPACK Performance Libraries

The debug session is launched. This may take a few seconds.

3. Execution is suspended on the function `main()`. You can now step through the source, or click on the **Resume** button (▶) to resume execution.

Output of the example program appears in the FSS view.



```
FSS #1 - cholesky
5.000000 15.000000 -5.000000
3.000000 3.000000 0.000000
-1.000000 1.000000 3.000000
```

This concludes the example. For more information on debugging and using the TASKING VX-toolset, refer to the TASKING VX-toolset for TriCore User Guide.