Contents

[Introduction 1](#_Toc104658429)

[Download GNAT 2020 / GtkAda 2](#_Toc104658430)

[Installing GNAT 2020 3](#_Toc104658431)

[Install GtkAda 7](#_Toc104658432)

[PolyORB 11](#_Toc104658433)

[Installing PolyORB 11](#_Toc104658434)

[Download MinGW 11](#_Toc104658435)

[Verify MinGW Installation 18](#_Toc104658436)

[Make 3.81 20](#_Toc104658437)

[Ada-Lua 20](#_Toc104658438)

[PolyORB 21](#_Toc104658439)

[Configure 21](#_Toc104658440)

[PolyORB\_Config.gpr 22](#_Toc104658441)

[Make 22](#_Toc104658442)

[Verify PolyORB 23](#_Toc104658443)

[Windows Environment 24](#_Toc104658444)

[Install Sisyfos 24](#_Toc104658445)

[Source Code 24](#_Toc104658446)

[Install Tubastga 25](#_Toc104658447)

[Source Code 25](#_Toc104658448)

[Lua Library 25](#_Toc104658449)

[Distributed Hello World 26](#_Toc104658450)

[Source Code 26](#_Toc104658451)

[Execution on one PC 28](#_Toc104658452)

# Introduction

This documentation is written because I think there is a need for some supporting text to people that wish to experiment with Ada and Distributed Annex E / PolyORB on Windows 10. The installation of GNAT 2020 Ada compiler is very similar to how you install any other software on Windows platforms, but the “Distributed Anned E” is not simple, because you need to learn a lot of additional tools in the process which can make it really difficult.

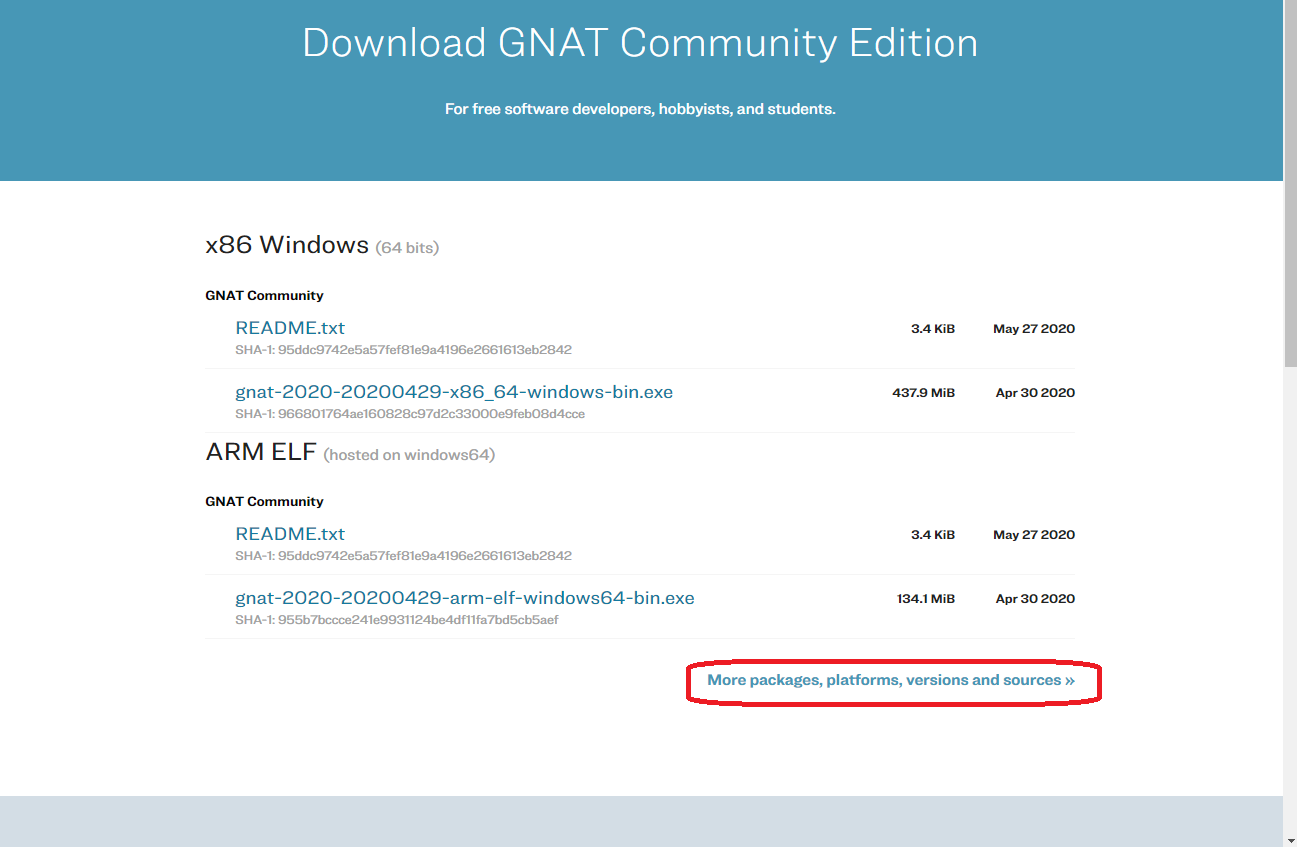
The document will (try☺ ) to make a note on all those small tricks that are useful to go through the installation, and also to make you able to ask questions in forum / mailing lists in a way to which participant there can contribute.

This document is based on an installation on Windows 10, 64-bit Operating stationary PC.

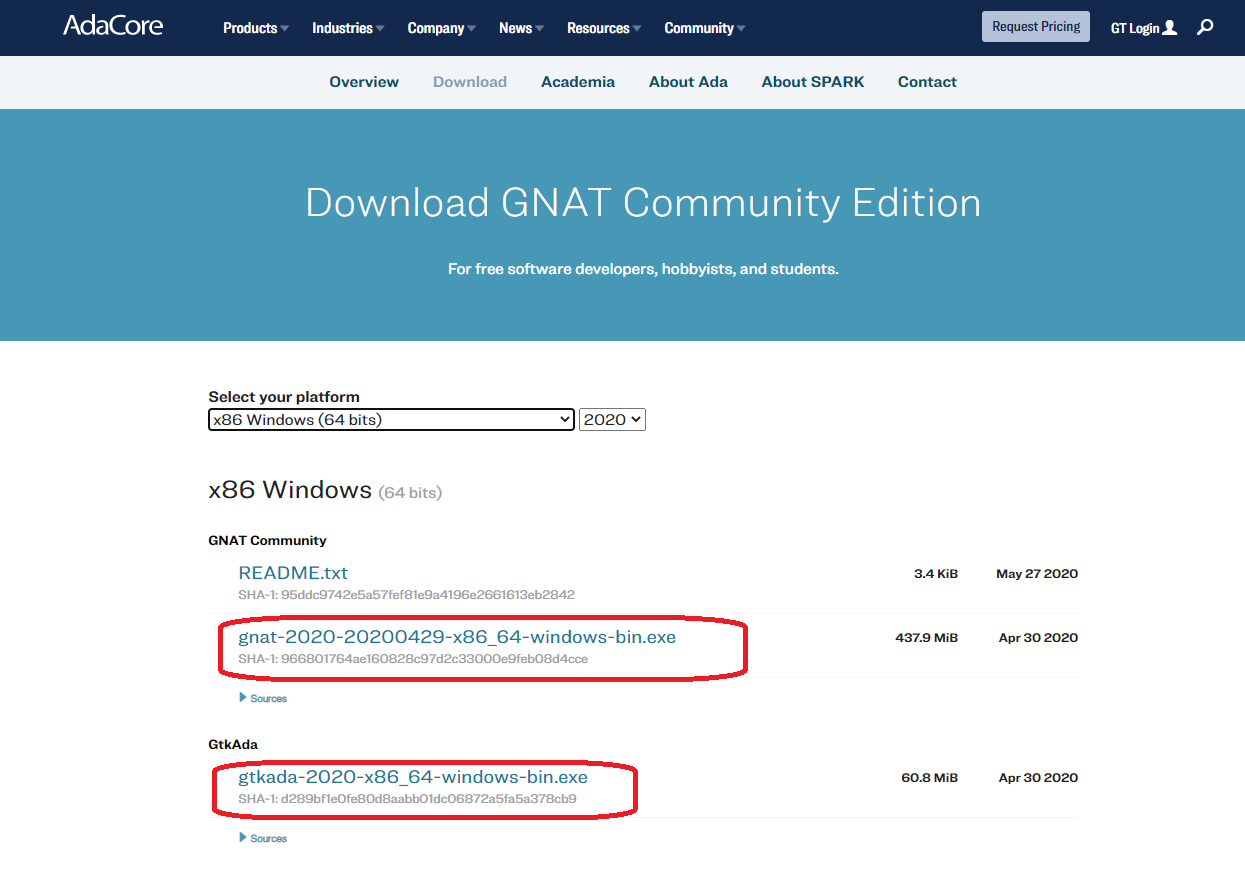
Do avoid installing any of this software on a path that contains spaces – as there are tools in this process that do not accept this.

