CDT Core Build System

Rethinking, Simplifying, Flexible

Doug Schaefer

Blackberry QNX, CDT Project Lead

EclipseCon Europe 2017

The Core Build Manifesto

- You can not separate build and launch. You build because you are going to launch somewhere. If you know where you are going to launch and what tools you are going to use (i.e. the launch mode), you can automatically figure out how to build and can hide all or at least a lot of that from the user.

Relation with Launch Bar

- The Launch Bar will auto generate Launch Configurations that match the Triple (mode, descriptor, target).

- For CDT projects, can we also generate Build Configurations?

Be More Flexible by Doing Less

- Allow for CDT builds to be managed externally
  - Qt's qmake, CMake, Arduino's build system
- Peel away the onion that is managed build
  - What's the minimal API necessary
- Reuse IBuildConfiguration from the Platform
  - Speaking of minimal…

The Minimal API

- IncrementalProjectBuilder
  - Build and Clean.

- IScannerInfo
  - Feed include paths, defined symbols, macro and include files to CDT's parsers
  - Per IResource, i.e. source file

- IConsoleParser, IMarker
  - Creating error markers for build errors detected on the console

The Minimal API

- IEnvironmentVariable
  - Contribute to the environment variables used to invoke build tools

- IBinary, Binary Parser
  - Select binary parser to find IBinary's
  - And determine the IBinary to be used when launching

The Core Build Core Interfaces

- ICBuildConfiguration
  - Implements core interfaces in support of a given build system
  - E.g. CMake, Qt’s qmake, Arduino

- IToolChain
  - Functionality common to a given toolchain and may be used with many build systems
  - E.g. GCC and compatibles, MSVC
  - Build Config will delegate to toolchain where it makes more sense

Relation to Managed Build

- Nothing other than using similar names
- In theory, should be able to flush MBS
- But in practice, we do need an MBS-like ICBuildConfiguration
  - That provides UI for build settings and generates build based on them
  - But only that, don’t force other build systems to sit on top
    - And can we please rescue Makefile Builds from MBS?

CBuilder

- IncrementalProjectBuilder for Core Build projects

- Sets up build console
- Gets active IBuildConfiguration, adapts to ICBuildConfiguration
- Calls build or clean

IToolChain

- Provides
  - Binary Parser Id
  - Error Parser Ids
  - IScannerInfo
  - Environment Variables

- Identified using bag of properties
  - Standard ones: OS, Arch
  - But extendible, e.g. OS Release

ICBuildConfiguration

- Main job is to implement build and clean
- Associated with a single IToolChain
  - Delegates to its services where appropriate
- Invokes build tools as it needs to do the build or clean
  - Hooks up those tools to console and error parsers
  - Does build output parsing
- Manages IMarker’s
- Finds IBinary[] from result of build, used by launch
- Caches IScannerInfo

How are ICBuildConfigurations Created?

- By ICBuildConfigurationProviders
  - Registered against Natures
- Originally created by Targetted Launch Config Delegates
  - Finds project from launch config
  - Take properties from Launch target and finds matching IToolChain
  - Calls build config manager to create from project, toolchain, and launch mode
- Now also created by CoreBuildLaunchBarTracker
  - Launch Bar listener tracking active mode, descriptor, and target
  - Adapts descriptor to project, target to toolchain, mode is mode
  - Finds or creates build config and sets it active on project

UI

- Originally, wasn’t any. Everything automatic.
  - Toolchains discovered by toolchain providers
- Added Build Settings tab group for Descriptor Editor
  - Stores properties as Launch Config Attributes
  - Copies them over to build config when making it active
- Added UI to add ToolChains and set properties
  - Added UI to order them so that when multiple toolchains match properties we can pick the preferred one
- Added UI to Build Settings tab to select toolchain to use for a given config

Next Steps – “Standard Make”

- More accurately - freeform build systems
- User specifies command to run for build and clean
  - UI similar to External Tools?
- Still needs to specify a toolchain so we can set up things
  - Paths for running commands
  - Scanner Info Provider
- Build output parser already to go (used by CMake)
  - Just need to send lines from console to it

TODO

- Build Console colorization and linking isn’t working
- UI for changing build settings from project properties
  - Set Active Build Config?

Issues

- What if you want multiple build systems per project?
  - Often see projects that support CMake, Autotools, Makefile
- What if the Launch Bar is not available?
  - Need to find an alternative UI to create and select build configs
- What other build systems should we add?
  - Autotools has been suggested
  - Would be great to have MBS

Issues

- How do we do remote builds?
  - Build tools run on a different host, could be target or third system
  - E.g. Docker, WSL, remote machines (cloud or embedded)
  - IRemoteConnection has great support for invoking remote tools
  - How do we run indexer? Jeff has this solved in his Docker support

Questions, Comments, Suggestions?