**PacketGenerator for the CommProtocol Library**

**Table of Contents**

1. [Description](#Description)
2. [Message](#Message)
   1. [Basics](#MessageBasics)
   2. [Fields](#MessageFields)
   3. [Advanced](#MessageAdvanced)
3. [PacketGen Usage](#PacketGen)
4. [VSBuild](#VSBuild)
   1. [No Diagrams](#VSBuildNoDiag)
   2. [Diagrams](#VSBuildDiag)
5. [UbuntuBuild](#UbuntuBuild)
   1. [No Diagrams](#UbuntuBuildNoDiag)
   2. [Diagrams](#UbuntuBuildDiag)

**Description**

An executable that converts files containing easy-to-make messages into C++ or C# code for packets.

**Writing A Message**

Basics

A message is the input for the PacketGenerator. It has a simple format where you can specify:

* Name of the packet

message PacketName  
{  
 uint8 field;  
   
 uint16 field2 ;  
  
 string fieldDefault = “default” ;  
  
}

* Fields
  + Field type
  + Field name
  + Field default value  
     (optional)

You can write as many messages as you want in your input file, just make sure to close your curly braces. Spacing/new lines don’t matter.

Table of Field Types

|  |  |  |
| --- | --- | --- |
| PacketGen Type | C++ Type | C# Type |
| uint8 | uint8\_t | Byte |
| uint16 | uint16\_t | UInt16 |
| uint32 | uint32\_t | UInt32 |
| uint64 | uint64\_t | UInt64 |
| int8 | int8\_t | SByte |
| int16 | int16\_t | Int16 |
| int32 | int32\_t | Int32 |
| int64 | int64\_t | Int64 |
| float | real32\_t | Single |
| double | real64\_t | Double |
| string | std::string | String |
| vector | std::vector | List |
| linked\_list | std::list | LinkedList |
| set | std::set | SortedSet |
| hash\_set | std::unordered\_set | HashSet |
| map | std::map | SortedDictionary |
| hash\_map | std::unordered\_map | Dictionary |

“Advanced” Messages

WARNING: Types with red text cannot generate diagrams for messages containing fields of this type

There is a little more to messages than what was shown above, but you may not need to use these features quite as often.

* Comments
  + You may want to describe your fields or packets. Doing so is easy.

/\*  
This is a comment about the entire message below.  
It allows for new lines and everything.  
\*/  
message Packet {  
 //This comment describes the “num” field  
 uint8 num;  
   
 /\*This comment describes the “name” field\*/  
 string name = “pack”;  
}

* + Not only will these comments not be considered as part of your code, they will automatically be transferred into your generated C++ or C# files

/\*\*

This is a comment about the entire message below.

It allows for new lines and everything.

…

//This comment describes the “num” field

uint8\_t num;

//This comment describes the “name” field

std::string name;

Packet.h

* Default parameters
  + There are multiple options when it comes to default values for your fields

message Packet {  
 //The default parameter in the constructor will be set to 1  
 uint8 num1 = 1;  
   
 /\*This value must be passed into the constructor, but the constructor called by the  
 Create() method will pass in a common default value (0 in this case)\*/  
 uint8 num2;  
   
 //Value must still be specified in constructor but Create() will pass in 5 to the constructor  
 uint8 num3 ~ 5;  
}

* Containers
  + Using containers is simple but the syntax is different from most languages

message Packet {  
 //A vector of type int16, the type inside the container comes after the variable name  
 vector numbers<int16>;

//A set of string pointers (you can only have pointers inside of containers)  
 set strSet <string\*>;  
  
 //The key is a string and the val is a uint8 pointer  
 hash\_map strNumMap<string, uint8\*>;  
}

* ObjSerializable
  + Packets aren’t the only thing PacketGen can make. It can create ObjSerializables, which can be used as packet fields or be stored in containers.