# Download GNAT 2020 / GtkAda

Go to AdaCore site to download GNAT Community Edition <https://www.adacore.com/download>

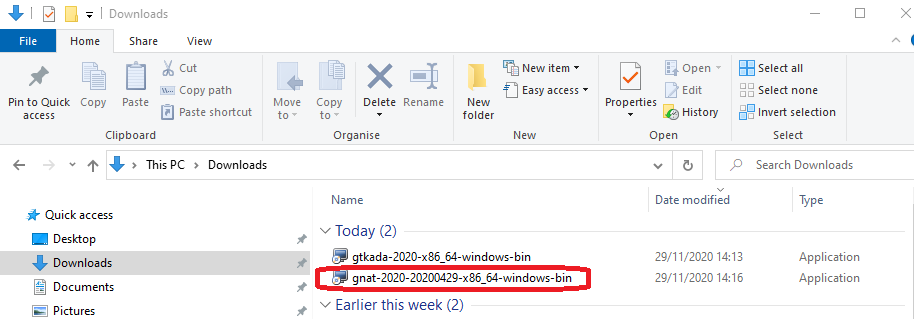


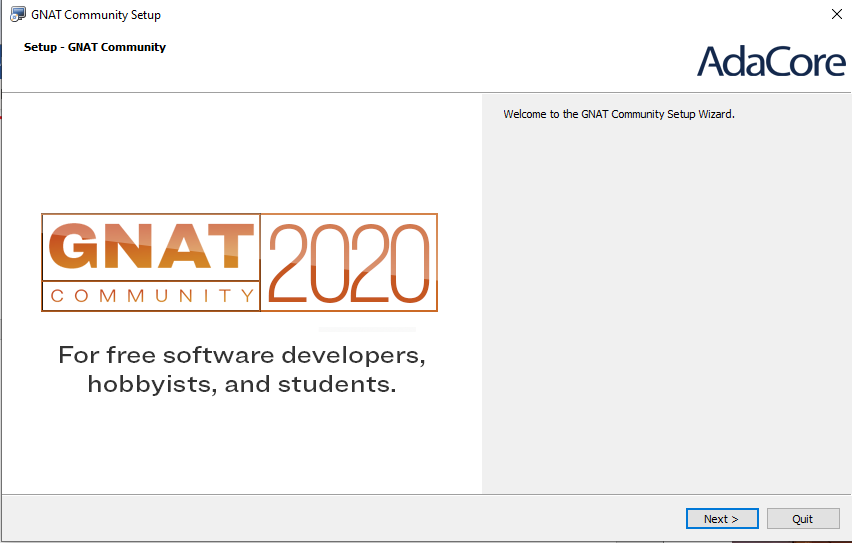
Click on the encircled area on the screen above. This will give you a community version of GNAT that can not be used for commercial development.



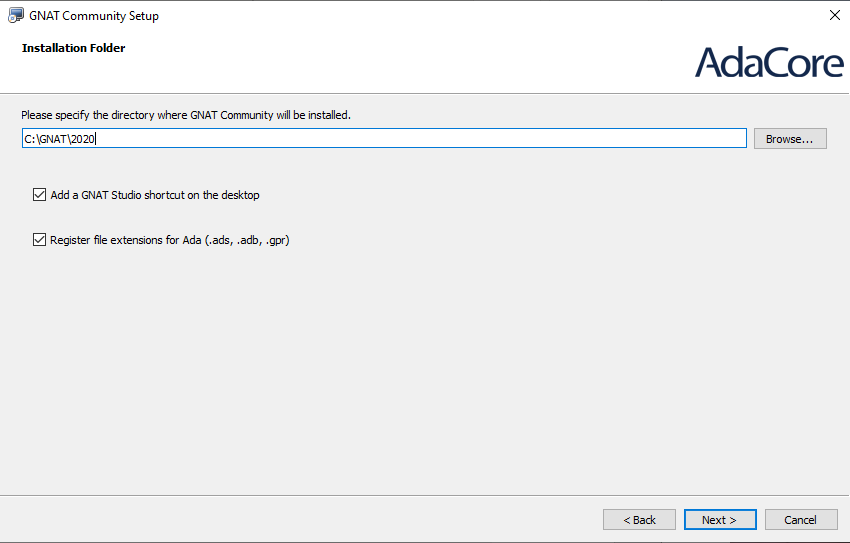
# Installing GNAT 2020

Open the downloaded file for GNAT first:

