C and Cpp - Project discovery

Depending on the source code you have and the way it is provided to you, you should choose a discovery method that best fits your needs. Your choices are as follows, ordered by package completeness (use the first method that matches your situation):

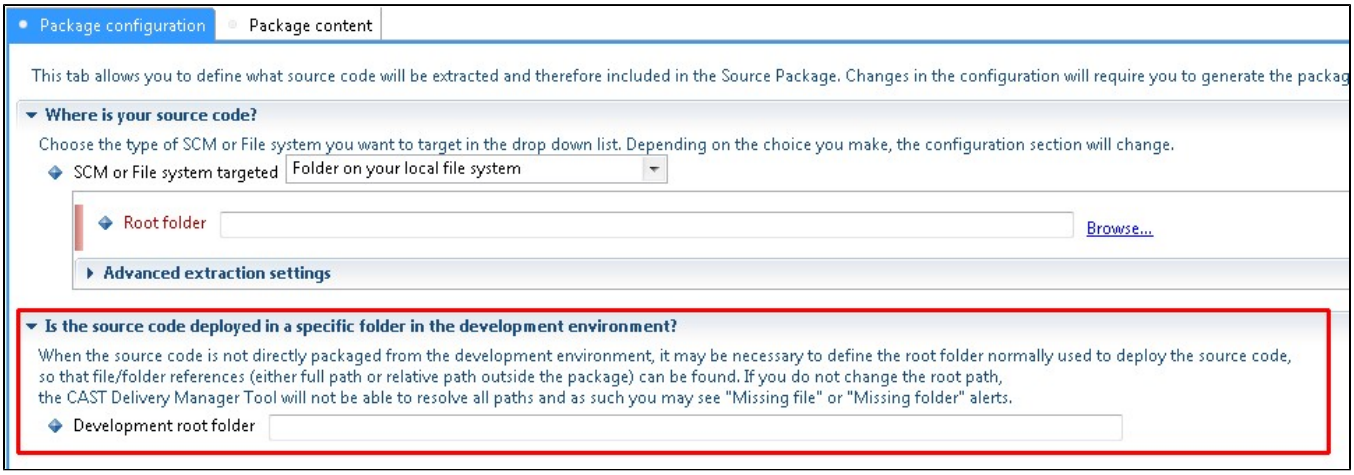
Use the standalone CPP Compilation Database Discoverer	Use the standalone **CPP Compilation Database Discoverer**. This discoverer requires the output of a special tool that is run while compiling the source code and will log all compile operations (i.e. the build log). The build log is then used to determine the projects (and therefore Analysis Units) that are present in your source code.
Use the Microsoft Visual C++ project discoverer	Use the **Microsoft Visual C++ project discoverer**. This discoverer is embedded into AIP Core
Use CMake to generate Visual Studio files or a compilation database	Use **CMake** to generate **Visual Studio files** or a **compilation database**, and fall back to one of the previous methods
Create your own custom discoverer (based on make files)	-
Manually configure the analysis	-

