TEAMPRISE EXTENSIONS FOR TEAM FOUNDATION BUILD

OVERVIEW

The following instructions are designed to help you build a Java application from a Microsoft Visual Studio Team Foundation Server Build (Team Build). Team Build is only able to execute MSBuild scripts to perform builds. Team Build creates a master build file - a bit like a boot strap build file. The default Team Build scripts have some useful functionality for labeling, downloading source, updating work items, calculating changes and performing other tasks. For .NET based builds, the default script fires off a child MSBuild process to build the .NET projects. The Teamprise Extensions for Team Foundation Build provide a custom MSBuild target and tasks that allow you to easily call Ant from within the TFSBuild.proj file created by Team Build. When a new build definition is created by Teamprise enabled clients, it is assumed that the Teamprise Extensions for Team Foundation Build are installed on all build agents on which the build definition may be run. You may also use these extensions without a Teamprise client.

As well as making the calling of Ant easier, the Teamprise Extensions for Team Foundation Build report build data to TFS and publish the results of JUnit test runs. The Team Build portions of Team Foundation Server have undergone a major overhaul as part of TFS 2008; therefore there are two versions of the custom target: one for TFS 2005 and for TFS 2008.

PREREQUISITES

You must have the following installed on the same server as your TFS Build Agent:-

- Java JDK (the latest one from Sun is recommended)

If you wish to publish the results of JUnit tests into Team Foundation Server then a Team Edition of Visual Studio Team Foundation Server must also be installed on the build agent to provide the “MSTest.exe” command line and to enable the publish functionality.

INSTALLING THE TEAMPRISE EXTENSIONS FOR TEAM FOUNDATION BUILD

The Teamprise Extensions for Team Foundation Build consist of the following components:

- Teamprise.Build.Ant.targets file
- Teamprise Build tasks assembly (Teamprise.Build.dll)
- JUnit transformation stylesheets

Due to the differences in the build functionality provided by TFS 2005 and TFS 2008, there are currently two versions of these, v1 for TFS2005 and v2 for TFS 2008.

USING THE INSTALLER

The easiest way to install the build extensions is to run the TeampriseBuildExtensions installer (msi) on the machine running the Team Foundation Build Server process (the build agent). This will install the necessary components into the MSBuild extensions path – usually located at %ProgramFiles%\MSBuild. It can also (optionally) install a private copy of Ant (v1.7.0) to perform the build.

MANUAL INSTALLATION

A .zip archive for manual installation is provided. If using this method, the contents of the archive should be extracted to disk and then the contents of the targets folder copied to %ProgramFiles%\MSBuild\Teamprise.

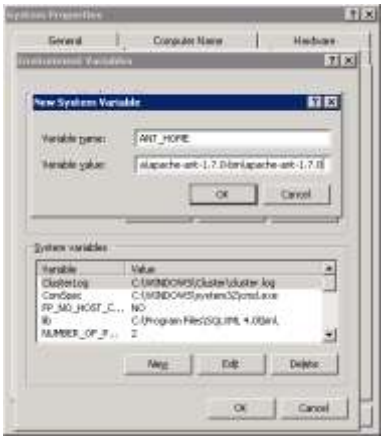
POST INSTALLATION TASKS

Once you have installed the Teamprise Extensions for Team Foundation Build onto the build agent, you may wish to configure the machine to provide the locations of Ant and your desired JDK. This is performed by setting the following environment variables at the machine level:

- ANT_HOME - The location of your chosen version of Ant. Note that if variable is not set, the build extensions will look for Ant in the %ProgramFiles%\MSBuild\Teamprise\apache-ant folder, the same path the installer uses if selected during the installation phase.
- JAVA_HOME - The version of Java that you wish to use to perform your builds. Note if you do not configure JAVA_HOME, the default JDK for your system will be used.

To set environment variables, go to Control Panel, System, Advanced and press the Environment Variables button.

Then, in the System Variables section, press New and enter the environment variable value.