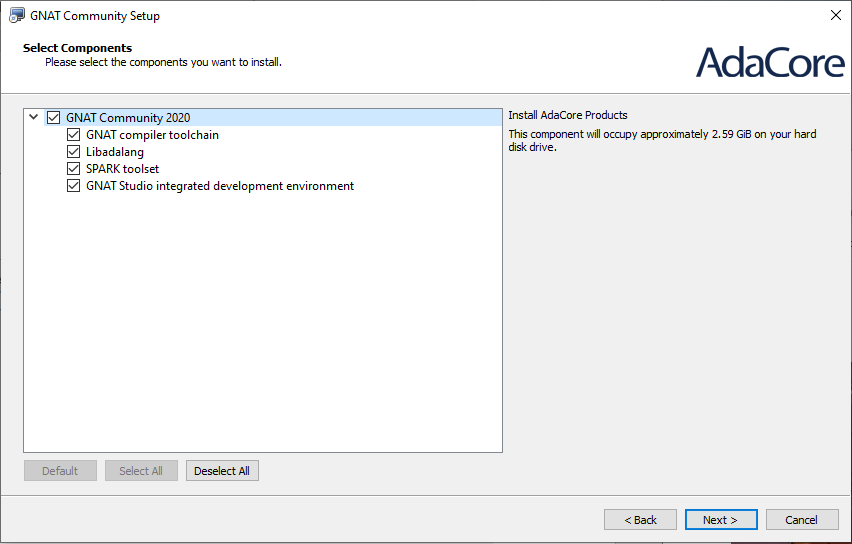




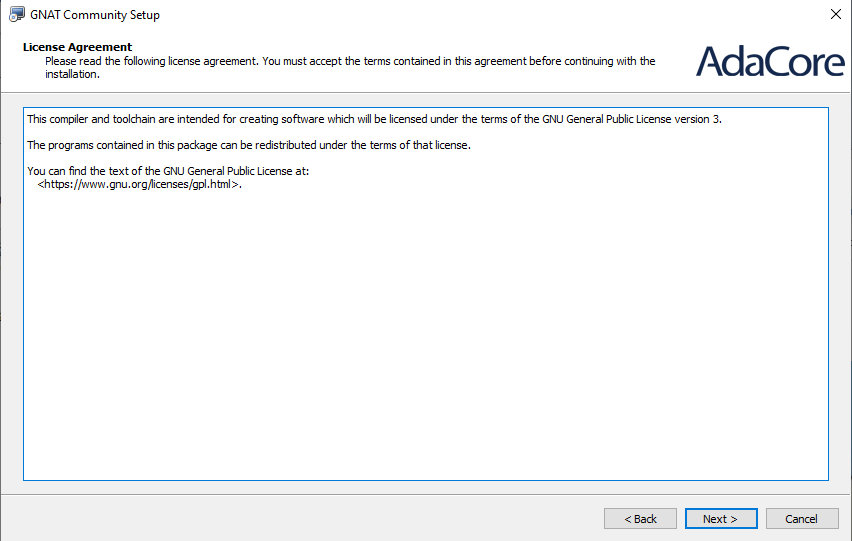
Press Next



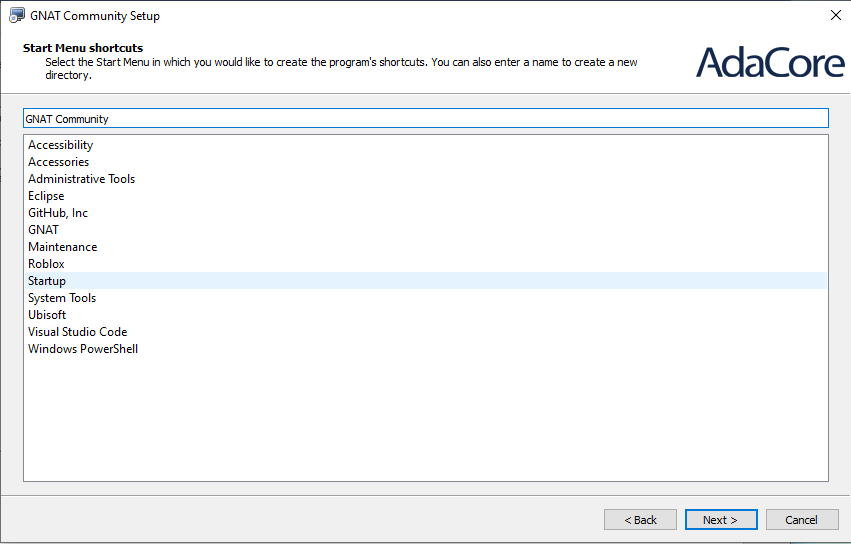
Use the default path.

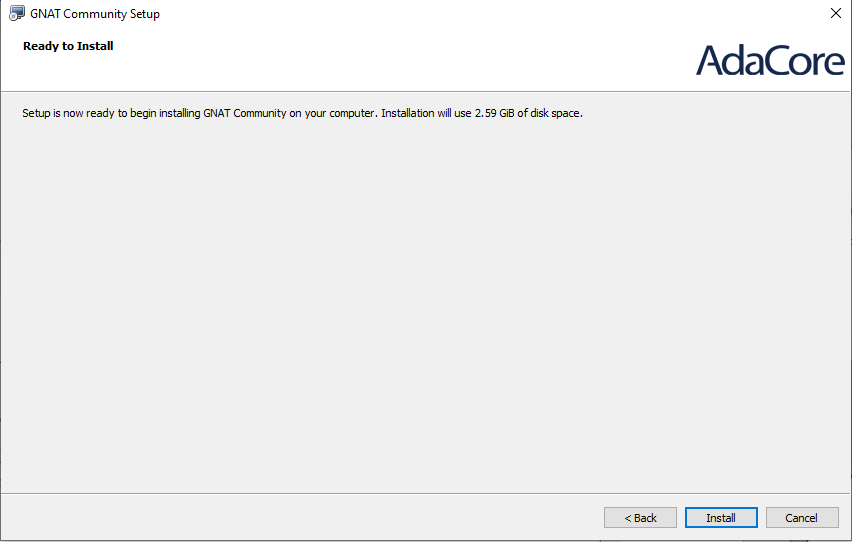


Use these defaults.

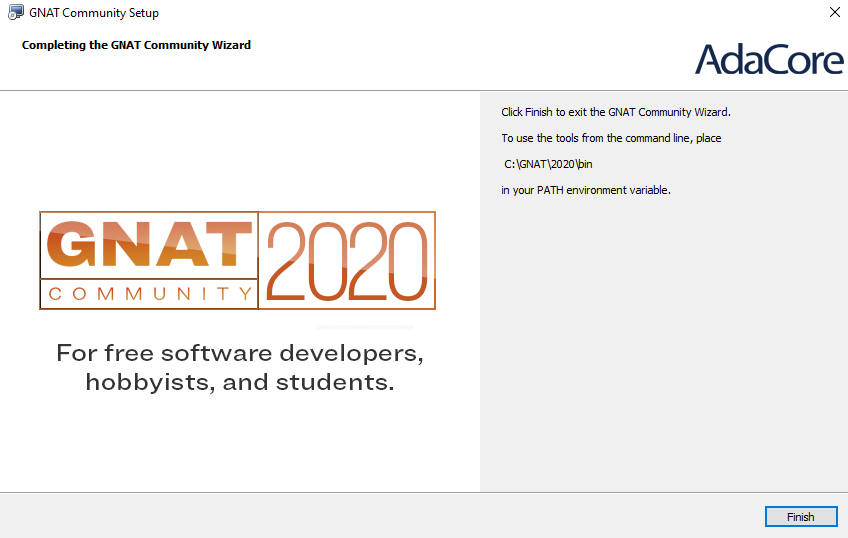


Press Next.





Press Install.



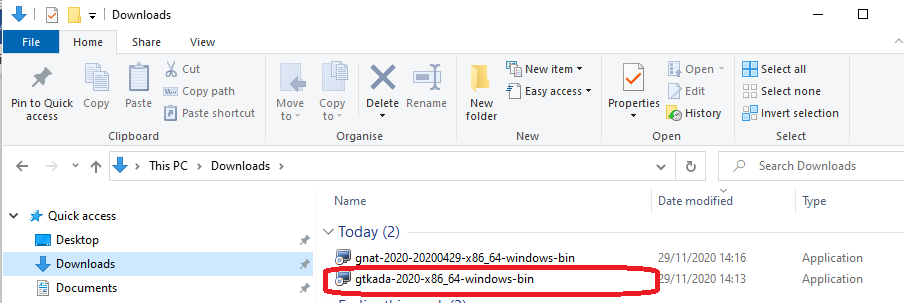
Press Finish

GNAT 2020 GPS and compiler should now be installed.

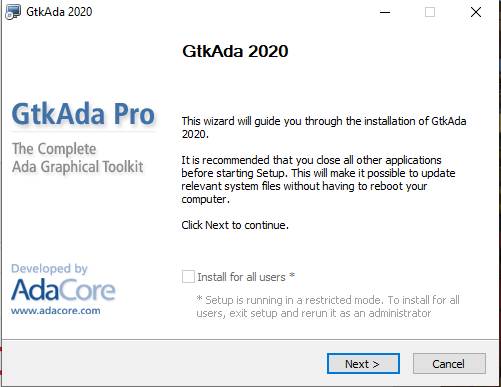
You should have an icon like this on the desktop.

You can try to start the Editor to verify the installation.

