Automating Builds with MSBuild and Team Foundation Server

John Bowen

- Software Engineer at InterKnowlogy
- Contact Information
  E-mail: jbowen@InterKnowlogy.com
  Blog: http://blogs.InterKnowlogy.com/JohnBowen

- More info on InterKnowlogy:
  www.InterKnowlogy.com

Topic Overview

- Why use build automation?
- MSBuild basics
- Running MSBuild
- Build execution and extensibility
- Team Foundation Build components of TFS
- Creating and running Team Builds
- Advanced MSBuild concepts

Build Automation Options

InterKnowlogy

- Build multiple configurations
- Copy binaries from other solutions
- Directly deploy/install
- Execute SQL scripts
- Perform static code analysis
- Run unit tests
- Generate documentation
- Send email notifications
- Automatic versioning
- Backup older builds
- Sign/encrypt files

MSBuild Background

- The Microsoft Build Engine (MSBuild) is the new build platform for Microsoft and Visual Studio 2005
- Can be run as independent command line tool
- Enables developers to build products in clean build lab environments where Visual Studio is not installed
- Uses XML project file format to describe build process
- Provides similar functionality to NAnt open-source tool

MSBuild Architecture

Team Build

TFS Targets and Tasks

Visual Studio

Built-in Targets and Tasks

MSBuild Engine

MSBuild Basic Concepts

InterKnowlogy

- Properties
  - Key/value pairs that can be used to configure builds
- Items
  - Inputs into the build system that are grouped into item collections based on their user-defined collection names
- Tasks
  - A unit of executable code used by MSBuild to perform atomic build operations
- Targets
  - Group tasks together in a particular order and allow sections of the build process to be called from the command line

Demo

- Visual Studio 2005 project file

Running MSBuild

InterKnowlogy

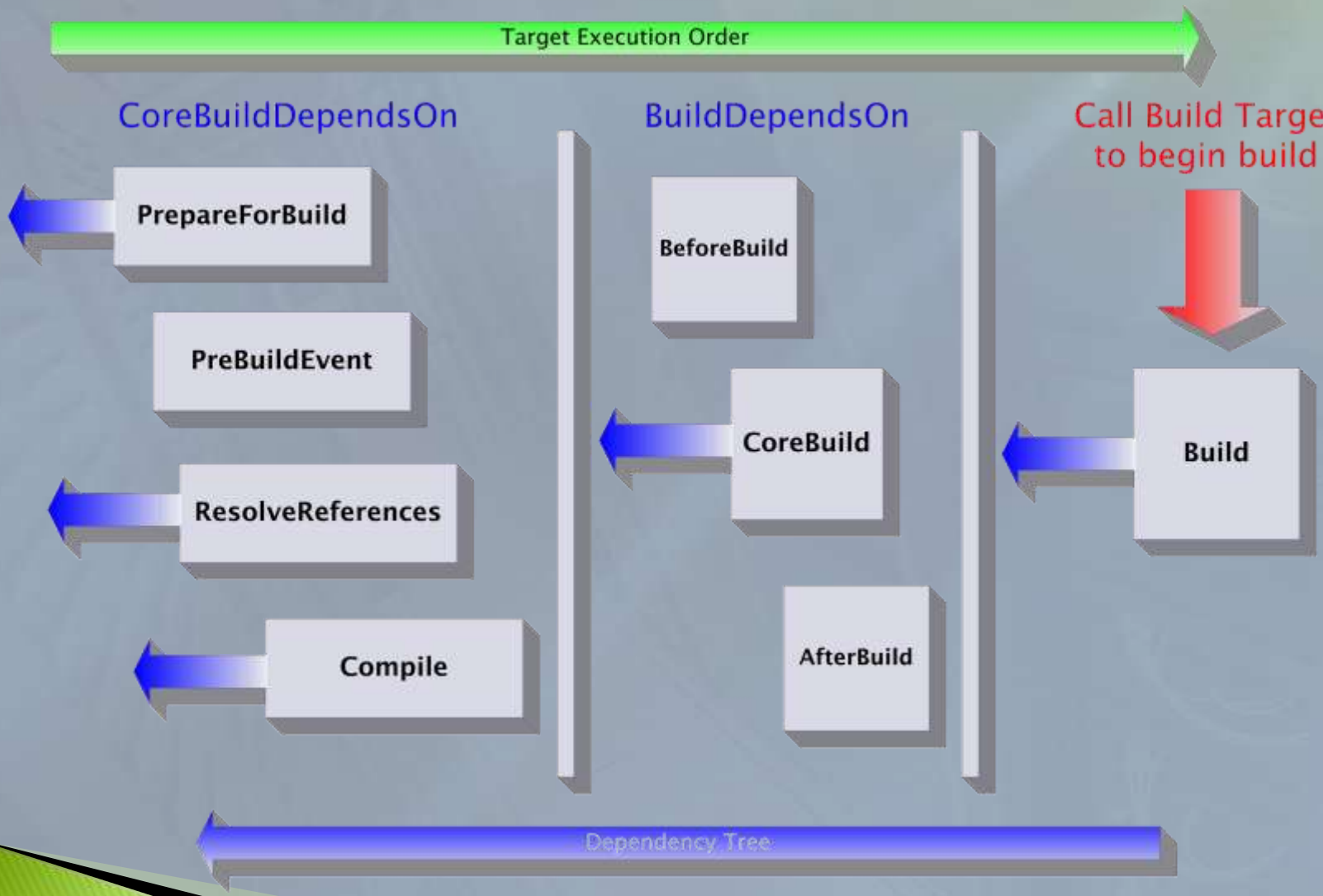
- Builds can be started inside Visual Studio or from the command line
- By default output is logged to the command line or Visual Studio Output window
- Logging to other output can be done using custom implementation of ILogger interface
- Verbosity setting can be used to provide more or less detail
  - quiet, minimal, normal, detailed, diagnostic