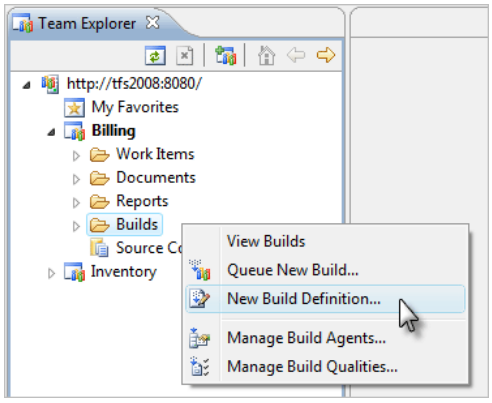


Using the Teamprise Extensions for Team Foundation Build

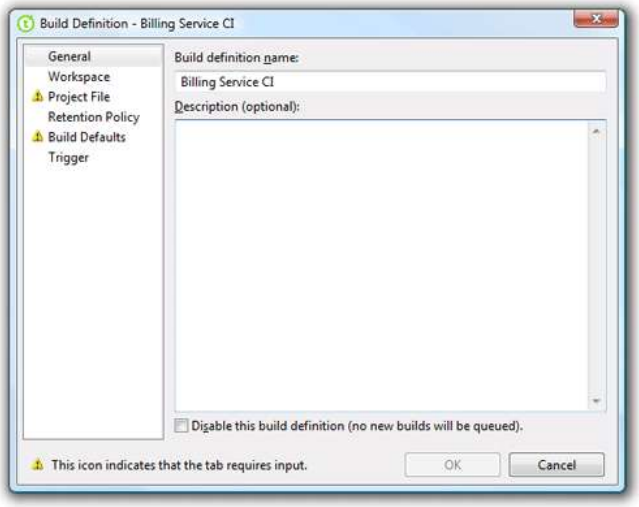
The simplest way to use the Teamprise Extensions for Team Foundation Build is to create a new build definition using either the Teamprise Plug-in for Eclipse or the Teamprise Explorer client. However, we will also explain how to manually create a TFSBuild.proj file that uses the extensions, and how to adapt existing Ant scripts to work within Team Foundation Build.

Create a Build Definition.

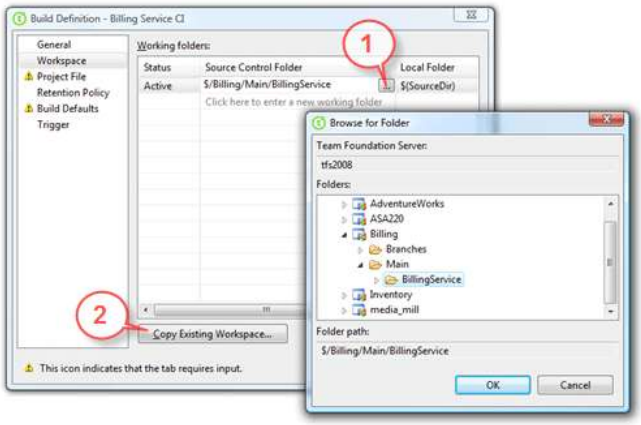
To create a new Team Foundation Server build definition, right click on the "Builds" node in the Team Explorer view and select "New Build Definition…"



When connected to a TFS 2008 server, you will see this build definition dialog.



Provide a name for the build, and (optionally) a description. Edit the workspace options by clicking on the Workspace section on the left hand side.