obj Citizen { //Will be converted into an ObjSerializable  
 string name = “test”; //All fields in objs should be given a default value  
 int8 age = 0;  
 uint32 ssn = 0;  
}  
obj CrimeInfo {  
 uint32 moneyGained = 0;  
 uint8 lawCode = 0;  
}  
message ArrestCitizens {  
 uint32 orderNumber;  
 #Citizen# officer; //A single citizen object named officer, # denotes a custom type  
 vector citizensToArrest<#Citizen#>; //A vector of Citizens  
 map crimes<#CrimeInfo\*#, #Citizen\*#>; /\*maps CrimeInfo pointer to a Citizen pointer. A  
 single individual may be responsible for multiple crimes so we use a pointer\*/  
}

**Executing PacketGen**

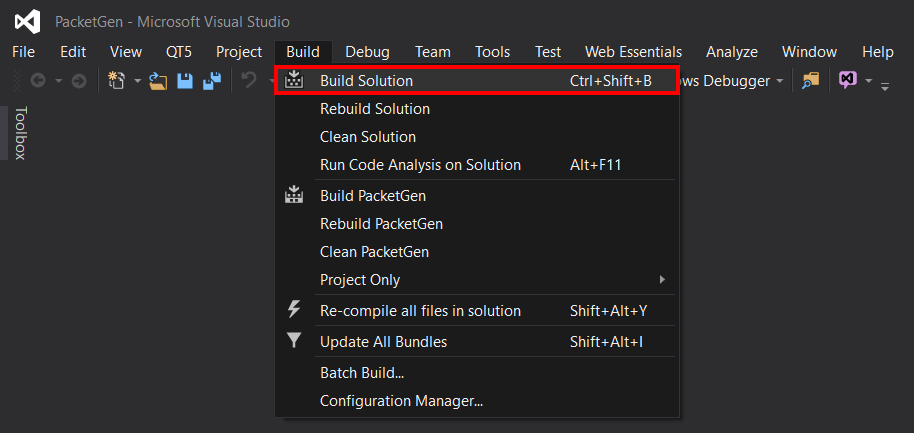
1. Enter the folder CommProto-PacketGen/prebuilt/*your\_os*/  
    (If your OS is not in the prebuilt folder, you will have to build PacketGen. Go to Build  
    Instructions to find out how.)
2. If applicable, open command prompt by holding **Shift** while **Right-Clicking** a blank space in the build folder. Then select Open Command Window Here.
3. Run PacketGen (below are the arguments, some of them are required)

|  |  |  |  |
| --- | --- | --- | --- |
| **ArgKey** | **Description** | **Required/Default** | **Example** |
| I | Path of input file | ***Required*** | -I=“C:/prog/example.ngen” |
| O | Path of output folder | ./ | -O=“C:/packs/” |
| L | Language/form of generated file cpp for C++, cs for C#, diag for diagrams | ***Required*** | -L=cpp |
| E | File ending of generated files | .h //for cpp  .cs //for cs  .png //for diag | -E=hpp |
| H | Command manual |  | -H |

1. Using PacketGen to build an example  
   ./PacketGen -I=“../../examples/NGCP\_Templates.ngen” -O=“./generated/cpp” -L=“cpp”  
   ./PacketGen -I=“../../examples/NGCP\_Templates.ngen” -O=“./generated/cs” -L=“cs”  
   ./PacketGen -I=“../../examples/NGCP\_Templates.ngen” -O=“./generated/diag” -L=“diag”

