# Contents

1 *User documentation* .......................................................... 3
   1.1 Introduction ................................................................. 3
   1.2 How does it work? .......................................................... 3
   1.3 Getting started .............................................................. 5
   1.4 Extensibility ................................................................. 7
   1.5 Contributors ................................................................. 7
   1.6 Contribution Guidelines .................................................. 8
   1.7 AppVeyor integration ....................................................... 9

2 *Api documentation* ............................................................. 13
   2.1 API documentation .......................................................... 13
Cake (C# Make) is a build automation system with a C# DSL to do things like compiling code, copy files/folders, running unit tests, compress files and build NuGet packages.

- GitHub: https://github.com/cake-build/cake
- NuGet: https://www.nuget.org/packages/Cake
1.1 Introduction

1.1.1 What is Cake?

Cake is a build automation system with a C# DSL to do things like compiling code, copy files/folders, running unit tests, compress files and build NuGet packages.

1.1.2 Goals

The goal of Cake is to be a first class alternative to those who want to write their build scripts in C# instead of Ruby, F# or Powershell.

1.2 How does it work?

Cake uses a dependency based programming model just like Rake, FAKE and similar build automation systems where you declare tasks and the dependencies between those.

1.2.1 Tasks

To define a new task, use the Task-method.

```csharp
Task("A")
  .Does(() =>
    {
    })
);
RunTarget("A");
```

1.2.2 Dependencies

To add a dependency on another task, use the IsDependentOn-method.
Task("A")
  .Does(() =>
  {
  });

Task("B")
  .IsDependentOn("A");
  .Does(() =>
  {
  });

RunTarget("B");

This will first execute target A and then B as expected.

### 1.2.3 Criterias

You can also control the flow by providing *criterias*. A criterion is a predicate that has to be fulfilled for the task to execute. The criterion does not affect however succeeding task will be executed.

Task("A")
  .WithCriteria(DateTime.Now.Second % 2 == 0)
  .Does(() =>
  {
  });

Task("B")
  .WithCriteria(() => DateTime.Now.Second % 2 == 0)
  .IsDependentOn("A");
  .Does(() =>
  {
  });

RunTarget("B");

Task A’s criteria will be set when the task is created while Task B’s criteria will be evaluated when the task is being executed.

For criterias with states that might change during the execution of the build script, consider using the lambda alternative.

### 1.2.4 Error Handling

In order to act on an error thrown from an individual task, you can use the *OnError* task extension. This will give you the opportunity to act on the error. If you want the error to propagate as normal, you can rethrow the exception.

Task("A")
  .Does(() =>
  {
  })
  .OnError(exception =>
  {
    // Handle the error here.
  });

To automatically swallow errors that occur in a task, you can use the *ContinueOnError* task extension. The *ContinueOnError* extension cannot be combined with *OnError*. 

---

Cake Documentation, Release 0.1.21

---
Task ("A")
    .ContinueOnError()
    .Does(() =>
    {
    });

1.2.5 Setup and Teardown

If you want to do something before the first or after the last task has been run, you can use Setup and Teardown. A use case for this might be when you need to start some kind of server and want to make sure it gets teared down properly.

Setup(() =>
{
    // Executed BEFORE the first task.
});

Teardown(() =>
{
    // Executed AFTER the last task.
});

Setup will only be called if a call to RunTarget is made and the dependency graph is correct. If Cake doesn’t run any tasks, neither Setup or Teardown will be called.

1.3 Getting started

Assume our build script have four steps where we want to:

1. Clean up old artifacts.
2. Build the code.
3. Run unit tests.
4. Package generated artifacts.

1.3.1 1. Create the build script

A build script like that could look something like this.

```csharp
var target = Argument<string> ("target", "Package");
var config = Argument<string> ("config", "Release");

Task ("Clean")
    .Does(() =>
    {
        // Clean directories.
        CleanDirectory ("./output");
        CleanDirectory ("./output/bin");
        CleanDirectories ("./src/**/bin/" + config);
    });

Task ("Build")
    .IsDependentOn ("Clean")
    .Does(() =>
    {
```
{ // Build the solution using MSBuild.
MSBuild("./src/Project.sln", settings =>
    settings.SetConfiguration(config));
});

Task("RunUnitTests")
    .IsDependentOn("Build")
    .Does(() =>
        // Run xUnit tests.
        XUnit("./src/**/bin/" + config + "/*.Tests.dll");
    });