Use the standalone CPP Compilation Database Discoverer

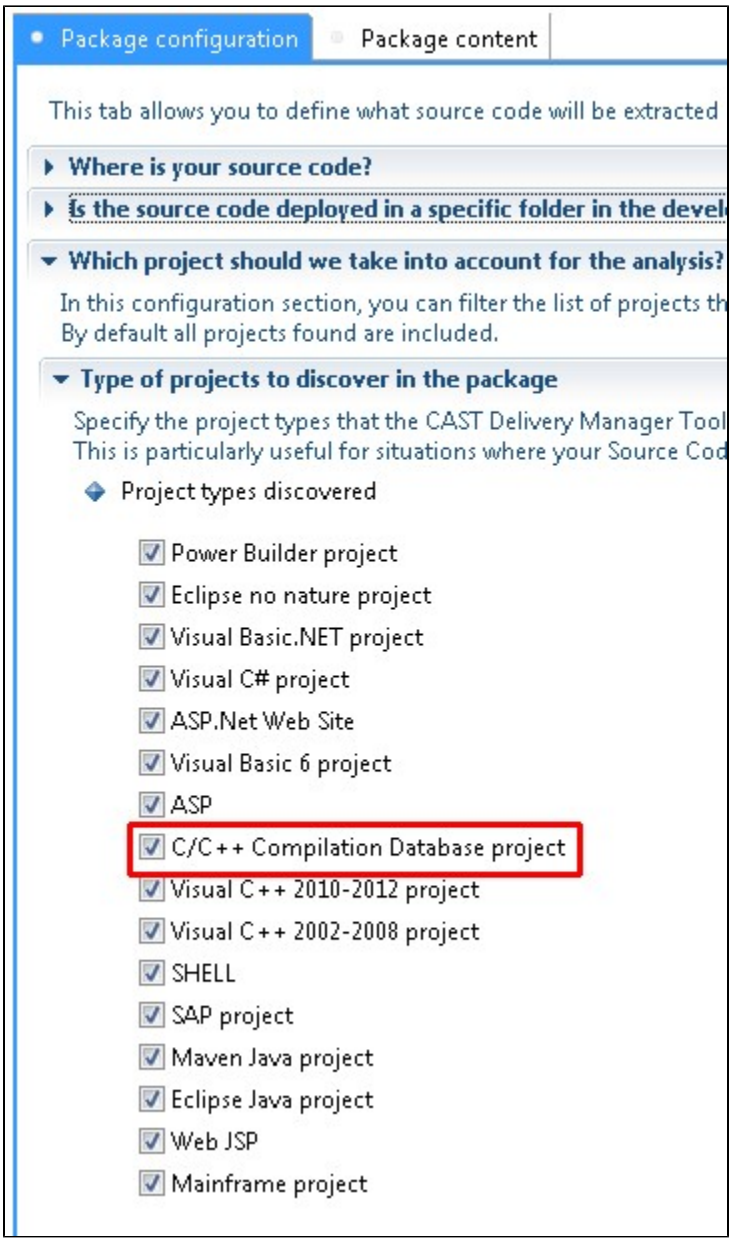
If your source code is designed to compile on **Linux,** you can use a **compilation logging tool** called Scan Build (freely available from CAST - instructions included) while compiling the source code to output two files **compile_commands.json** and **compile_config.json** that will contain all the information necessary to compile your source code. These two Scan Build files can be delivered as input with your source code in the CAST Delivery Manager Tool: as long as you have also installed the CPP Compilation Database Discoverer, then the Scan Build output files will be taken into account when you package the source code and projects will be created based on what is found in the Scan Build files (**compile_commands.json** and **compile_config.json**)

Note the following:

- You may need to create additional source code packages in the CAST Delivery Manager Tool to deliver the header folders
- If you are packaging the code (i.e. running the CAST Delivery Manager Tool) on a machine different from the one where the compilation took place, you will need to make use of the "remapping feature" of the DMT to allow the **CPP Compilation Database Discoverer** to match the folders you are working with to the folders mentioned in the build log. This remapping feature is provided through the option **Is the source code deployed in a specific folder in the development environment**:



- When the **CPP Compilation Database Discoverer** is installed, the following check box will be available in the CAST Delivery Manager Tool - ensure this is ticked so that discovery functions correctly. If the checkbox is not visible, then the extension has either not been installed or has not been installed correctly:



2) Use the Microsoft Visual C++ project discoverers

These discoverers are **embedded into the CAST Delivery Manager Tool** - currently the following versions are supported:

- ***.vcproj files**
  - Visual C++ 2003
  - Visual C++ 2005
  - Visual C++ 2008
- ***.vcxproj files**
  - Visual C++ 2010
  - Visual C++ 2012
  - Visual C++ for versions > 2012 The Analysis Unit generated for this project is mapped to a Visual C++ 2012 environment profile

Visual C++ 2013, 2015, 2017 and 2019 projects, will be discovered as a **Visual C++ 2012** project (i.e. the Analysis Unit generated for this project is mapped to a Visual C++ 2012 environment profile). You can therefore:

- either change the analysis options in the CAST Management Studio so that:
  - "**IDE used for this Analysis Unit**" is set to "**Not Specified**"
  - "**STL Support**" is set to "**Cast emulation**"
- or have **Visual C++ 2012** installed on the analysis machine and analyse the code as a Visual C++ 2012 project

The following is supported:

- Configures a project for each Microsoft Visual Studio 2003 - 2008 project (*.vcproj file) and each Visual Studio 2010/2012 project (*.vcxproj file) identified.
- Handles Inherited Property Sheets and standard Visual Studio macros:
  - $(SolutionDir)
  - $(ProjectName)
  - $(InputName)
  - $(ProjectDir)
  - $(InputDir)
- Only the Release configuration is identified. If there is no Release, then the first configuration is identified instead.

