Part 1: Installation under Windows

1. Install Visual Studio 2019 Build Tools (free). Current location is here: [https://www.visualstudio.com/thank-you-downloading-visual-studio/?sku=BuildTools HYPERLINK "https://www.visualstudio.com/thank-you-downloading-visual-studio/?sku=BuildTools&rel=15#"& HYPERLINK "https://www.visualstudio.com/thank-you-downloading-visual-studio/?sku=BuildTools&rel=15#"rel=15#](https://www.visualstudio.com/thank-you-downloading-visual-studio/?sku=BuildTools&rel=15)

2. Microsoft Visual C++ 2019 Redistributable Package + latest update available.

Also install the latest Windows SDK version <https://developer.microsoft.com/en-us/windows/downloads/sdk-archive>

4. Install the Microsoft Compute Cluster Pack SDK (this is for distributed execution under MPI).

Add an environment variable named HPCDIR to the installation folder, such that this path is valid: HPCDIR\Include\mpi.h. (In case that the package will be updated in the future and the paths are like Inc\mpi.h or Lib\mpi.h, please rename the folders inside installation package as Include\mpi.h and Lib\).

5. Copy the AGAPIACompiler folder to your disk and add an environment variable with name AGAPIAPATH and as value: yourpath\CompilerAndTools\CompileAndBuildTools\ (for instance, mine is C:\AGAPIA\CompilerAndTools\CompileAndBuildTools\).

6. Set environment variable AGAPIA\_MSBUILDPATH to the location of the MSBuild.exe in your system – the microsoft’s building tool for Visual studio. Mine is at address: C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\MSBuild\Current\Bin\MSBuild.exe

Part 2: How to create a program or use one of the existing examples

1. Copy any examples from “Examples” folder in the distribution to your disk.

2. Edit Def.txt and write there the filenames/directories of headers, libraries, source files (C/C++) you need to build your project, right under the right comment:

E.g:

% // Include files

#include <stdio.h>

% // Additional include directories <-- This has no directories below, every entry is optional.

% // Additional library directories

C:\AdditionallLib

% // Additional linker libs/dll etc

Sobel.dll

% // Additional source files

Image.cpp

3. Modify the agapia.txt file as you wish.

4. Create or modify existing MainInput.txt where you should write the input for the MAIN module. You’ll have to write a value for all the variables in the interfaces a value, otherwise they won’t be initialized. The order should be: variables in the east side then those on the north. E.g. for the examples in section 6:

filenamein images.tga

nrofprocs 3

If there is no input for MAIN, you can leave this file empty.

4. Edit the execution.bat, and modify only after “exectype=type” the following things:

* Use exectype=iterative for iterative building, or exectype=distributed for distributed building.
* Leave Def.txt, “agapia.txt” and MainInput.txt as they are.
* Add the name of your own C/C++ code or resource files here.

E.g. from section 6:

exectype=distributed Def.txt agapia.txt MainInput.txt TGAReader.h TGAReader.cpp images.tga

4. Run execution.bat

If succeeded and there are no compilation errors in your code, it will generate you an exe file in that folder with name AgapiProgram.exe. NOTE: PLEASE ENSURE THAT THERE ARE NO ERRORS (RED TEXT IN THE BUILDING STATUS).

If you chose to run iterative, then you can execute directly the executable. If distributed mode was selected, you'll have to use the MPI Cluster package installed. An example is to call:

mpiexec -n 4 AgapiaProgram.exe and it will run with 4 processes.

IMPORTANT NOTICE/RESTRICTIONS

* Notice that in the current version of compiler, you must write the **name of the modules and input interfaces in the same line**. Also, both **output interfaces must be on the same line**. Also, the name of the modules **must be written with capitals**.

E.g:

module NAME {listen … } {read …}

{

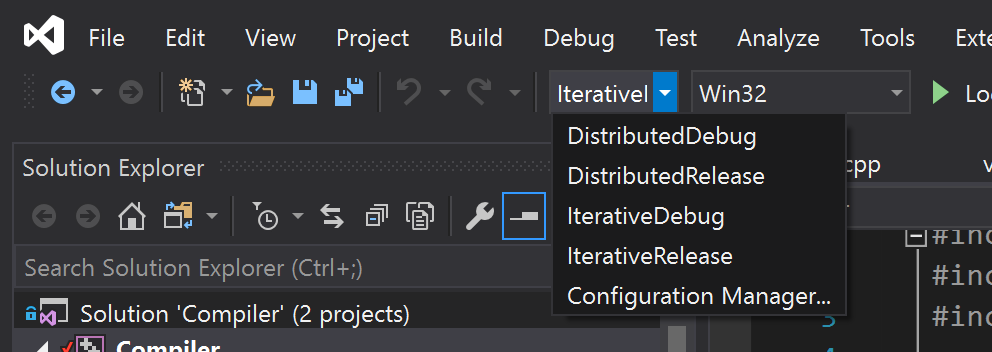
}{speak …} { write …}

* C/C++ specific code errors – easy to fix just see the errors in red when running “execution.bat”. After the bat finishes executing, you’ll have a file create in the current directory named “AgapiaToCCode.cpp”. You can see in this one correctly the line numbers and what errors are.
* Use only “//” comments in C/C++ code
* Avoid using process vector accesses too much (like [vector@[i].nr](mailto:vector@%5Bi%5D.nr)). Accessing such a structure is O(log N), N being the number of items in that structure. So, try to cache the values instead of doing accesses.
* Compiler.vcxproj might be broken by the crash of toolchain if it is modified during the crash. To solve this there is a Compiler.vcxproj –BACKUP in the same folder (search for Compiler folder in the distribution).

