Overview of *Freemason* Build System

Rafi Einstein
November 2006

RADVISION
Delivering the Visual Experience

Agenda

- Introduction
  - Rationale
  - Implementation Notes
- Requirements
- Architecture
- Module Definition
- Tool and Product Definitions
- Target Definitions
- Project Roadmap

RADVISION
Delivering the Visual Experience

Introduction

Rationale

- Standardization of the build process
  - Automation
  - Rigorous process
    - “Sourcification” of build methods
  - Use of canonical build tools
- Usability
  - Uniform build process interface (as much as possible)
    - For projects, targets, and tools
  - Work alongside of various IDEs
    - i.e., VS, Tornado, Eclipse
    - Reference for other build tools
    - Integration within the IDE

Rationale

- Low signature on most projects
  - Mostly declarative
- Arbitrary complexity where required
- Build performance
  - Use of pre-built modules: risks and benefits
  - Build parallelization

Implementation

- Using GNU make (with extensions) as a primary engine
  - Extensions enable improved diagnosis
- Framework of makefiles
- Runs on Windows (native, Cygwin) and Linux
  - Wherever make, sh and Perl are available
- Medium-level learn curve for maintainers

Agenda

- Introduction
- Requirements
- Architecture
- Module Definition
- Tool and Product Definitions
- Target Definitions
- Project Roadmap

RADVISION
Delivering the Visual Experience

Requirements

RADVISION
Delivering the Visual Experience

Requirements

- Multiple target platforms (Win32, VxW 5.5, VxW 6.3, Linux)
- Multiple host (builder) platforms (Windows, Linux)
- Usability
  - Concise operation
  - Complete automation
    - Source-to-Board by one command
- Simple module setup
  - Introduction of new modules, new source files
- Build system configurability
  - Targets
  - Tools
  - Builder hosts
- Dynamic module configuration
  - Source file selection
  - Compilation/Link options
  - Build variants

RADVISION
Delivering the Visual Experience

Requirements

- Module dependencies
  - Specification of module dependencies
  - Shallow/Deep build
  - Use of pre-built modules
- File dependencies
  - Auto detection of C file dependencies
  - Generated source files (*)
  - Precompiled headers (*)
- Independence
  - Separation from host environment
  - Use of canonical build tools
- Traceability
  - Makefile hierarchy and preprocessing
  - Tool invocations and outputs
- IDE Integration
- Documentation

Agenda

- Introduction
- Requirements
- Architecture
  - Concepts
  - Framework Structure
- Module Definition
- Tool and Product Definitions
- Target Definitions
- Project Roadmap

Architecture

RADVISION
Delivering the Visual Experience

Concepts

- Builder Host
- Builder OS
- Target Architecture (*1)
  - i386, PPC-604
- Target OS
- Target Platform (*2)
- Tool
  - CC Tool
- Product
  - Program, Library, Shared Object (DLL), FLS, etc.
- Module
  - Attributes (black-box)
  - Build method (white-box)
- Source View (*3)

Framework Structure

- core
  - bindir
  - depends
  - log
  - variant

- builder-host/@/HOST (*1)
- builder-os/@/OS

- module
  - product/@/PRODUCT
    - target-arch/ARCH
    - target-os/OS

- product/@/PRODUCT
  - target-arch/ARCH
  - target-os/OS

- target-arch/@/ARCH
- target-os/@/OS
- target-platform/@/PLATFORM

- tool/@/TOOL
  - builder-os/OS
  - target-arch/ARCH
  - target-os/OS
  - product/PRODUCT
    - target-arch/ARCH
    - target-os/OS

- tool/cc/@
  - depends
  - target-arch/ARCH
  - target-os/OS
  - product/PRODUCT
    - target-arch/ARCH
    - target-os/OS

Note: @ - point of dispatch

RADVISION
Delivering the Visual Experience

Agenda

- Introduction
- Requirements
- Architecture
- Module Definition
  - Build Method Specification
  - Module Attributes Specification
  - Samples
- Tool and Product Definitions
- Target Definitions
- Project Roadmap

Module Definitions

RADVISION Delivering the Visual Experience

Build Method Specification

- Configuration
  - Module
    - Name
    - Product
  - Host
    - OS
  - Target
    - Architecture
    - OS
  - Platform
  - Tools
- Framework Definitions
- Module dependencies
- Module resources
- Framework Rules

Build Method Sample: Program

```
MODULE_NAME=test1
PRODUCT=prog

#------------------------------------------------------------------------------------------

SRC_ROOT=..
include $(SRC_ROOT)/freemason/4/main

include $(MK)/defs

#------------------------------------------------------------------------------------------

include $(VROOT)/NBU_SW_INFRA/InfraStruct/defs.mk

CC_INCLUDE += ..
CC_SRC_FILES= \
        test1.cpp \
        main.cpp

#------------------------------------------------------------------------------------------

include $(MK)/rules
```

Configuration

Framework definitions

Module dependencies

Module resources

Framework rules

RADVISION
Delivering the Visual Experience

Build Method Sample: Library

```
MODULE_NAME=mcuInfraStruct
PRODUCT=lib

#-------------------------------------------------------------------------------------------

SRC_ROOT=../..
include $(SRC_ROOT)/freemason/4/main

include $(MK)/defs

#-------------------------------------------------------------------------------------------

CC_CXX_FLAGS += -DGCC_PRINT

CC_INCLUDE += \
	include \
	$(VROOT)/RVLOGGER/include \
	$(VROOT)/RVFC/include

CC_SRC_BASE=Source
CC_SRC_FILES=\
	mcuInfraInterfaceAgent.cpp \
	mcuInfraLogger.cpp \
	mcuInfraMemPool.cpp \
	mcuInfraMsgQReader.cpp \
	mcuInfraMsgQWriter.cpp \
	mcuSmStateMachine.cpp \
	mcuTimer.cpp \
	mcuTimerDeltaQ.cpp \
	mcuTimerManager.cpp

#-------------------------------------------------------------------------------------------

include $(MK)/rules
```