**Visual Studio PacketGen Build Instructions**

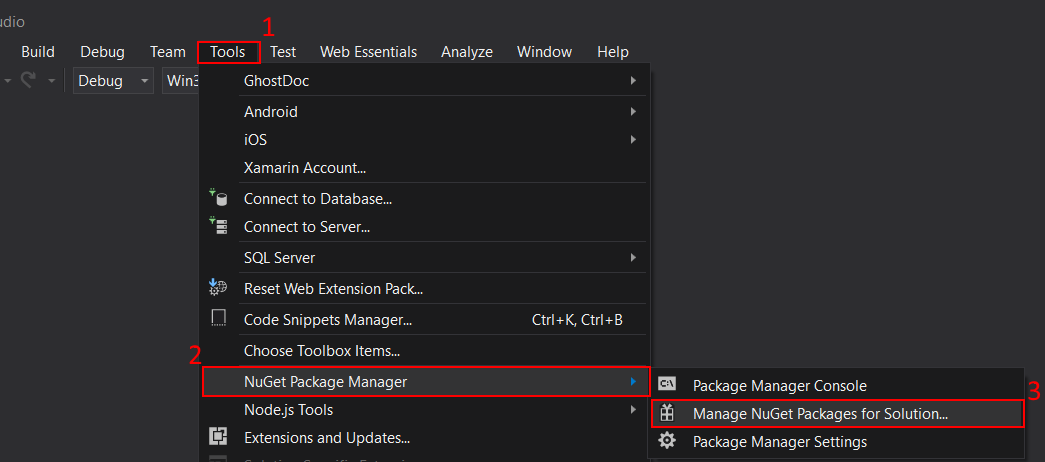