Task("CopyFiles")
    .IsDependentOn("RunUnitTests")
    .Does(() =>
        var path = "./src/Project/bin/" + configuration;
        var files = GetFiles(path + "/**/*.dll")
            + GetFiles(path + "/**/*.exe");
        // Copy all exe and dll files to the output directory.
        CopyFiles(files, "./output/bin");
    });

Task("Package")
    .IsDependentOn("RunUnitTests")
    .Does(() =>
        // Zip all files in the bin directory.
        Zip("./output/bin", "./output/build.zip");
    });

RunTarget(target);

1.3.2 2. Run the script

To run our build script we invoke Cake.exe with the script file as the first argument, and (optionally) the name of our target task as the second argument. We can also tell Cake how much information we’re interested in with the built in verbosity parameter, which is very useful when debugging a script.

C:> Cake.exe build.cake -target=Package -verbosity=diagnostic

This is just an example. Much more functionality is already implemented such as support for MSBuild, MTest, xUnit, NUnit, ILMerge, NuGet pack/restore and the most common file system operations such as file/folder manipulation and globbing.

1.3.3 3. Try it out

To see an actual build script in action, fork or clone the GitHub repository and run build.cmd which will download Cake from NuGet and run the build.cake script.
1.4 Extensibility

This section covers different ways of using external code in a Cake script, such as loading other scripts and referencing assemblies on disc or on NuGet.

1.4.1 Loading external scripts

To reference another Cake script, use the `#l` (or `#load`) directive. This directive will only import the script once, even if it’s defined multiple times in multiple scripts.

```cake
#l stuff.cake
```

or

```cake
#load stuff.cake
```

1.4.2 Referencing assemblies

To reference an assembly, use the `#r` (or `#reference`) directive.

```cake
#r "System.Drawing.dll"
#r "./lib/Stuff.dll"
```

or

```cake
#reference "System.Drawing.dll"
#reference "./lib/Stuff.dll"
```

1.4.3 Referencing NuGet add-ins

To reference a Cake add-in package hosted on NuGet, use the `#addin` directive. If the referenced add-in contains a Cake script alias, this will be imported automatically.

```cake
#addin Cake.Slack
```

You can also provide an (optional) NuGet package source to be used.

```cake
#addin Cake.Slack "TheNuGetPackageSource"
```

1.5 Contributors

Contributions have been accepted from:

- Patrik Svensson (@patriksvensson)
- Mattias Karlsson (@devlead)
- Fredrik Leijon (@FredrikL)
- Viktor Elofsson (@vktr)
- Jeff Doolittle (@jeffdoolittle)
- Richard Simpson (@RichiCoder1)
1.6 Contribution Guidelines

1.6.1 Introduction

So you’re thinking about contributing to Cake? Great! It’s really appreciated.

1.6.2 Where to start?

Start by either suggest a feature in the issue list that you want to see in Cake, or assign yourself to an existing feature/bug. Make sure that no one else is working on it already though. It’s also appreciated to have some kind of discussion about the issue if it’s not an easy fix, to make sure that it’s later.

If your suggestion applies to a broad range of users and scenarios, it will be considered for inclusion in the core Cake assemblies.

Make sure that your contribution follows this contribution guide.

Submitting

When your contribution is ready, send a pull request. Bonus for topic branches. *Funny .gif will be your reward. 😊*

1.6.3 Coding

Dependencies

The assemblies Cake.Core and Cake.Common should have no dependencies except the .NET BCL library.

Dependencies for Cake executable is acceptable in specific circumstances. If you want to introduce a dependency to the Cake executable, make sure you bring it up in the issue.

Unit Tests

Make sure to run all unit tests before creating a pull request. You code should also have reasonable unit test coverage.

Documentation

If your contribution change behavior or adds new functionality, it’s really appreciated (but not mandatory) to update the documentation at https://github.com/cake-build/documentation.

Coding Standards

Normal .NET coding guidelines apply. See the Framework Design Guidelines for suggestions.

If you have access to ReShaper, your code should have no ReShaper warnings or errors with the default settings.
1.6.4 License

By contributing to Cake, you assert that:

1. The contribution is your own original work.
2. You have the right to assign the copyright for the work (it is not owned by your employer, or you have been given copyright assignment in writing).
3. You license it under the terms applied to the rest of the Cake project.

1.7 AppVeyor integration

This section will describe how to use Cake with your AppVeyor CI builds.

1.7.1 1. Add prerequisites

Start by copying `NuGet.exe` to your tools folder. Cake uses the `tools` path as a convention for finding stuff it needs such as unit test runners and other tools.