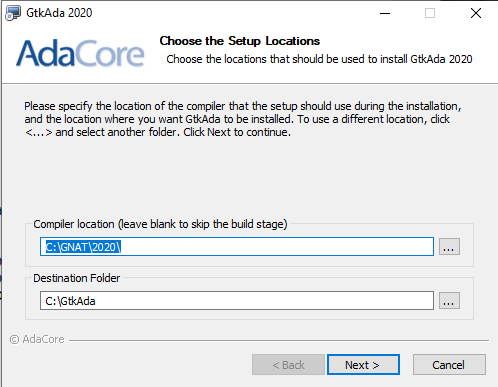
# Install GtkAda



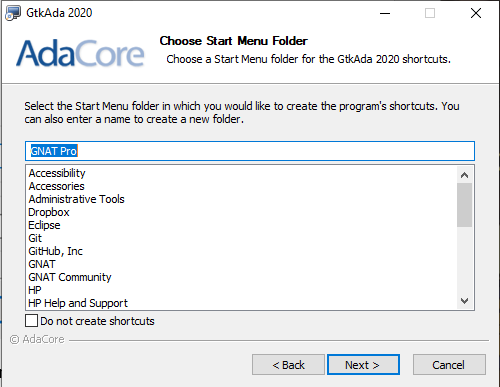
The following window will appear:



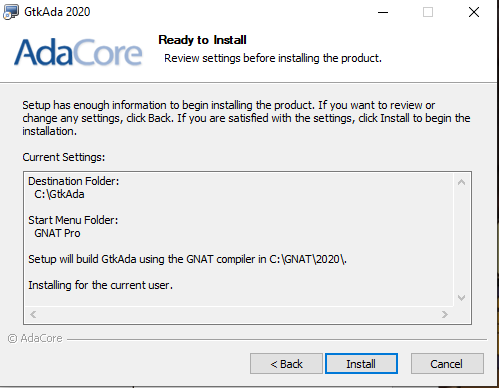
Press Next



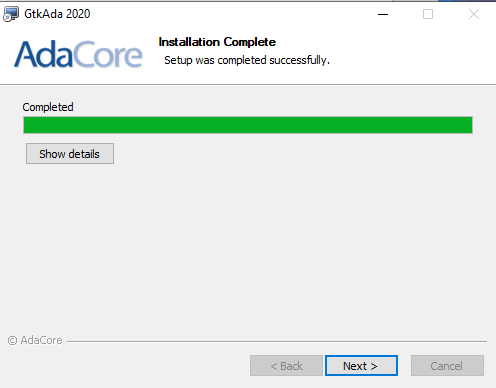
Use the proposed paths (unless you changed them for some reason during earlier steps). If no path is proposed by the installation then use the “…” button to chose the folder “C:/GNAT/2020/”



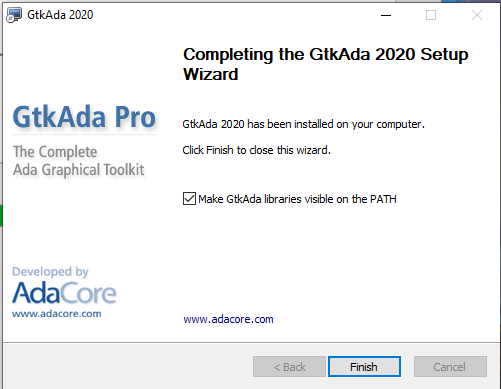
Use the proposed Menu setup.



Press Install.



Press Next.



Press Finish.

# PolyORB

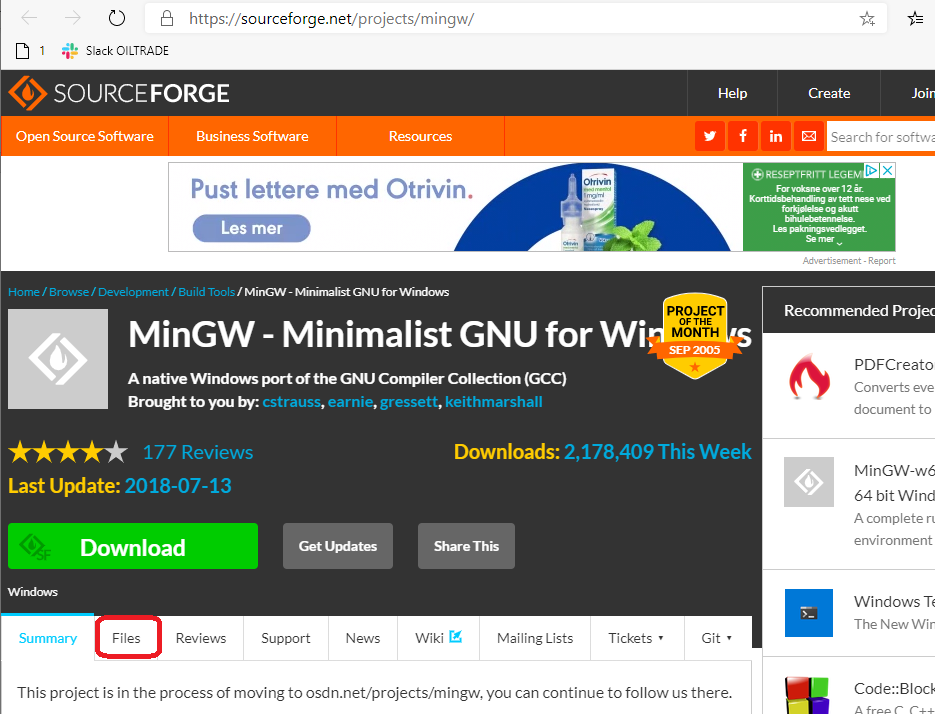
## Installing PolyORB

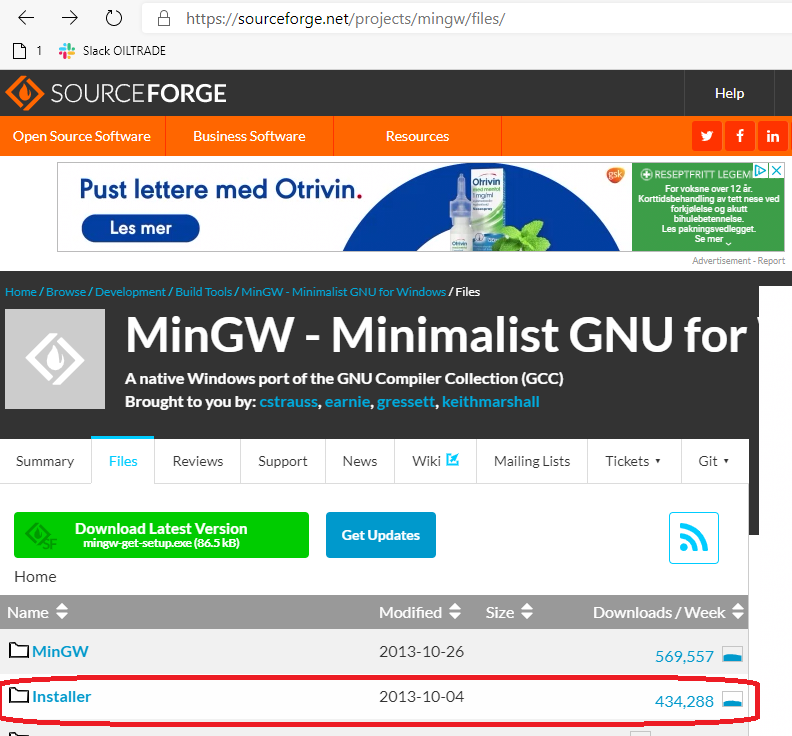
It is now getting trickier. PolyORB is not supplied as a binary package, only as source code. This means you need to compile and build the source code. This must be done in a Unix-like environment because the tools used in this process are from Unix-world.

