

# CDT Managed Build System

Position Paper for Eclipse Languages Symposium  
Mikhail Sennikovsky, October 19, 2005

This paper presents the CDT Managed Build System that is used for automatic buildfile generation in CDT, and the ability to use this system for any kind of compiled languages.

There are two types of builders available in CDT. Both builders use an external builder utility and external tools (e.g. compiler, linker, etc.) for building the project.

1. “Standard Make”. When using the standard make builder (standard make projects), a user has to write the buildfiles manually.
2. “Managed Builder” When using the managed builder (managed build projects), the buildfiles are automatically generated by the managed build system.

There are several major advantages of the Managed Build System (MBS):

1. Automatic buildfiles generation. MBS fully automates the buildfile generation, so a user does not have to deal with the buildfile syntax, build settings are controlled via UI. Also a user does not have to modify buildfiles each time the set of project resources is changed (e.g. files/directories are added/removed/renamed), MBS regenerates the buildfiles when necessary.
2. Providing build information. It often happens that the information specified for the tool in the command line, is also needed by other parts of the CDT. E.g. a compiler may use a built-in set of include directories and macros that are also needed by the parser. MBS can automatically calculate this information based upon the current managed build settings and provide this info to the rest of the CDT.
3. MBS allows ISVs to define a set of very useful call-backs, e.g. an ISV can provide an additional environment to be used when launching the builder process. This allows ISVs, e.g. to automatically add the tool-chain binaries paths to the PATH environment, so users will not have to manually modify the environment in order to make the project build correctly.

The managed build system generates the buildfiles and automatically obtains the build information based upon the following info:

1. Tool-chain/tool/builder definitions provided by an ISV used for the given project
2. User-modified/specified settings
3. The set of project resources.

## Tool-chain/tool/builder definitions

An ISV can integrate the tool-chains by contributing to the “org.eclipse.cdt.managedbuilder.core.buildDefinitions” extension point. This extension point provides the mechanism to define and describe in a special grammar external tools that should be used for building the project.

Here is a short description of the main elements used in the MBS grammar:

MBS has a notion of a *tool-chain*. A tool-chain is a set of tools and a builder used for building the project. MBS tool-chain definition specifies tools and builder definitions used for building.

The *tool* definition represents an external tool and the way it is used while building. The definition contains

- A tool command – the command that is used for launching the tool
- A set of definitions of tool options. Each *option* definition specifies the type of the value the option represents, the way the value is represented in the tool invocation command, the default option value and the human readable name for the option
- A set of tool *inputs*, that describe what types of resources the given tool can build
- A set of tool *outputs*, that describe what types of resources the given tool generates

The *builder* definition represents a builder to be used for building. It defines the builder command and arguments to be used for launching the builder and the buildfile generator that generates buildfiles based upon the project build info.

Also *project types* can be defined. The project type definition groups the set of *configurations* (e.g. “Release”, “Debug”) that could be used for the project. Each configuration specifies the build artifacts produced when the configuration is built, the tool-chain used for building the project, the default settings for the tools options and other configuration-specific information.

When the managed project is being created, a user is asked to choose the project type and a set of configurations to be associated with the project. So, each managed build project has the Managed build information associated with it.

### **Editing the build settings**

As was mentioned previously, MBS provides UI that is used for editing the build settings. MBS implements the project and the file property pages used for editing project-wide and resource-specific build settings respectively.

MBS generates the appropriate UI contents based upon the tool-chain information provided by an ISV.

A user is able to modify the build settings, which includes:

- Tool settings: tool command and tool option values for each tool that will be used while building. MBS generates the appropriate UI based on the tool/option definitions provided by an ISV. For each tool’s option MBS generates a label that describes the given option and the control used for viewing/editing the option value. The type of this control depends on the value type the option could contain, e.g. if the option definition specifies that the option represents the list of include directories, the list with button bar is created that allows the user to view/add/remove the list of directories, if an option takes a value from a pre-determined range of choices, a combo box is created that displays a human readable representation of each value (provided by an ISV) and allows user to choose the value to be used, if an option accepts any kind of user-specified string value, an edit-box is created, etc.
- Build settings: builder command and arguments, etc.
- Custom build steps settings. This allows user to define the set of commands to be launched before or after the project build and also provides the capability to define tools other than those specified by an ISV, that are to be used for building some set of project resources.
- Builder process environment
- Other build settings that include build macro settings, error parser settings, binary parser settings, etc.

## **Building the project**

When the managed project build is requested, the build-file generator is invoked that generates the appropriate buildfiles for the project resources based upon the project managed build info. MBS provides the buildfile generator that generates buildfiles for the gnu make, though there are no restrictions on the builder to be used, so an ISV can provide a custom buildfile generator in order to allow using any custom builder.

Having the set of project resources and associated build settings, a buildfile generator is capable of generating the buildfile. The generator detects the tool command to be used for building the given project resource based upon the tool inputs/outputs information and tool option settings.

After the necessary buildfiles are generated the external builder is invoked to build the project. User-defined and ISV-defined environment is included in the set of environment variables used for the builder process.

## **Providing build information to other parts of CDT**

MBS defines some special option types that can be used to specify options that represent includes, libraries, objects and preprocessor symbols for the tool.

Based on the values of those options, MBS automatically calculates the build info and passes it to the rest of the CDT.

## **MBS C/C++-specific features, using MBS for building non-C/C++**

MBS contains a few C/C++-specific features that actually don't put any restrictions on using MBS with other languages.

- MBS grammar defines some C/C++-specific option types, that are used to represent some C/C++-specific data (include paths, defined preprocessor symbols). These options actually do not specify any special way of handling them in the buildfile generation, but are used to allow MBS to provide the build information to the rest of the CDT, as described above. So those options could be simply omitted in case of their inapplicability for the given tool/compiler, with no restrictions to the buildfile generation.
- The other MBS C/C++-specific feature is the project nature filter that the tool definition can contain. A tool definition can specify a filter for the nature of the project the tool is to be used for. The nature that could be specified currently is 'cnature' to represent the C project nature, 'ccnature' to represent the C++ project nature and 'both' to specify that there is no nature filter. The inability to specify project nature filters other than C or C++ does not really bring any problems in using MBS with non-C/C++ languages. In general the decision of what tool to use for building the given resource is made based upon the resource type and a set of input types that the tool accepts, so the decision is made based upon the resource type rather than upon the project nature. This concept allows implementing mixed-language projects that could contain sources of any language types simultaneously; this is actually one of the main goals for the compiled language IDE. One of the future plans of the CDT is to allow mixed projects that contain both C and C++ sources that will use the above concept. So the project nature filter may be not needed at all in the future.

As follows from the above description, the managed build system puts no restrictions on the types of the tool-chains/tools/builders to be used for building, so MBS could be used for integrating any types of compilers/linkers/builders to allow building any types of sources. For example, the Photran project expects to support managed build projects in their next release.