Building with MPC

Charles Calkins
Principal Software Engineer
Object Computing, Inc.
calkinsc@ociweb.com

OCI's Business Model

- Training
  - Offer over 60 1- to 4-day classes
    - OO design, languages, middleware, web tech
- Custom software development
  - Build proprietary applications for customers
    - What I do
- Support of Open Source software
  - TAO, OpenDDS, QuickFIX/FAST, etc.
    - These are multi-platform – need a build system

Makefile, Project and Workspace Creator (MPC)

- Started in 2002
- Motivation was to build TAO on multiple architectures and compilers
  - Windows, Linux, …
  - MS Visual Studio, gcc, Borland, …
- Wanted a uniform way to express projects
  - …rather than creating Makefiles, plus a Visual Studio solution, plus… by hand

MPC Structure

- Portable – written in Perl
- Free – is Open Source, BSD-style license
- Define a *project* in a **`.mpc`** file
  - Represents a compiled module
    - Executable or library, generally
- Define a *workspace* in a **`.mwc`** file
  - Represents a collection of projects
- Generate build files with:
  - **`mwc.pl -type`** <type> <file>**`.mwc`**

Hello World Executable, C++

- Suppose this directory structure:

```
Hello\
  HelloCPP\
     Hello.cpp
```

- Want to build under Linux, so create a Makefile

HelloCPP Makefile

```
BIN = HelloCPP
OBJS = Hello.o
CXX = g++
CPPFLAGS = -g
LD = $(CXX) $(CPPFLAGS)
LDLIBS = -ldl -lpthread

all: $(BIN)

$(BIN): $(OBJS)
  $(LD) $(OBJS) $(LDLIBS) -o $@

Hello.o: Hello.cpp
  $(CXX) $(CPPFLAGS) -c -o $@ Hello.cpp
```

HelloCPP in Visual Studio 2010

- Want to also build under Visual Studio
  - File → New Project…
  - Choose C++ → Win32 Console Application
  - Name it **`HelloCPP`**
  - Application Settings → Empty project toggle
  - Right click on the HelloCPP project, Add → Existing Item…
  - Select **`Hello.cpp`**

Time Passes…

- Already lots of error-prone steps, but over time the task gets bigger…
  - Add more projects
  - Use new or upgraded libraries
    - Include and link paths, and library names, can change
  - Change compiler warning level across all projects
  - Change output directory of all projects
  - … and lots more
- After modifying all of the Makefiles, now do the Visual Studio projects, too
- Cumbersome, and easy to get wrong
  - e.g., offshored developers changed build paths to their system, checked in code to CVS - failed for us the next morning!

Scale

- With only a few projects, changes by hand are possible, but in the real world…
  - Flowlink (what I do)
    - 155 projects in the workspace
    - 700,000 lines of code
  - TAO/CIAO/DAnCE/OpenDDS
    - 266 projects in the workspace
    - 1.2 million lines of code

Use MPC Instead

```
Hello\
  HelloCPP\
     Hello.cpp
     HelloCPP.mpc   ← MPC file
  Hello.mwc  ← MWC file
```

MPC Files

In **`Hello.mwc`**:

```
workspace {
}
```

In **`HelloCPP.mpc`**:

```
project {
}
```

MPC Files – For Defaults, That's All

- For the workspace:
  - Subdirectories searched for MPC files to add
- For the project:
  - All CPP files in the current directory automatically included
  - Source searched for **`main()`**, if found then it is an executable, else it is a library project
  - Project name is set to the name of the MPC file
  - Default behavior can be overridden if needed, but projects often need little to nothing in each MPC file

Typical MPC Options

- **`exename =`** <n>, **`libname =`** <n>
  - Set project name to override default
- **`Source_Files {}`**
  - Specify files to compile
- **`after +=`** <n>
  - Establish dependency
    - build current project after specified project
- **`libs +=`** <n>
  - Libraries to link to

Why MPC is Better

- MPC adds project inheritance
  - Workspace composition, too
  - Can't do this with Makefiles or Visual Studio
- Create a *base project* in an **`.mpb`** file

In **`mybase.mpb`**:

```
project { … options … }
```

In **`myproject.mpc`**:

```
project: mybase { … options … }
```

Project Inheritance

- Set options in base projects and inherit from them (base projects can even inherit from other base projects)
- Options set only in one place, so only one place to change when they need modification
- For instance, a base project could represent a particular set of libraries – only need to update the one place when version numbers change
  - e.g. Apache POI for MS document management, as used by Flowlink