Build Method Sample: Library

```
MODULE_NAME=rvfc
PRODUCT=lib

#-----------------------------------------------------------------------------------------------

SRC_ROOT=../..
include $(SRC_ROOT)/freemason/4/main

include $(MK)/defs

#-----------------------------------------------------------------------------------------------

CC_INCLUDE += include .. Logger/inc $(VROOT) $(VROOT)/RVLOGGER/include

ifeq ($(TARGET_OS),vxworks-5.5)
P=Vx
else ifeq ($(TARGET_OS),win32)
P=Win
else
$(error Unexpected platform.)
endif

CC_SRC_FILES=\
        FileSystem/rvfcFfs.cpp          \
        Logger/srvAgent.cpp             \
        Logger/srvBuffMng.cpp           \
        Logger/srvConnect.cpp           \
        Logger/srvLogger.cpp            \
        Logger/srvOldLogger.cpp         \
        Logger/srvParam.cpp             \
        Logger/srvPrsTbl.cpp            \
        Logger/srvQue.cpp               \
        Logger/srvRsrc.cpp              \
        MsgQueue/rvfcQueue.cpp          \
        MsgQueue/rvfc$(P)MsgQueue.cpp   \
        Semaphore/rvfc$(P)Semaphore.cpp \
        Socket/rvfc$(P)Socket.cpp       \
        Sys/RvfcSys$(P).cpp             \
        Thread/rvfcThreadBase.cpp       \
        Thread/rvfc$(P)Thread.cpp       \
        Time/$(P)Time.cpp

#-----------------------------------------------------------------------------------------------

include $(MK)/rules
```

RADVISION
Delivering the Visual Experience

Module Attributes Specification

- Attributes
  - Module name
  - Module location
  - Product type
  - Module dependencies
  - Integration parameters
- Process
  - Deep build
    - Module dependencies are referenced in DFS order
  - Module artifacts are added to the proper tool argument variables
    - Can be overridden manually

RADVISION
Delivering the Visual Experience

Module Attributes Sample

```
ifndef _NBU_SW_INFRA_InfraStruct_
_NBU_SW_INFRA_InfraStruct_=1

#-------------------------------------------------------------------------------------------------------

MODULE=mcuInfraStruct
MODULE_DIR=$(VROOT)/NBU_SW_INFRA/InfraStruct
MODULE_PRODUCT=lib

include $(MK)/module/config
include $(MK)/module/defs

#-------------------------------------------------------------------------------------------------------

endif # _NBU_SW_INFRA_InfraStruct_
```

Agenda

- Introduction
- Requirements
- Architecture
- Module Definition
- Tool and Product Definitions
  - Generic Tool
  - CC Tool
  - Product Definitions
- Target Definitions
- Project Roadmap

Tool and Product Definitions

RADVISION
Delivering the Visual Experience

Generic Tool Specification

- builder-host/HOST/<localhost-TOOL-defs>
- tool/TOOL
  - target-arch/ARCH
  - target-os/OS
  - product/PRODUCT
    - target-arch/ARCH
    - target-os/OS
- Dispatchers are `tool/defs` and `tool/rules`
  - Unless we use CC tool

CC Tool

- Structure is similar to that of a generic tool
- Extra services
  - Uniform preprocessing
  - Compilation rules
  - Automatic source file dependencies generation
  - Construction of directories for binary files
- Tool selected by setting `CC_TOOL`
  - Implicitly, `TOOL` is set to `CC`

In project configuration file:

```
ifeq ($(TARGET_OS),win32)
CC_TOOL=msc-12
else ifeq ($(TARGET_OS),vxworks-5.5)
CC_TOOL=diab-5.0
endif
```

Product Definitions

- Products
  - Program, Library (.a/.lib), Shared Object (.so/.dll), FLS, RPM, Java Jar, .Net Assembly, etc.
- Template
  - product/PRODUCT
    - target-arch/ARCH
    - target-os/OS
- Instances
  - product
  - module/product
  - tool/TOOL/product
  - tool/cc/product

Agenda

- Introduction
- Requirements
- Architecture
- Module Definition
- Tool and Product Definitions
- Target Definitions
- Project Roadmap

Target Definitions

RADVISION
Delivering the Visual Experience

Target Classification

- target-arch/ARCH
  - Typically, CPU of the target
    - Open Issue!
  - If not specified, (HOST_ARCH) is used
- target-os/OS
  - If not specified, (HOST_OS) is used
- target-platform/PLATFORM
  - Typically specifies Architecture, OS, and build tools
  - Build tools can also be specified in a project common definitions file, based on the selected platform
  - If not specified, (TARGET_OS)-(TARGET_ARCH) is used

Agenda

- Introduction
- Requirements
- Architecture
- Module Definition
- Tool and Product Definitions
- Target Definitions
- Project Roadmap

RADVISION
Delivering the Visual Experience

Project Roadmap

RADVISION
Delivering the Visual Experience

Project Roadmap

- Iterations
  - Refinement of *Freemason* features
  - Setup of MCU Windows and VxWorks 5.5 build environments
- Setup of TAMAR-related facilities
  - VxWorks 6.3 target
  - Diab 5.4 tools
  - MCU VxWorks 6.3 build environment

RADVISION
Delivering the Visual Experience

Project Roadmap

- IDE integration
- Build parallelization
  - make –j
  - distcc, ccache
  - Incredibuild (not likely)

Thank you

RADVISION
Delivering the Visual Experience