InterKnowlogy

Demo

- Building from the command line

MSBuild Target Execution



Dependency Execution

- Targets have DependsOn attribute to control target execution order
- MSBuild keeps track of executed targets to prevent multiple executions of a target
- Many standard targets are actually empty and only exist as dependency placeholders
- Built-in extensibility points are provided through existing overridable targets
- Additional targets can be added by overriding DependsOn attributes

Extensibility Points

InterKnowlogy

- Built-in User Targets
  - BeforeCompile, AfterCompile
  - BeforeBuild, AfterBuild
  - BeforeRebuild, AfterRebuild
  - BeforeClean, AfterClean
  - BeforePublish, AfterPublish
  - BeforeResolveReference, AfterResolveReferences
  - BeforeResGen, AfterResGen
- Overridable DependsOn Properties
  - BuildDependsOn
  - CleanDependsOn
  - CompileDependsOn

Built-in Tasks

InterKnowlogy

- AL (Assembly Linker)
- AspNetCompiler
- AssignCulture
- CallTarget
- Copy
- CreateItem
- CreateProperty
- Csc
- Delete
- Error
- Exec
- FindUnderPath
- GenerateApplicationManifest
- GenerateBootstrapper
- GenerateDeploymentManifest
- GenerateResource
- GetAssemblyIdentity
- GetFrameworkPath
- GetFrameworkSdkPath
- LC
- MakeDir
- Message
- MSBuild
- ReadLinesFromFile
- RegisterAssembly
- RemoveDir
- ResGen
- ResolveAssemblyReference
- ResolveComReference
- ResolveKeySource
- ResolveNativeReference
- SGen
- SignFile
- Touch
- UnregisterAssembly
- Vbc
- VCBuild
- Warning
- WriteLinesToFile