The following information is discovered:

- Source files:
  - resolved in File System
  - matches pre-defined extensions
  - Not excluded from build
- Compile directive
- includes
- macros
- Visual Studio version & MFC usage
- Console

If a property sheet (.vsprops) is referenced from a Visual C++ project (.vcproj or .vcxproj) through the use of an environment variable, no alert is triggered for this variable when discovering the project and no remediation can be added for this alert. Therefore, the property sheet may not be found during the discovery (which leads to badly configured analyses).

Where an environment Variable is used to reference a property sheet, you have several possibilities:

- Before the discovery, you can change the content of your C++ project so that it refers to the property sheet through a relative path, instead of through an Environment Variable. You can write a script to automate this process.
- You can also ignore the missing property sheet during the discovery and modify the settings of the Analysis Units in the CAST Management Studio (mostly add the include paths and macro definitions that would have been provided by the property sheet). A good way to share the additional settings between several Analysis Units is to create a new Environment Profile to handle the required changes.

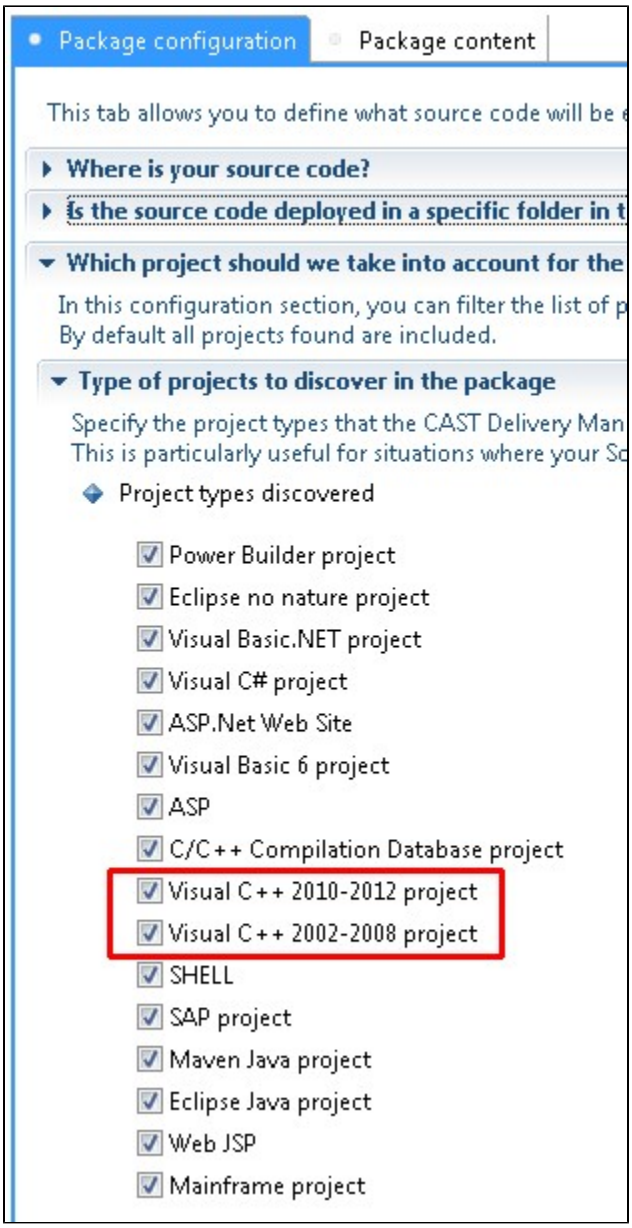
Remember that:

- The discoverer only understands standard projects (no advanced MSBuild features are supported)
- The Visual C++ project files may depend on environment variables - if this is the case, you can add these in the CAST Delivery Manager tool after generating the package (see the remediation for missing variable).
- The appropriate release of **Visual C++ must be installed on the machine** on which the analysis will run (i.e. the machine from which the CAST Management Studio is run) because the C and Cpp Analyzer requires the headers.

When the source code has been delivered and the Analysis Units have been automatically created, CAST highly recommends that an initial analysis in "**test** " mode is run. Since the discoverer does not support advanced MSBuild features, the test analysis may uncover missing headers or macros - if this is the case, then the project may use these features.

In the majority of cases, however, you are likely only to encounter missing headers and since the required include paths will have been automatically discovered from the Visual C++ projects, missing headers is a good indication that the source code files were forgotten and are not part of the source code delivery.

