Copilot

Certainly! Let's delve into the concept of **CMake generators**:

1. **What Is a CMake Generator?**
    - A **CMake generator** is responsible for writing input files that configure a **native build system**. These input files guide the build process by specifying how to compile and link source code files.
    - CMake itself doesn't directly compile or link source files; instead, it generates configuration files for the chosen build system.
2. **Build Systems and Generators:**
    - A **build system** encompasses a set of tools used for compiling and linking source code. It may also involve auxiliary tools during the build process.
    - CMake generates files and folders specific to the chosen build system, allowing the building of source files referenced in `CMakeLists.txt` and supporting `.cmake` files.
    - Different build systems may be available on a system (e.g., Visual Studio, MinGW, Make, Ninja).
3. **Types of CMake Generators:**
    - **Command-Line Build Tool Generators**:
        - These generators are for command-line build tools like **Make** and **Ninja**.
        - Examples include Makefile generators for various platforms (Borland, MSYS, MinGW, Unix, etc.).
    - **IDE Build Tool Generators**:
        - These generators are for Integrated Development Environments (IDEs) that include their own compilers.
        - Examples include Visual Studio (multiple versions) and Xcode.
    - **Extra Generators**:
        - These create configurations for alternative IDE tools (e.g., CodeBlocks, CodeLite, Eclipse).
        - Extra generators must be used alongside an IDE or command-line generator .
4. **Input Files:**
    - When you have a set of C++ files in your project, these files are indeed part of the input.
    - The generator prepares input files (e.g., makefiles) for the native build system (e.g., Make) to interpret and execute the build process.

Remember that CMake's flexibility allows you to choose the most suitable generator based on your project and platform! 😊