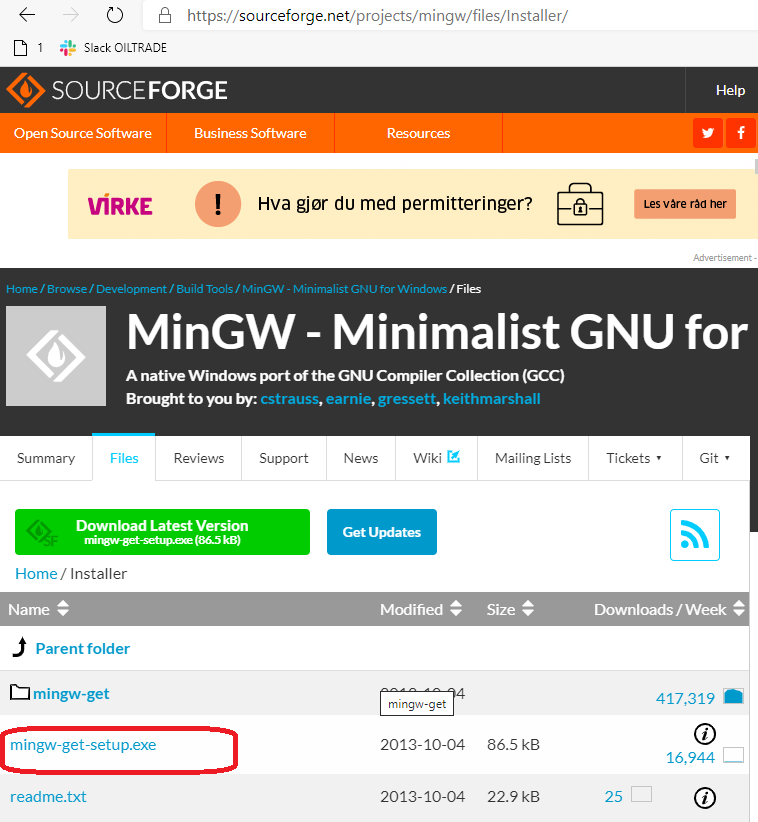
To do this in Windows 10, we need to install a Linux-for-Windows tool. The tool I use is MinGW. This is a big piece of software on it’s own, and it pretty much feels like Unix when you are using it.

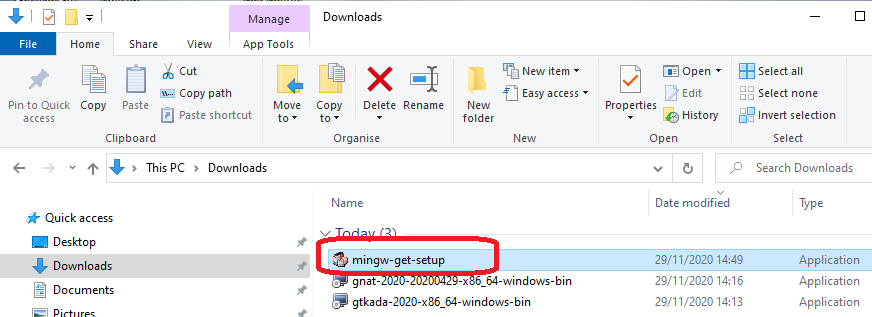
## Download MinGW

MinGW can be downloaded by going to <https://sourceforge.net/projects/mingw/>

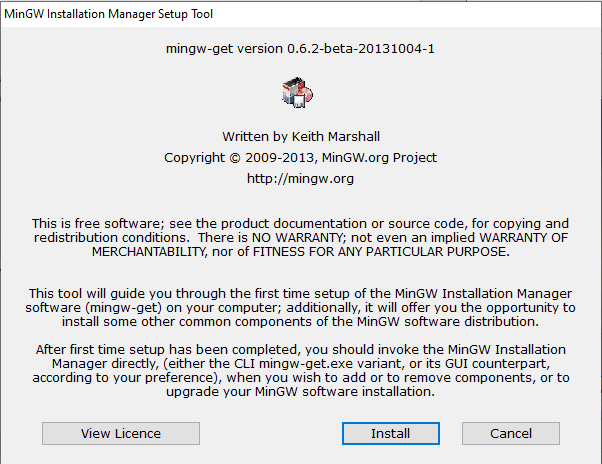




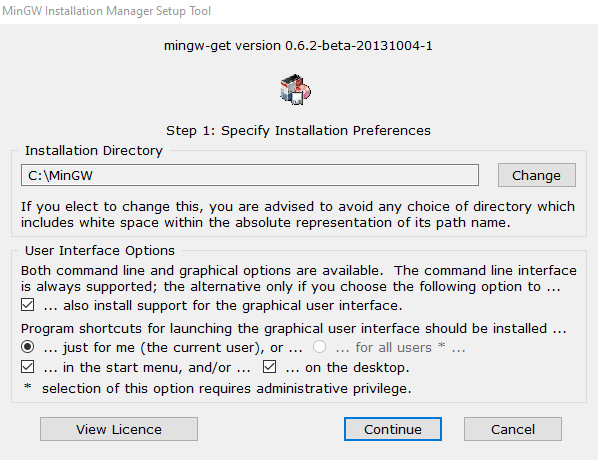




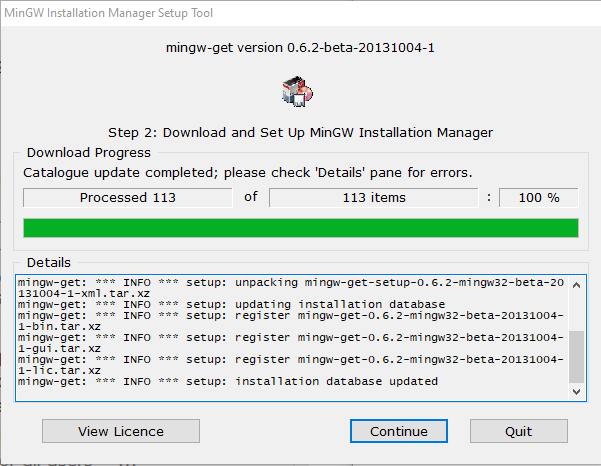
Start the file mingw-get-setup.



Press Install.

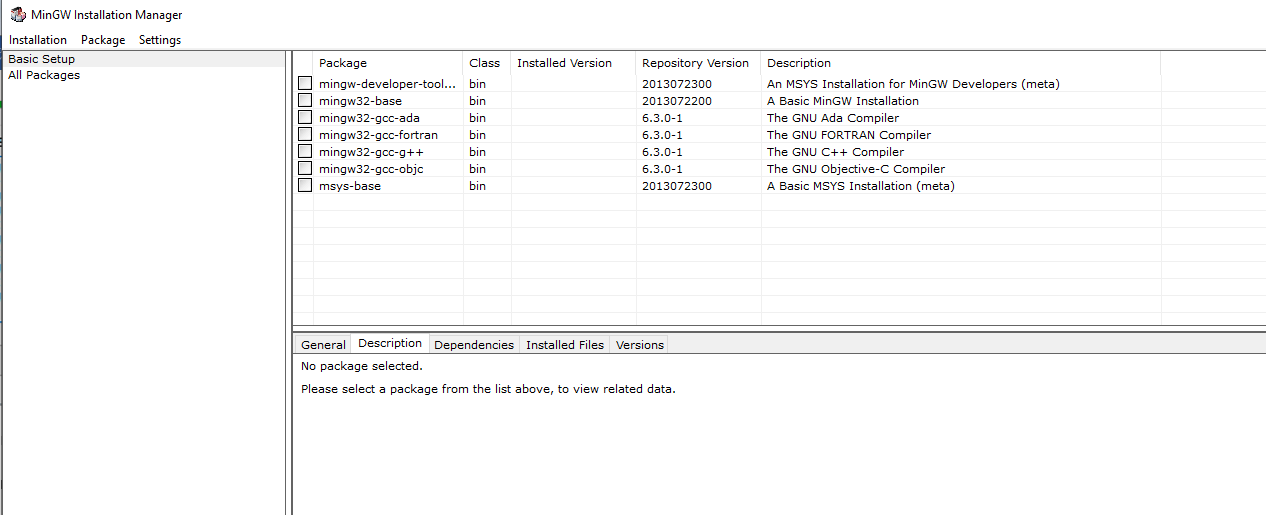


Press Continue.



Press Continue.