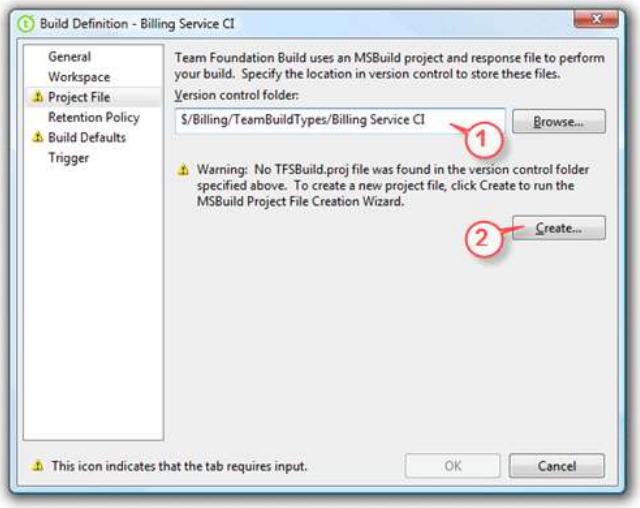


The Workspace section allows you to customize the TFS workspace that will be used to download the files required for the build. By default, it will download all files in the Team Project (which is usually more than you need); therefore edit the workspace (1) to limit it to the path that makes up your project or you may copy one of your existing TFS workspaces (2) and use that as a template.

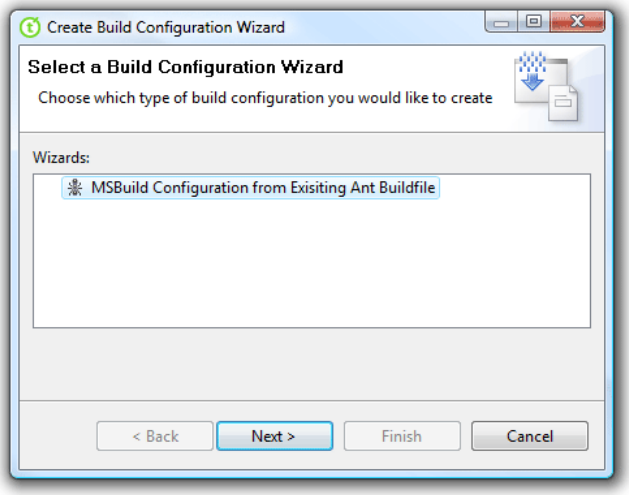
You may also specify a project file (TFSBuild.proj) file to use for the build, the retention policy, build defaults and trigger for the build definition. For more information on these see the Teamprise help documentation at http://help.teamprise.com/3.1/topic/com.teamprise.explorer.help/explorerdoc/build_def_v2.html

Create a Team Foundation Build Project File (TFSBuild.proj)

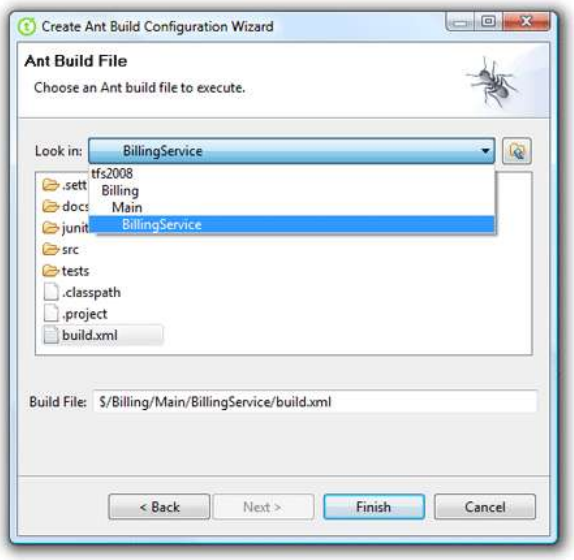
In the Project File section of the build definition dialog you specify the version control folder to use for your build configuration. By default, this is located in a folder named "TeamBuildTypes" in your Team Project; however in TFS 2008 you can customize the location of this folder using the following dialog:



If there is no TFSBuild.proj file located in the folder specified in (1), then the create button (2) will be enabled. Press this to create a new project file that will utilize the Teamprise Extensions for Team Foundation Build using the Create Build Configuration Wizard.



Select “MSBuild Configuration from Existing Ant Buildfile” and press Next. You will then be presented with the Create Ant Build Configuration Wizard, which allows you to select the Ant file that will be used to perform your build.



Note that you must select a file that will be included in the workspace declaration defined earlier; otherwise the build will fail because it will be unable to find the build file during the build process. Once you have selected the desired Ant build file from version control, press “finish” and a TFSBuild.proj file will be generated for your build and checked into version control in the build configuration folder location specified in the project file section.