In the CAST Delivery Manager Tool, this discoverer is embedded by default and is materialized with the following two checkboxes - ensure they are ticked so that discovery functions correctly:



3) Use CMake to generate Visual Studio files or a compilation database

CMake project files (CMakeLists.txt) are not directly understood by the C and Cpp Analyzer extension, however:

CMake to Visual C/C++ Studio

Using CMake, you can generate **.vcxproj** files from these projects and thereby make use of the **Microsoft Visual C++ project discoverer** embedded in the CAST Delivery Manager Tool. You can read the CMake documentation for more information about how to do this. A sample command line might look like:

```
mkdir build
cd build
cmake -G "Visual Studio 12 Win64"
```

If you use this method, you will need to "pretend" to compile the code on Windows. If this code has not been designed for that purpose, it may fail to compile. If, when you run the analysis, the log file contains very few errors, you can continue, otherwise, you may want to consult the next section **CMake to build log** (but using the parameters of the Analysis Units created in this process can be a good basis for the configuration).

CMake to build log (compilation database)

Another option with CMake is to generate compilation databases and use them in conjunction with the **CPP Compilation Database Discoverer**. These databases are very similar to what is generated by the Scan Build tool (see previous section **CPP Compilation Database Discoverer**), but they may not contain all the necessary details. We can't guarantee perfect results if you use this method, but it should still provide adequate results.

4) Create your own custom discoverer (based on make files)

If you use a simple project format, you may want to create your own "discoverer" that understands these projects. A cost-effective version of this option would consist of writing a tool that generates .vcxproj files matching your projects, or a JSON file similar to what the build log discoverer generates.

5) Manually configure the analysis

If none of the previous methods are suitable, you can manually create the Analysis Unit in the CAST Management Studio. This option requires a good understanding of C++ to configure include paths and macros. Please ensure you read and understand background information about C++ compilation before you decide to use this method.

CAST highly recommends that if you need to manually create the Analysis Units, that you get help from someone who knows how the project you want to analyze is built. A C++ projects often consists of different libraries, each one with its own specific compilation options and therefore organizing this can proove costly and complex. To help with this process, CAST AIP includes a **test analysis mode**. This analysis is faster than a full analysis, and contains a report about things that went wrong and information about how the configuration of the analysis could be improved. CAST recommends that you obtain a "green light" from the test analysis mode before trying a full analysis.

The first step in this manual process is to make sure that you have all the headers available. Either because they are part of the delivery, or because they are available somewhere on the analysis machine (you can get them by installing a compiler, copying the headers, downloading a third party library etc.). Then the process of configuring an analysis unit will be iterative:

- Get all required files in the delivery and run the packaging and delivery actions in the CAST Delivery Manager Tool
- Create the correct Analysis Units that you require (usually one per folder, or one per project file). The use of the C and Cpp File Discoverer may be of help for this step.
- Add include paths (possibly helped by reading project files, or a real compilation log)
- Add macros (possibly helped by reading project files, or a real compilation log)
- Run the analysis (in test mode), examine the log
- Repeat the process until the log is clean.

How to read a compilation log

If you run a real compilation (or get a real compilation log), you may be able to see the options you should define in your Analysis Units. Macros are usually defined with -D and include paths with -I. If the compilation logdoes not show that, you may try to increase its verbosity or use the -N option of make to get a debug run of the compilation.

Where should I start? Macros or include paths?

There is no easy answer. Usually, header files contain the definition of many, many macros. It is therefore better to start and correct the include paths first. However, in some cases, the choice of header files that will be included itself depends on a macro. So you should define this macro correctly before trying to solve problems with this header.

Normally, the only macros that you should define in an Analysis Unit configuration are macros related to the configuration. For instance, macros such as WIN32, UNIX, GCC_VERSION, USE_SSL can be correct. Macros like ANA_WIZARDDLG_EXPORT, CHECK, XXX_H are suspect. Macros that match a keyword such as **int**, **void** etc. are **always** wrong.

Almost all the macros that you must define are used inside a **#if** or **#ifdef** directive. You should almost never directly define macros that appear directly in normal source code. Those are usually defined in headers files, sometimes controlled by configuration macros (the ones that may need to be defined).

If you find yourself having to define more than two dozen macros, you probably have another problem: the macros are not set to the correct values, some include paths are still missing etc. We have seen users defining hundreds of macros: this will never lead to a good analysis and each added macro will only decrease the analysis quality and increase the risk of crashes. Very often, removing all of the macros and adding just one can solve all macro related issues. The art is to find which macro to add.