The following screen will appear:

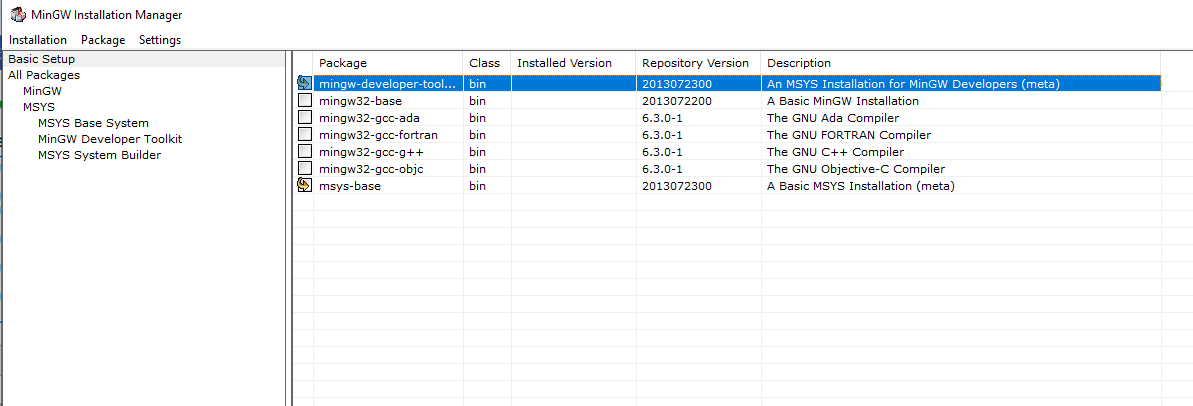


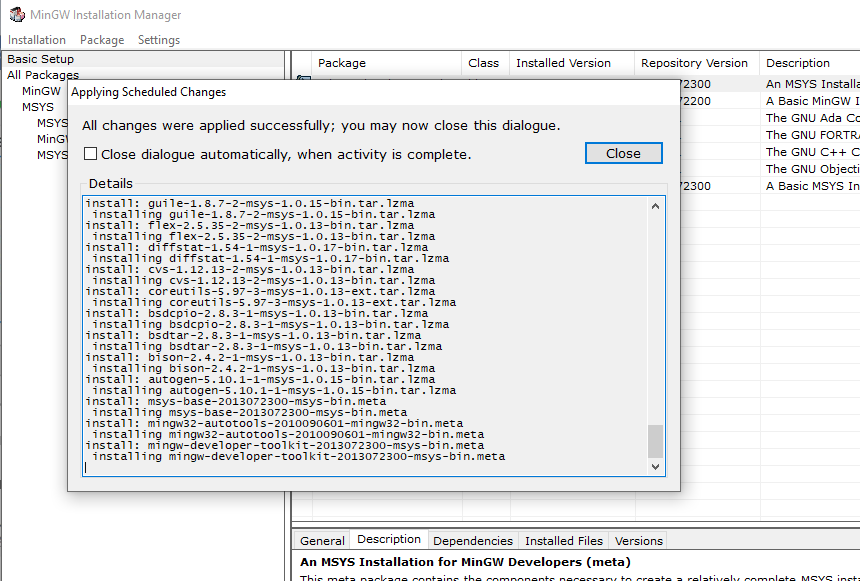
In this window select “Basic Setup”, in the list to the right chose mingw-developer-toolkit and mingw32-basis.

Right-click on this line and select “Mark for Installation”.

Then go to main menu and chose “Apply Changes”.

In the small dialogue that appears press “Apply”.

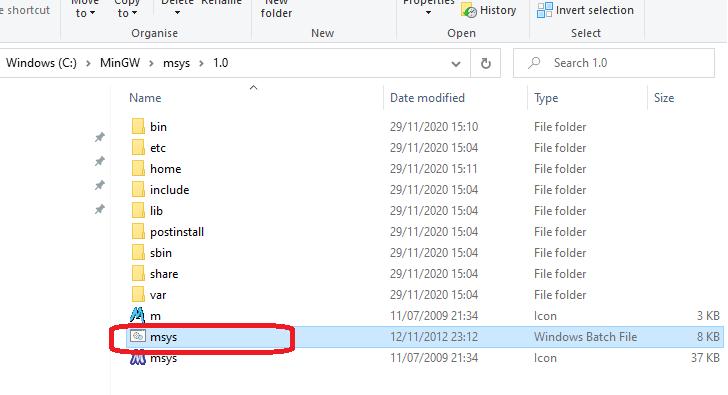




Press Close.

Close this window.

In Explorer:



Start the terminal. You should make a shortcut on your Desktop to this program.

## Verify MinGW Installation

From Windows 10 start the following :



I window like the following should appear.

It is now useful to check if this environment can see any version of GNAT 2020 that we installed earlier. To check this, type:

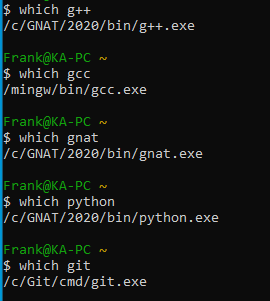
which g++

which gcc

which gnat

which python

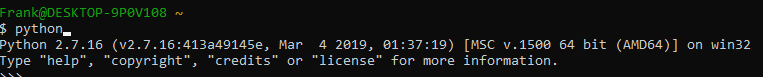
which git



The executables g++, gcc and python should refer to the version in you GNAT compilers folder.

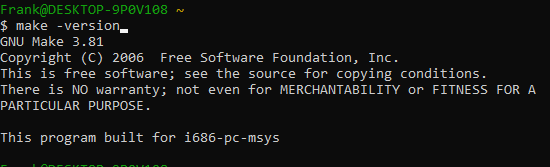
The executable git needs to be installed as a Windows application. (It seems to be hard to install git within mingw-framework. When you install git – install it with a path that doesn not contain any “spaces”. It is a general challenge with this installation process that the **Mingw-tools does not allow spaces in paths**.

Check the version of Python – it needs to be 2.7.



The response from these two, should be referring to /c/GNAT/2020/gcc and gnatmake (see screenshot above). This means that in this environment, as it is configured now, you will execute gcc and gnatmake from this installation. If these tests shows a different result – it might be because there are more than one installation of GNAT 2020 on your PC. You should look into this, by investigating the PATH environment variable in Windows 10, e.g by making sure your last installation occur first among the paths there. If this doesn’t get right you will most probably get error messages and problems that no one really can explain or help you to solve.

## Make 3.81



## Ada-Lua

Go to a folder where you wish to work. In the following we wil use C:\Ada\Git

rm -r ada-lua

git config --global core.autocrlf false

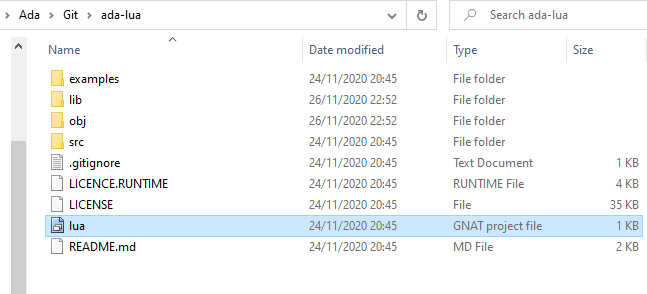
git clone <https://github.com/AdaCore/ada-lua>

In the folder ada-lua, create the folders:

obj

lib

Open the project file in ada-lua project:



Perform “Build All”

Ignore errors about incorrect line terminator.

Close this project.