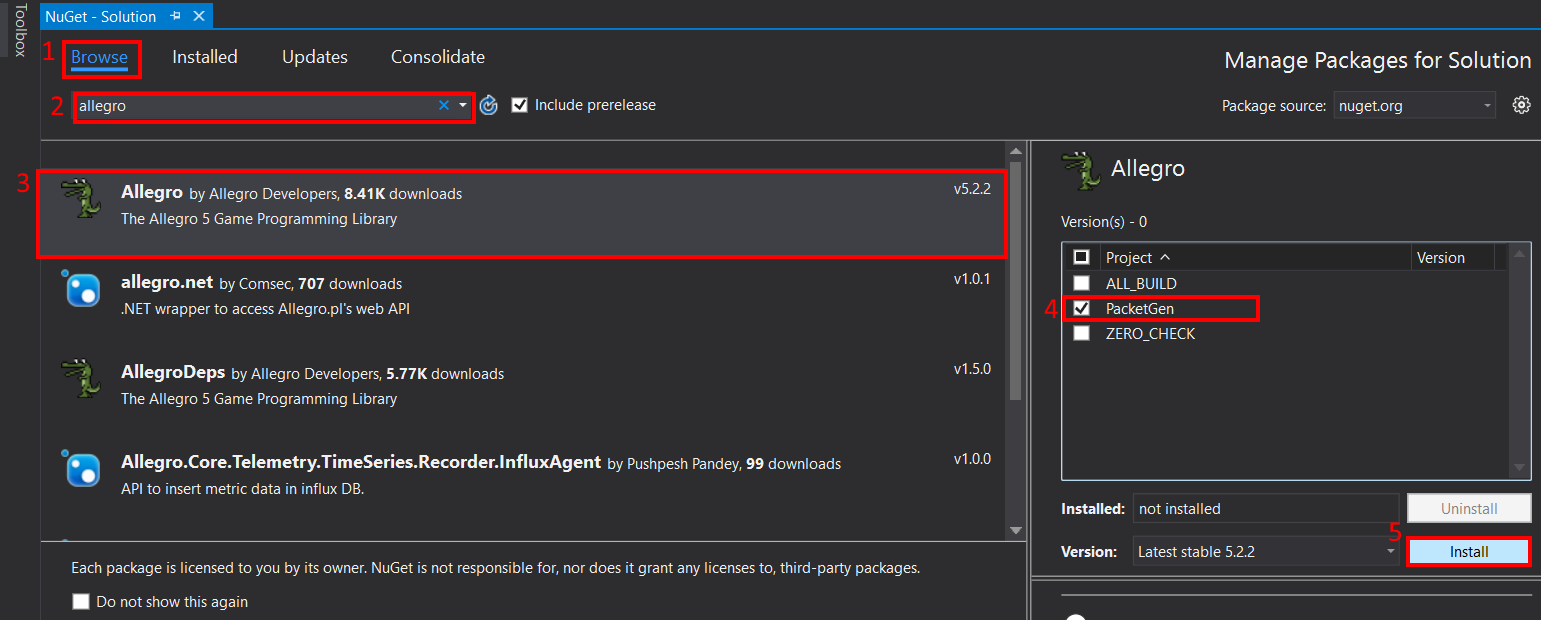
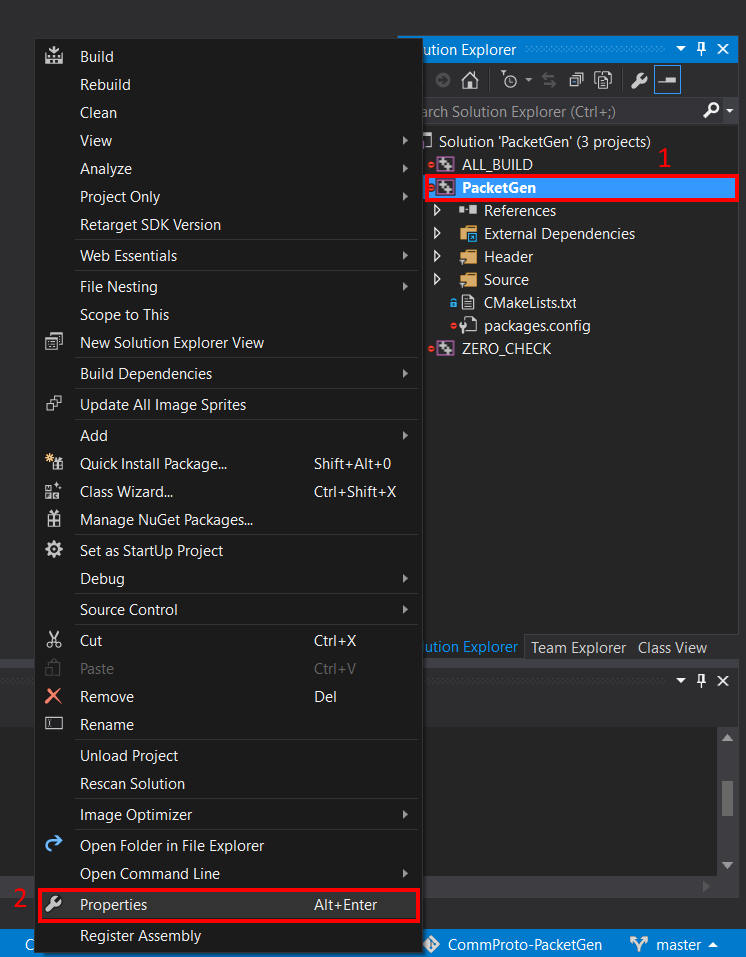