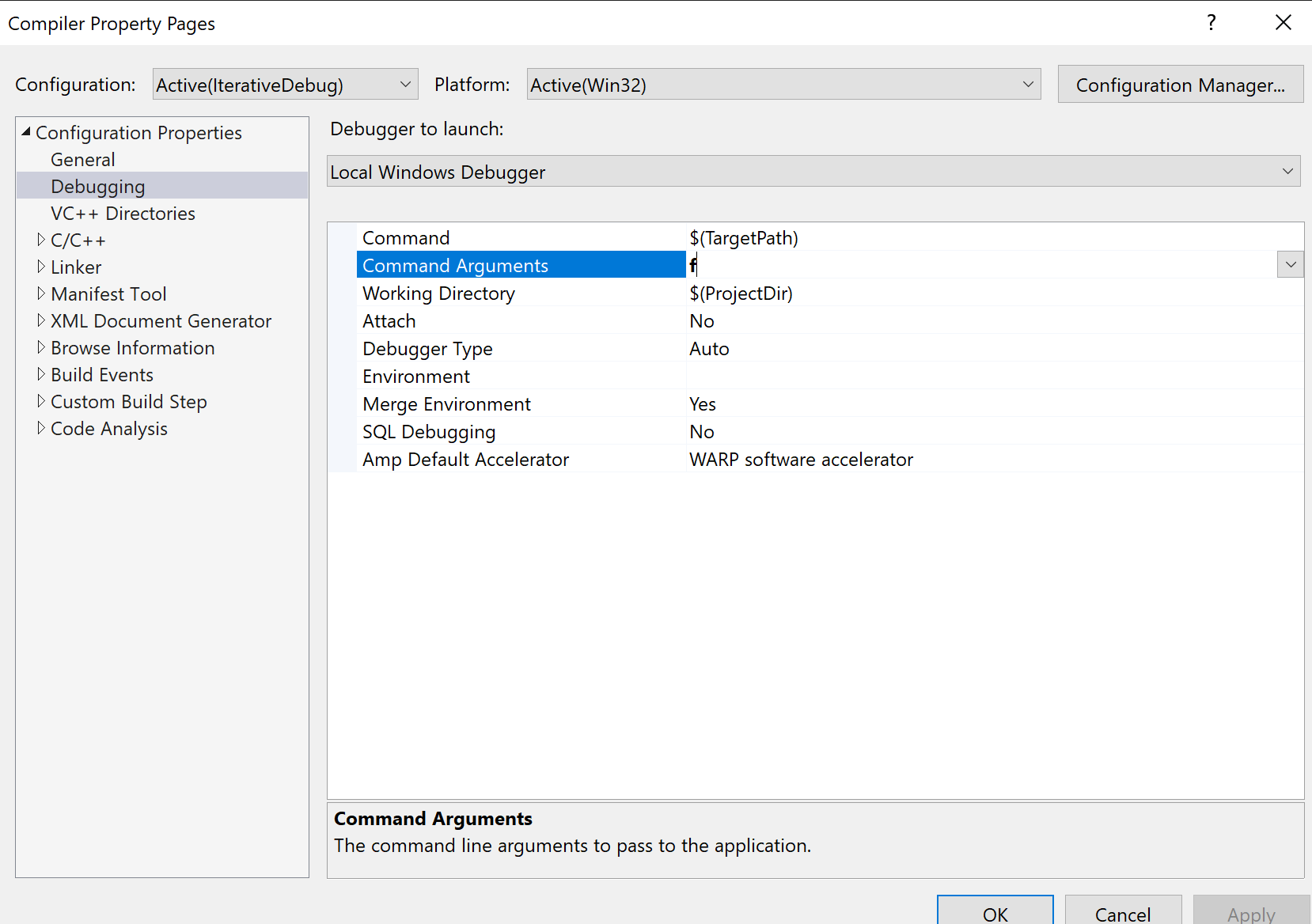
Part 3: How to debug your program with Visual Studio.

The classical method would be to print debug messages at runtime. However there is another method to debug code directly in Visual Studio Professional (support for Express edition will be added too).

* Copy all of your files from your folder to AGAPIAPATH\Compiler (where most of the sources are..).
* Open Compiler.sln and Use “Iterative Debug” as configuration.



* Run once using “g” argument. Then, run without “g” argument and put the breakpoints in file “AgapiaToCCode.cpp” – this will contain the code of atomic modules as C/C++ functions. Or use "f" to always generate code.



* You can now use visual studio to put breakpoints in AgapiaToCCode.cpp and check the flow of your modules.