## PolyORB

Given that your projects are in c:/Ada/Git

Go to the folder of /c/Ada/Git within a MinGW command line window.

cd /c/Ada/Git

git config --global core.autocrlf false

git clone -b 20.2 <https://github.com/AdaCore/PolyORB>

*NB: Per 28 May 2022 we chose tag 20.2 because there is a bug in latest versions in master of this project.*

## Configure

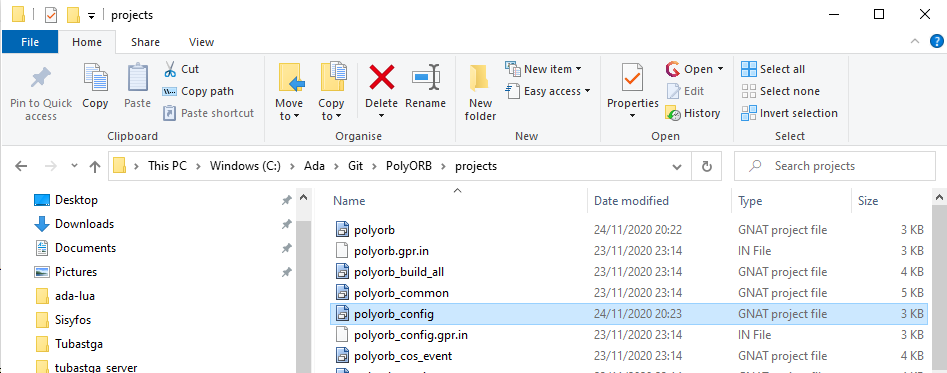
cd /c/Ada/Git/PolyORB

support/reconfig

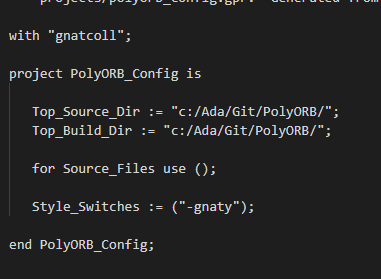
./configure -prefix=c:/GNAT/2020 --with-appli-perso="corba dsa moma" --with-proto-perso="giop soap srp" --with-corba-services="event ir naming notification time" --with-openssl --with-gnatcoll

## PolyORB\_Config.gpr

Open PloyORB\_Config.gpr and edit the two paths there as show below.



Open polyorb\_config.gpr in a text editor and fix the paths in this file:



## Make

You are now perhaps☺ ready to make and install PolyORB.

In MinGW Shell perform:

make top\_builddir=c:/Ada/Git/PolyORB

make install top\_builddir=c:/Ada/Git/PolyORB top\_srcdir=c:/Ada/Git/PolyORB

This process if lasting for several minutes.

## Verify PolyORB

Quite a few executables and libraries are built during this process:

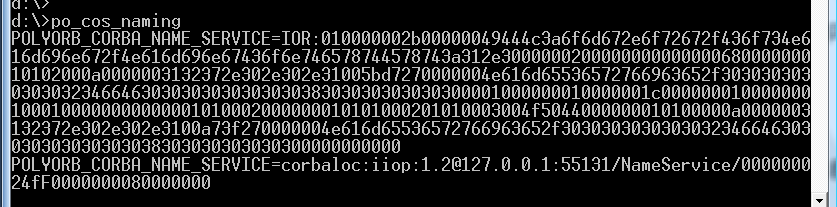
In C:\GNAT\2020\bin you should find:

gnatprfh.exe  
iac.exe  
po\_gnatdist.exe  
po\_cos\_naming.exe  
po\_cos\_naming\_shell.exe  
ir\_ab\_names.exe  
po\_ir.exe  
po\_catref.exe  
po\_createref.exe  
po\_names.exe  
po\_dumpir.exe

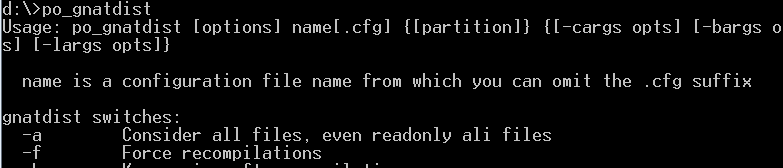
In C:\GNAT\2020\lib you should find a folder called “polyorb”

Plus many more ☺

Try to open a Command Prompt in Windows 7 and type Po\_cos\_naming. You should get the following output. Use CTRL-C to stop this (nothing more will happen at this point).

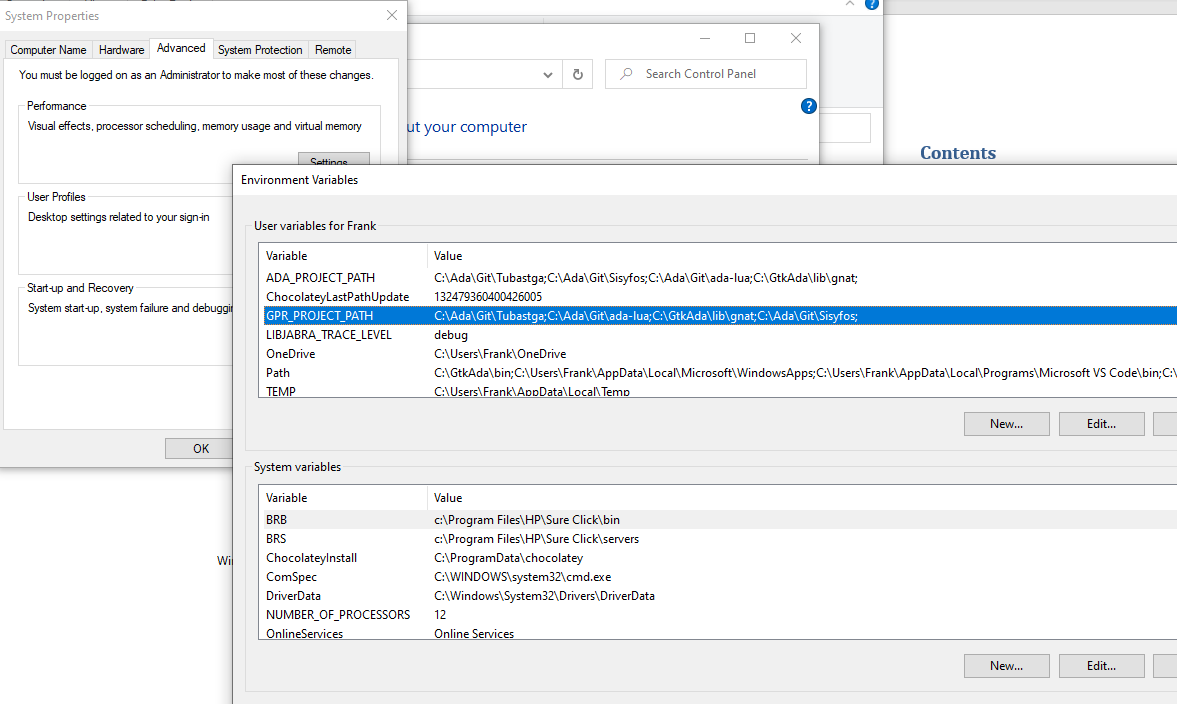


Another test is po\_gnatdist. This is the program that performs the build of a distributed application.



# Windows Environment

Add GPR\_PROJECT\_PATH and ADA\_PROJECT\_PATH to Windows Enviromnet variables.