- MyProject/
  - tools/
    * NuGet.exe

1.7.2 2. Add build script

Add a build script called `build.cake` to the project root. In this tutorial, we’ll just create a really simple build script for demonstration.

```csharp
// Get the target.
var target = Argument<string>("target", "Default");

Task("Default")
  .Does(() =>
    {
      Information("Hello from Cake!");
    });

RunTarget(target);
```

1.7.3 3. Add bootstrapper script

Create a old fashioned batch file called `build.cmd` that will download Cake and execute the build script.

```cmd
@echo off

:Build

cls

if not exist tools\Cake\Cake.exe (
  echo Installing Cake...
  tools\NuGet.exe install Cake -OutputDirectory tools -ExcludeVersion -NonInteractive -Prerelease
```
echo Starting Cake...
tools\Cake\Cake.exe build.cake -target=Default -verbosity=diagnostic

1.7.4 4. Configure AppVeyor

Now we need to tell AppVeyor how to start the Cake build. Do this by setting the build script for your AppVeyor project to build.cmd. Save your settings and you should be done.

![AppVeyor Test](image)

1.7.5 5. Try it out

The next triggered build will now execute the Cake build script as expected.
1.7. AppVeyor integration
2.1 API documentation

2.1.1 Arguments

Determines whether or not the specified argument exist.

```csharp
bool HasArgument(string name)
```

Gets an argument and throws if the argument is missing.

```csharp
T Argument<T>(string name)
```

Gets an argument and returns the provided `defaultValue` if the argument is missing.

```csharp
T Argument<T>(string name, T defaultValue)
```

2.1.2 Assembly Info

Creates an assembly information file.

```csharp
void CreateAssemblyInfo(FilePath outputPath, AssemblyInfoSettings settings)
```

 Parses an existing assembly information file.

```csharp
AssemblyInfoParseResult ParseAssemblyInfo(FilePath assemblyInfoPath)
```

2.1.3 Compression

Zips the specified directory.

```csharp
void Zip(DirectoryPath rootPath, FilePath outputPath)
```

Zips the files matching the specified pattern.

```csharp
void Zip(DirectoryPath rootPath, FilePath outputPath, string pattern)
```

Zips the specified files.

```csharp
void Zip(DirectoryPath rootPath, FilePath outputPath, IEnumerable<FilePath> filePaths)
```
Zips the specified files.

```csharp
void Zip(DirectoryPath rootPath, FilePath outputPath, IEnumerable<string> filePaths)
```

### 2.1.4 Directory Operations

#### Delete

Deletes the specified directories.

```csharp
void DeleteDirectories(IEnumerable<DirectoryPath> directories, bool recursive)
```

Deletes the specified directories.

```csharp
void DeleteDirectories(IEnumerable<string> directories, bool recursive)
```

Deletes the specified directory.

```csharp
void DeleteDirectory(DirectoryPath path, bool recursive)
```

#### Clean

Cleans the directories matching the specified pattern. Cleaning the directory will remove all it’s content but not the directory itself.

```csharp
void CleanDirectories(string pattern)
```

Cleans the specified directories. Cleaning a directory will remove all it’s content but not the directory itself.

```csharp
void CleanDirectories(IEnumerable<DirectoryPath> directories)
```

Cleans the specified directories. Cleaning a directory will remove all it’s content but not the directory itself.

```csharp
void CleanDirectories(IEnumerable<string> directories)
```

Cleans the specified directory.

```csharp
void CleanDirectory(DirectoryPath path)
```

#### Create

Creates the specified directory.

```csharp
void CreateDirectory(DirectoryPath path)
```

#### Copy

Copies a directory to the specified location.

```csharp
void CopyDirectory(DirectoryPath source, DirectoryPath destination)
```
2.1.5 Environment

Retrieves the value of the environment variable or null if the environment variable do not exist.

```csharp
string EnvironmentVariable(string variable)
```

Checks for the existance of a value for a given environment variable.

```csharp
bool HasEnvironmentVariable(string variable)
```

2.1.6 File Operations

Copy

Copies an existing file to a new location.

```csharp
void CopyFileToDirectory(FilePath filePath, DirectoryPath targetDirectoryPath)
```

Copies an existing file to a new file, providing the option to specify a new file name.

```csharp
void CopyFile(FilePath filePath, FilePath targetFilePath)
```

Copies all files matching the provided pattern to a new location.

```csharp
void CopyFiles(string pattern, DirectoryPath targetDirectoryPath)
```