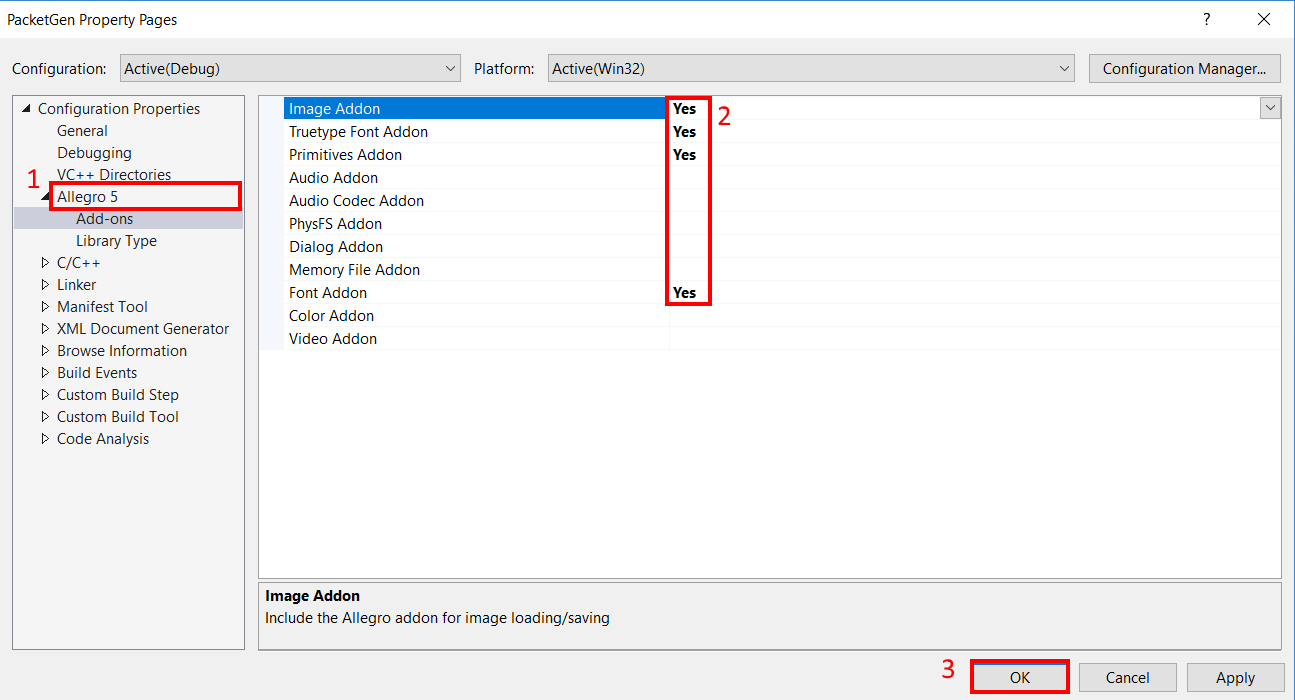
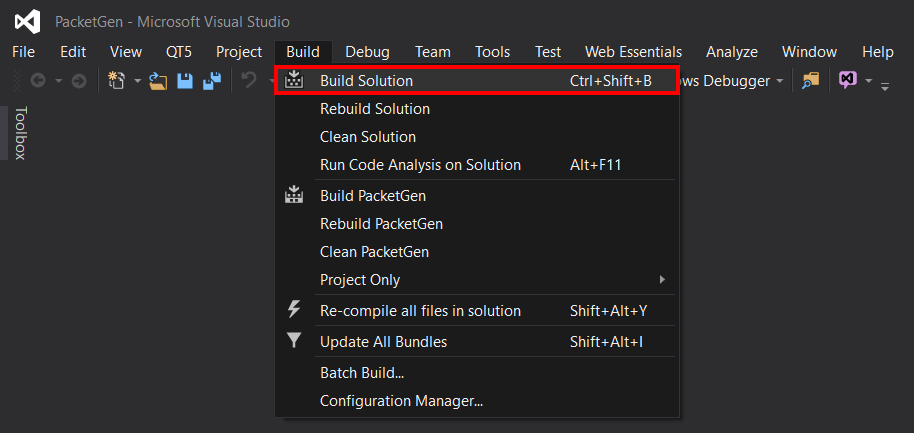
No Diagram Support