# Install Sisyfos

## Source Code

git config --global core.autocrlf false

git clone <https://github.com/Medhu/Sisyfos>

# Install Tubastga

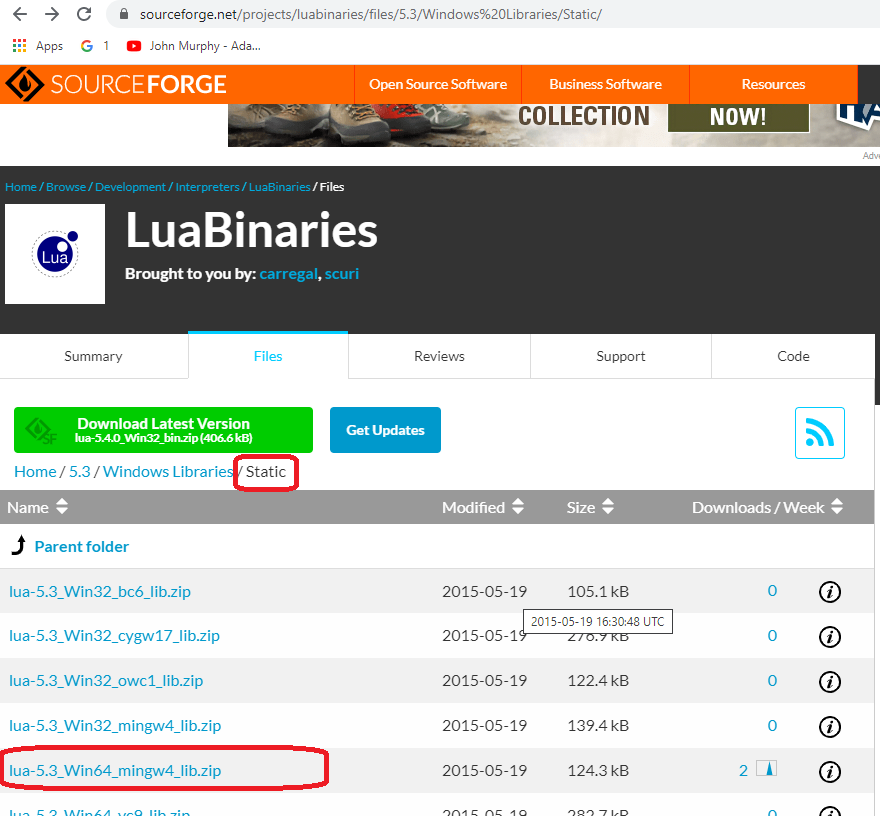
## Source Code

git config --global core.autocrlf false

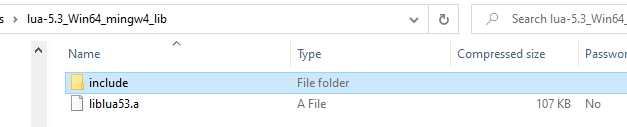
git clone <https://github.com/Medhu/Tubastga>

## Lua Library

Download lua-5.3\_Win64\_mingw4\_lib



Unzip it and pick up the file: liblua53.a



Copy the file into the Tubastga folder and rename it to “**liblua.a**”

# Distributed Hello World

We will create a small distributed solution to “Hello World”. We will have a server that supplies a text string to a client. The client will print this text to the console and the server will print to its console what he sent to the client.

## Source Code

This is the main program for the client. It is calling a function in a package, Server.Get\_Text. This function will return a text string based on the number that is passed into the function.

|  |
| --- |
| with Server;  with Text\_IO; use Text\_IO;  procedure Main\_Client is  Text\_From\_Server : String(1..11);  begin  Text\_From\_Server := Server.Get\_Text(1);  Text\_IO.Put\_Line("Yes, server said:" & Text\_From\_Server);    Text\_From\_Server := Server.Get\_Text(87);  Text\_IO.Put\_Line("Yes, server said:" & Text\_From\_Server);  Text\_From\_Server := Server.Get\_Text(4);  Text\_IO.Put\_Line("Yes, server said:" & Text\_From\_Server);  end Main\_Client; |

The next source code shows the main program for the server. Since our server doesn’t do anything except sitting there, we just present a text to show that it is executing and then delay for 10 seconds.

|  |
| --- |
| with Server;  with Text\_IO;  procedure Main\_Server is  begin  Text\_IO.New\_Line;  Text\_IO.New\_Line;  Text\_IO.Put\_Line("Hello World server by Frank J Jorgensen 2010");  Text\_IO.Put\_Line("Developed by using Ada 2005 / GNAT 2010");  delay 10.0;  end Main\_Server; |

In the server we place an Ada package. The package is specified with “Remote\_Call\_Interface”. This set some limitation to what you can do in the specification part but it allows us to tell the compilator that this package can be accessed by other partitions in the distributed application. In this case it means that the client code (which will be a partition on its own) can call this package.

|  |
| --- |
| package Server is  pragma Remote\_Call\_Interface;  function Get\_Text (pNumber : in Integer) return String;  end Server; |
| with Text\_IO;  package body Server is  function Get\_Text (pNumber : in Integer) return String  is  sRes : String(1..11);  begin  case pNumber is  when 1 =>  sRes := "Hello World";  when 4 =>  sRes := "Another day";  when others =>  sRes := "Don't know ";  end case;  Text\_IO.Put\_Line("Server received pNumber:" & pNumber'Img & " and will return:" & sRes);  return sRes;  end Get\_Text;  end Server; |

To make the compilator build this application distributed as we want it to be we need to create a file to configure what pieces of the source code makes the server and what makes the client.

|  |
| --- |
| configuration helloworld is  pragma Version (False);  pragma Starter (None);  helloworld\_client : Partition;  helloworld\_server : Partition := (Server);  procedure main\_server is in helloworld\_server;  procedure main\_client;  for helloworld\_client'Main use main\_client;  for helloworld\_server'Host use "localhost";  end helloworld; |

To compile and build this, open a Command Prompt and type:

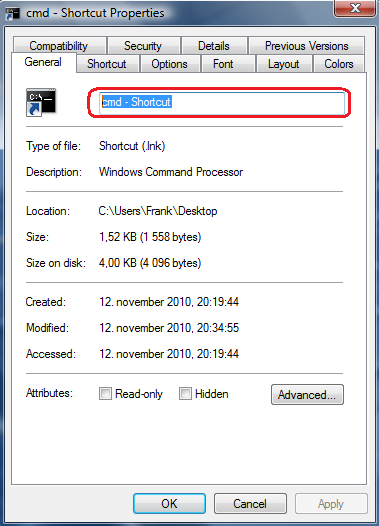
po\_gnatdist helloworld.cfg

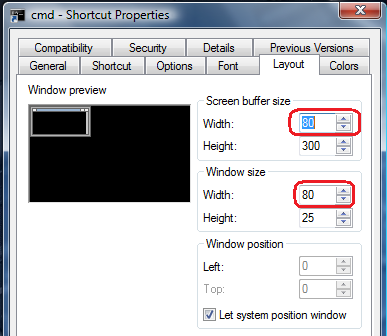
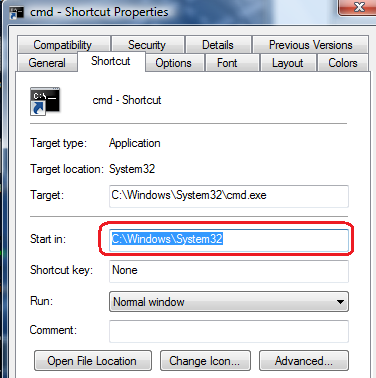
After this is done (takes a few seconds) you will find two executables in your folder helloworld\_client.exe and helloworld\_server.exe.

## Execution on one PC

You need to arrange a bit before executing the application. It is very useful to create some scripts to simplify this work a little. First we will look at the fully manual approach.

1. Create a shortcut for Command Prompt on your desktop
2. Open properties for the shortcut
3. Change the title of the shortcut to “HelloWorld”



1. Go to the Layout-tab  
   
2. Change Screen Buffer Size and Window Size to 150
3. Go to the Shortcut-tab  
   
4. Change the “Start in” to be in the folder where your HelloWorld-project is.
5. From this shortcut, start three Command Prompts. We will call them A, B and C.
6. In A you type po\_cos\_naming
7. Po\_cos\_naming will return an output. This output is needed to make the server and client to get in contact with CORBA process. There are actually two strings in the output that can be used to identify the CORBA process. The simplest is to use corbaloc. Left-click in top-left corned of Command Prompt, chose Edit-Mark from meny there. Mark the string that looks like:  
   POLYORB\_CORBA\_NAME\_SERVICE=corbaloc:iiop:1.2@192.168.0.199:49211/NameService/000000024fF0000000080000000  
   Then press Enter and the string is on clipboard and you can paste it in somewhere.
8. Open a text editor. In this editor, paste the string.
9. In the text editor, construct a statement similar to this, using the string you got from po\_cos\_naming:  
   set POLYORB\_DSA\_NAME\_SERVICE=corbaloc:iiop:1.2@192.168.0.199:49211/NameService/000000024fF0000000080000000
10. Copy and paste this statement into Command Prompt B and C.
11. Now in Commend Prompt B, execute helloworld\_server
12. In Command Prompt C, execute helloworld\_client