Copies existing files to a new location.

```csharp
void CopyFiles(IEnumerable<FilePath> filePaths, DirectoryPath targetDirectoryPath)
```

Move

Moves an existing file to a new location.

```csharp
void MoveFileToDirectory(FilePath filePath, DirectoryPath targetDirectoryPath)
```

Moves existing files matching the specified pattern to a new location.

```csharp
void MoveFiles(string pattern, DirectoryPath targetDirectoryPath)
```

Moves existing files to a new location.

```csharp
void MoveFiles(IEnumerable<FilePath> filePaths, DirectoryPath targetDirectoryPath)
```

Moves an existing file to a new location, providing the option to specify a new file name.

```csharp
void MoveFile(FilePath filePath, FilePath targetFilePath)
```

Delete

Deletes the specified files.
void DeleteFiles(string pattern)

Deletes the specified files.
void DeleteFiles(IEnumerable<FilePath> filePaths)

Deletes the specified file.
void DeleteFile(FilePath filePath)

### 2.1.7 Globbing

**Files**

Gets all files matching the specified pattern.

```csharp
FilePathCollection GetFiles(string pattern)
```

**Directories**

Gets all directory matching the specified pattern.

```csharp
DirectoryPathCollection GetDirectories(string pattern)
```

### 2.1.8 ILMerge

Merges the specified assemblies.

```csharp
void ILMerge(FilePath outputFile, FilePath primaryAssembly, IEnumerable<FilePath> assemblyPaths)
```

Merges the specified assemblies.

```csharp
void ILMerge(FilePath outputFile, FilePath primaryAssembly, IEnumerable<FilePath> assemblyPaths, ILMergeSettings settings)
```

### 2.1.9 Logging

Writes an error message to the log using the specified format information.

```csharp
void Error(string format, params object[] args)
```

Writes a warning message to the log using the specified format information.

```csharp
void Warning(string format, params object[] args)
```

Writes an informational message to the log using the specified format information.

```csharp
void Information(string format, params object[] args)
```

Writes a verbose message to the log using the specified format information.

```csharp
void Verbose(string format, params object[] args)
```

Writes a debug message to the log using the specified format information.
void Debug(string format, params object[] args)

2.1.10 MSBuild

Builds the specified solution using MSBuild.
void MSBuild(FilePath solution)

Builds the specified solution using MSBuild.
void MSBuild(FilePath solution, Action<MSBuildSettings> configurator)

2.1.11 MSBuild Resource

Parses project information from project file
ProjectParserResult ParseProject(FilePath projectPath)

Parses project information from solution file
SolutionParserResult ParseSolution(FilePath solutionPath)

2.1.12 MSTest

Runs all MSTest unit tests in the assemblies matching the specified pattern.
void MSTest(string pattern)

Runs all MSTest unit tests in the assemblies matching the specified pattern.
void MSTest(string pattern, MSTestSettings settings)

Runs all MSTest unit tests in the specified assemblies.
void MSTest(IEnumerable<FilePath> assemblyPaths)

Runs all MSTest unit tests in the specified assemblies.
void MSTest(IEnumerable<FilePath> assemblyPaths, MSTestSettings settings)

2.1.13 NuGet

Pack

Creates a NuGet package using the specified Nuspec file.
void NuGetPack(FilePath nuspecFilePath, NuGetPackSettings settings)
**Restore**

Restores NuGet packages for the specified target.

```csharp
void NuGetRestore(FilePath targetFilePath)
```

Restores NuGet packages using the specified settings.

```csharp
void NuGetRestore(FilePath targetFilePath, NuGetRestoreSettings settings)
```

**Push**

Pushes a NuGet package to a NuGet server and publishes it.

```csharp
void NuGetPush(FilePath packageFilePath, NuGetPushSettings settings)
```

**AddSource**

Adds NuGet package source using the specified name & source to global user config

```csharp
void NuGetAddSource(string name, string source)
```

Adds NuGet package source using the specified name, source & settings to global user config

```csharp
void NuGetAddSource(string name, string source, NuGetSourcesSettings settings)
```

**RemoveSource**

Removes NuGet package source using the specified name & source from global user config

```csharp
void NuGetRemoveSource(string name, string source)
```

Removes NuGet package source using the specified name, source & settings from global user config

```csharp
void NuGetRemoveSource(string name, string source, NuGetSourcesSettings settings)
```

**HasSource**

Checks if NuGet package source exists in global user config using the specified source

```csharp
bool NuGetHasSource(string source)
```