1. Create a build folder in CommProto-PacketGen
2. Enter the build folder
3. Open command prompt from file explorer, hold **Shift** while **Right-Clicking** a blank space in the build folder. Then select Open Command Window Here.
4. Run CMake in command prompt: cmake ../
5. Open PacketGen.sln
6. Build the solution by clicking Build in the top left of visual studio, in the drop down menu select Build Solution  
   

\*Before running the solution, you may want to specify some Command Line Arguments <http://stackoverflow.com/questions/298708/debugging-with-command-line-parameters-in-visual-studio>

Diagram Support

\*Steps that are different from no diagram steps will have text colored red

1. Create a build folder in CommProto-PacketGen
2. Enter the build folder
3. Open command prompt from file explorer, hold **Shift** while **Right-Clicking** a blank space in the build folder. Then select Open Command Window Here.
4. Run CMake in command prompt: cmake -Ddiag=ON ../
5. Open PacketGen.sln
6. Open NuGet package manager by clicking Tools in the top left of visual studio, in the drop down menu select NuGet Packet Manager, in the side menu select Manage NuGet Packages for Solution  
   
7. In the NuGet window, select Browse in the top left. In the search bar just below browse, type allegro, click the search result titled Allegro, check the box to the left of PacketGen when selecting what projects to install Allegro on, click Install  
   
8. Click OK in the Review Changes pop up window
9. Access the PacketGen Project properties by right clicking the PacketGen project in the solution explorer, then select Properties  
   
10. In Configuration Properties select Allegro 5. Set Image, Truetype Font, Primitives, and Font addons to Yes. Then click OK.  
    
11. Build the solution by clicking Build in the top left of visual studio, in the drop down menu select Build Solution  
    

\*You may want to set Command Arguments before running

<http://stackoverflow.com/questions/298708/debugging-with-command-line-parameters-in-visual-studio>

**Ubuntu PacketGen Build Instructions**

No Diagram Support

1. Create a build folder in CommProto-PacketGen: mkdir build
2. Enter the build folder: cd build
3. Run CMake in command prompt: cmake ../
4. Build PacketGen: make

Diagram Support

\*Steps that are different from the No Diagram will be red

1. Install the Allegro library package by running these commands:  
   sudo add-apt-repository ppa:allegro/5.2  
   sudo apt-get update

sudo apt-get install liballegro5-dev

1. Create a build folder in CommProto-PacketGen: mkdir build
2. Enter the build folder: cd build
3. Now we can run cmake, but we must tell it that we want to link to allegro and specify where the allegro shared libraries are
   1. cmake
   2. -Ddiag=ON tells cmake we want to enable the diagram feature
   3. -Dalleg=ON indicates we will be linking to allegro
   4. -Dallegdir=/usr/lib/x86\_64-linux-gnu/ the directory of the allegro libraries (by default is set to /usr/lib/x86\_64-linux-gnu/)
   5. -Dalleglib=liballegro.so the path of the main allegro library relative to allegdir (by default is set to liballegro.so)
   6. -Dttflib=liballegro\_ttf.so the path of the Truetype Font Addon relative to allegdir (by default is set to liballegro\_ttf.so)
   7. -Dprimlib=liballegro\_primitives.so the path of the Primitives Addon relative to allegdir (by default is set to liballegro\_primitives.so)
   8. -Dimglib=liballegro\_image.so the path of the Image Addon relative to allegdir (by default is set to liballegro\_image.so)
   9. -Dfontlib=liballegro\_font.so the path of the Font Addon relative to allegdir (by default is set to liballegro\_font.so)  
        
      So your cmake command may be as simple as: cmake -Ddiag=ON -Dalleg=ON ../ if the paths to the allegro libraries match these default values (which they should)  
      or it may be as complex as:  
        
      cmake -Ddiag=ON -Dalleg=ON -Dallegdir=/lib/   
      -Dalleglib=liballeg.so -Dttflib=../liballegro\_ttf.so   
      -Dimglib=imgalleg.so -Dprimlib=../usr/lib/alleg\_prim.so ../
4. Build PacketGen: make

**Other OS PacketGen Build**

* Building the non-diagram version of PacketGen should be easy and similar to Ubuntu instructions
* Building the diagram version requires the Allegro library with the Image, Truetype Font, Primitives, and Font addon. There should be some documentation or existing binaries for Allegro on your OS. After you’ve installed Allegro you can follow the Ubuntu instructions but ignore step #1