InterKnowlogy

Demo

- Message and Copy Tasks

Team Build Background

InterKnowlogy

- Team Foundation Build provides the functionality of a public build lab and is part of Visual Studio Team Foundation Server
- Implemented as an extension of MSBuild
- Builds on an independent build server providing consistent clean builds

Build Type Files

- Build Types consist of three files contained in a folder named for the Build Type
  - TFSBuild.proj – primary build script
  - WorkspaceMapping.xml
  - TFSBuild.rsp
- Build Types are found in the TeamBuildTypes folder in the Team Project source control root
- Files must be manually checked in and out for editing

Team Build Lifecycle

- Team Build by default runs EndToEndIteration target
  - Target is empty but causes dependency execution
- Work is done primarily inside Core targets
- Extensibility points are provided as overridable targets before and after Core targets

BeforeEndToEndIteration
BuildNumberOverrideTarget
InitializeEndToEndIteration
    BeforeClean
    CoreClean
    AfterClean
Clean
    InitializeBuild
        InitializeWorkspace
        BeforeGet
        CoreGet
        AfterGet
        BeforeLabel
        CoreLabel
        AfterLabel
    PreBuild
        BeforeCompile
        CoreCompile
        AfterCompile
    Compile
        GetChangeSetsAndUpdateWorkItems
    PostBuild
        BeforeTest
        CoreTest
        AfterTest
    Test
    PackageBinaries
TeamBuild
    BeforeDropBuild
    CoreDropBuild
    CopyLogFiles
    AfterDropBuild
DropBuild
AfterEndToEndIteration
EndToEndIteration

Creating and Running Builds

InterKnowlogy

- Builds can be created inside VS from the Build->New Team Build Type… menu item
  - Can be added manually as files into source control
- Builds can be started from Build->Build Team Project… or tfsbuild.exe at command line
  - Builds can only be stopped with tfsbuild.exe
- Build logs are created in the drop folder and linked from build reports
  - BuildLog.txt – full logger output
  - ErrorsWarningsLog.txt – error and warning output

InterKnowlogy

Demo

- Create and run new Team Build Type

Advanced Concepts

InterKnowlogy

- Refactoring to external targets files
- Item metadata
- Custom Tasks

Importing .targets Files

InterKnowlogy

- Custom Targets that will be reused can be refactored out into a .targets file
- External files are included with Import tag
- MSBuild and Team Build both use targets files which are included in every proj file
- During execution an external file behaves as though its entire content has been substituted for the Import tag

InterKnowlogy

Demo

- Show sample .targets file

Items and Metadata

InterKnowlogy

- Items act like object collections in contrast to name-value pair Properties
- Items have metadata associated with each member of the collection
  - Well-known metadata
    - automatically generated path and filename data
  - Custom metadata
    - user created and assigned at time of item creation
- Access items as vector(@), scalars(%), or through transforms

Demo

- Item metadata

Custom Tasks

- Implement ITask interface
- Can derive from Task to handle basic plumbing code
- Compiled into dll to be made available to MSBuild
- Referenced through UsingTask statement in project file

InterKnowlogy

Demos

- Custom Tasks
  MSBuild Sidekick
  Team Build .targets

Summary

- MSBuild Projects
  - Properties, Items, Tasks, Targets
  - Command line interface
  - Dependency execution
- Team Build
  - Customized implementation of MSBuild
  - TFSBuild.proj, TFSBuild.rsp, WorkspaceMapping.xml
- Advanced Concepts
  - Importing custom targets files
  - Item metadata
  - Custom Tasks

References and Further Info

- Deploying .NET Applications: Learning MSBuild and ClickOnce by Sayed Y. Hashimi and Sayed I. Hashimi, APress, 2006
- MSBuild on MSDN
  - http://msdn2.microsoft.com/en-us/library/wea2sca5.aspx
- Custom Tasks
  - http://msbuildtasks.tigris.org/
- MSBuild Sidekick
  - http://www.attrice.info/msbuild/