Checks if NuGet package source exists in global user config using the specified source & settings

```csharp
bool NuGetHasSource(string source, NuGetSourcesSettings settings)
```

**2.1.14 NUnit**

Runs all NUnit unit tests in the assemblies matching the specified pattern.

```csharp
void NUnit(string pattern)
```

Runs all NUnit unit tests in the assemblies matching the specified pattern.
void NUnit(string pattern, NUnitSettings settings)

Runs all NUnit unit tests in the specified assemblies.

void NUnit(IEnumerable<string> assemblies)

Runs all NUnit unit tests in the specified assemblies.

void NUnit(IEnumerable<FilePath> assemblies)

Runs all NUnit unit tests in the specified assemblies.

void NUnit(IEnumerable<string> assemblies, NUnitSettings settings)

Runs all NUnit unit tests in the specified assemblies.

void NUnit(IEnumerable<FilePath> assemblies, NUnitSettings settings)

---

2.1.15 Process

Starts the process resource that is specified by the filename.

int StartProcess(FilePath fileName)

Starts the process resource that is specified by the filename and settings.

int StartProcess(FilePath fileName, ProcessSettings settings)

---

2.1.16 Release Notes

Parses all release notes.

IReadOnlyList<ReleaseNotes> ParseAllReleaseNotes(FilePath filePath)

Parses the latest release notes.

ReleaseNotes ParseReleaseNotes(FilePath filePath)

---

2.1.17 Signing

Signs the specified assembly

void Sign(string assembly, SignToolSignSettings settings)

Signs the specified assembly

void Sign(FilePath assembly, SignToolSignSettings settings)

Signs the specified assemblies

void Sign(IEnumerable<string> assemblies, SignToolSignSettings settings)

Signs the specified assemblies

void Sign(IEnumerable<FilePath> assemblies, SignToolSignSettings settings)
2.1.18 Text

Creates a text transformation from the provided template.

```csharp
TextTransformation<TextTransformationTemplate> TransformText(string template)
```

Creates a text transformation from the provided template on disc.

```csharp
TextTransformation<TextTransformationTemplate> TransformTextFile(FilePath path)
```

2.1.19 WiX

Candle

Compiles all .wxs sources matching the specified pattern.

```csharp
void WiXCandle(string pattern, CandleSettings settings)
```

Compiles all .wxs sources in the provided source files.

```csharp
void WiXCandle(IEnumerable<FilePath> sourceFiles, CandleSettings settings)
```

Light

Links all .wixobj files matching the specified pattern.

```csharp
void WiXLight(string pattern, LightSettings settings)
```

Links all .wixobj files in the provided object files.

```csharp
void WiXLight(IEnumerable<FilePath> objectFiles, LightSettings settings)
```

2.1.20 xUnit

Runs all xUnit.net tests in the assemblies matching the specified pattern.

```csharp
void XUnit(string pattern)
```

Runs all xUnit.net tests in the assemblies matching the specified pattern.

```csharp
void XUnit(string pattern, XUnitSettings settings)
```

Runs all xUnit.net tests in the specified assemblies.

```csharp
void XUnit(IEnumerable<string> assemblies)
```

Runs all xUnit.net tests in the specified assemblies.

```csharp
void XUnit(IEnumerable<FilePath> assemblies)
```

Runs all xUnit.net tests in the specified assemblies.

```csharp
void XUnit(IEnumerable<string> assemblies, XUnitSettings settings)
```
2.1.21 xUnit v2

Runs all xUnit.net v2 tests in the assemblies matching the specified pattern.

```csharp
void XUnit(IEnumerable<FilePath> assemblies, XUnitSettings settings)
```

Runs all xUnit.net v2 tests in the assemblies matching the specified pattern.

```csharp
void XUnit2(string pattern)
```

Runs all xUnit.net v2 tests in the specified assemblies.

```csharp
void XUnit2(string pattern, XUnit2Settings settings)
```

Runs all xUnit.net tests in the specified assemblies.

```csharp
void XUnit2(IEnumerable<string> assemblies)
```

Runs all xUnit.net v2 tests in the specified assemblies.

```csharp
void XUnit2(IEnumerable<FilePath> assemblies)
```

Runs all xUnit.net v2 tests in the specified assemblies.

```csharp
void XUnit2(IEnumerable<string> assemblies, XUnit2Settings settings)
```

Runs all xUnit.net v2 tests in the specified assemblies.

```csharp
void XUnit2(IEnumerable<FilePath> assemblies, XUnit2Settings settings)
```