POIBase.mpb

```
project {
  expand(FL_ROOT) {
    $FL_ROOT  // base off of environment variable
  }           // can rebase without problem

  libpaths += $(FL_ROOT)\bin\ikvm\poi-3.8
  libpaths += $(FL_ROOT)\bin\ikvm\ikvm-
  7.1.4532.2\bin
  lit_libs += $(FL_ROOT)\bin\ikvm\poi-3.8\poi-
  3.8.dll
  lit_libs += $(FL_ROOT)\bin\ikvm\ikvm-
  7.1.4532.2\bin\IKVM.OpenJDK.Core.dll
  lit_libs += $(FL_ROOT)\bin\ikvm\ikvm-
  7.1.4532.2\bin\IKVM.Runtime.dll
}
```

Tool-Specific Differences

- Can use base projects to hide tool differences – just inherit and not worry about it, e.g., Visual Studio vs. gcc to enable OpenMP support:

In **`openmp.mpb`**:

```
project {
  specific(vc8,vc9,vc10,vc11) {
    openmp = true
  }
  specific(make) {
    genflags += -fopenmp
  }
}
```

Base Project Location

- Many are bundled with MPC in **`%MPC_ROOT%\config`** that are automatically referenced
- Found if in the same directory as the MWC file
- For organization, can put custom ones in a subdirectory of the workspace, say **`config`**, and reference it in the MWC file (or set it on the command line)

```
workspace {
  cmdline += -include config
```

Features

- Enable behavior by user choice
  - e.g., only include a project if a dependent library is installed; use **`requires`** to specify needed features
  - Can put in a base project so projects that inherit from it have that feature applied (don't compile anything that needs Qt if Qt isn't installed, say – see **`%MPC_ROOT%\config`**)
- Use command-line options or features file to identify which features are enabled
- Features are enabled by default, so need to explicitly disable them

Features

In **HelloCPP.mpc**:

```
project {
  requires += hellofeature
}

C:\src\Hello>%MPC_ROOT%\mwc.pl -type vc10
  -features hellofeature=0 Hello.mwc
Generating 'vc10' output using Hello.mwc
Skipping HelloCPP (HelloCPP.mpc), it
  requires hellofeature.
Generation Time: 0s
```

Exclude

- Use **`exclude`** to skip projects based on the tool used

```
workspace {
  CompileAlwaysProject
  exclude(!prop:windows) {
    WindowsOnlyProject
  }
}
```

Multi-language

- MPC supports multiple languages, too
  - e.g., for C#, if **`vc8`** to **`vc11`** is selected, the MS C# compiler is configured, but if the type is **`make`**, **`gmcs`** from Mono is used

In **`MyWorkspace.mwc`**:

```
workspace {
  CPPProject
  mycsharpprojects {
    cmdline += -language csharp
    CSharpProject1
    CSharpProject2
  }
}
```

MPC Output

- Currently (**`-type`** <type>):
  - **`automake, bcb2007, bcb2009, bds4, bmake, cc, cdt6, cdt7, em3, ghs, gnuace, html, make, nmake, rpmspec, sle, vc6, vc7, vc71, vc8, vc9, vc10, vc11, wb26, wb30, wix`**
- Most types are for build files, but special ones, too, such as:
  - **`wix`** – generate XML for WiX installer toolset

So, Why Use MPC?

- Straightforward, simple syntax
  - Project files can even often be nearly empty
- Generate project files across architectures and compilers from a single description
  - Still need to write multi-platform code, e.g.,
    - Boost or ACE libraries for C++
    - C# or Java languages
- Centralize build settings across hundreds of projects
  - Flowlink is Windows-only, but this has helped greatly

References

- To get MPC
  - http://www.ociweb.com/products/mpc
  - svn://svn.dre.vanderbilt.edu/DOC/MPC/trunk
- To see MPC examples:
  - My Middleware News Briefs (C++, C#, Java)
    - http://www.ociweb.com/mnb
  - ACE/TAO/OpenDDS (C++), QuickFAST (C++, C#)
    - http://www.theaceorb.com/
    - http://www.opendds.org/
    - http://code.google.com/p/quickfast/