The file when created will have the following structure.

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="DesktopBuild"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003" ToolsVersion="3.5">

  <!-- Do not edit this -->
  <Import
Project="$(MSBuildExtensionsPath)\Microsoft\VisualStudio\TeamBuild\Microsoft.TeamFoundation.
Build.targets" />
  <Import Project="$(MSBuildExtensionsPath)\Teamprise\v2\Teamprise.Build.Ant.targets" />

  <ProjectExtensions>
    <ProjectFileVersion>2</ProjectFileVersion>
    <Description></Description>
    <BuildMachine>buildserver.mycompany.com</BuildMachine>
  </ProjectExtensions>

  <PropertyGroup>
    <TeamProject>Billing</TeamProject>
    <BuildDirectoryPath>UNKNOWN</BuildDirectoryPath>
    <DropLocation>\\UNKNOWN\drops</DropLocation>
    <SkipWorkItemCreation>false</SkipWorkItemCreation>
    <WorkItemType>Bug</WorkItemType>
    <WorkItemFieldValues>
         System.Reason=Build Failure;System.Description=Start the build using Team
         Build</WorkItemFieldValues>
    <WorkItemTitle>Build failure in build:</WorkItemTitle>
    <DescriptionText>This work item created on a build failure.</DescriptionText>
    <BuildlogText>The build log file is at:</BuildlogText>
    <ErrorWarningLogText>The errors/warnings log file is at:</ErrorWarningLogText>
    <UpdateAssociatedWorkItems>true</UpdateAssociatedWorkItems>
  </PropertyGroup>

  <ItemGroup>
    <!--  Ant Call Configuration.
      The build file called should be included in the workspace of the build definition.
    -->
    <AntBuildFile Include="$/Billing/Main/BillingService/build.xml">
      <Targets></Targets>
      <Properties>
         BinariesRoot=$(BinariesRoot);BuildDefinitionName=$(BuildDefinitionName);BuildDefini
         tionUri=$(BuildDefinitionUri);BuildDirectory=$(BuildDirectory);BuildNumber=$(BuildN
         umber);DropLocation=$(DropLocation);LogLocation=$(LogLocation);SourceGetVersion=$(S
         ourceGetVersion);TestResultsRoot=$(TestResultsRoot);TeamProject=$(TeamProject);Work
         spaceName=$(WorkspaceName);WorkspaceOwner=$(WorkspaceOwner)</Properties>
      <Lib></Lib>
    </AntBuildFile>

    <JUnitLogFiles Include="$(BinariesRoot)\**\TEST-*.xml" />

  </ItemGroup>

</Project>
```

The Import statement at the top is calling the Teamprise.Build.Ant.targets file. This safely inserts the call to Ant into the Team Foundation Build process. It uses the `AntBuildFile` item group to specify the server path of the file to build – this is converted into a local path as part of the build process. It also specifies which JUnit XML results files to include as part of the build report (by default every file called TEST-*.xml in the `BinariesRoot` folder on the build server.

MODIFY EXISTING ANT SCRIPTS

In the properties section of the `AntBuildFile` configuration, common Team Build properties are passed through into the Ant build call as Ant properties. To make use of these properties in your Ant build script, you will need to use them as Ant properties that control your build process.

For example, the property `BinariesRoot` indicates where the binary results of the build should be placed on disk. In Java builds, this is sometimes called the output directory or the dist directory. The contents of the `BinariesRoot` folder are copied to a network share (called the `DropLocation`) at the end of the build process to make them available to your team. Therefore any permanent artifacts created by your build process should be placed within `BinariesRoot` to make sure they are part of your build.

The following Ant target is a sample of how pick up the passed BinariesRoot value inside your Ant Script.

```
<target name="init-output">
   <echo message="BinariesRoot is [${BinariesRoot}]" />

   <condition property="dir.output" value="${BinariesRoot}">
      <isset property="BinariesRoot" />
   </condition>
   <property name="dir.output" value="${basedir}/output" />

   <echo message="output directory is [${dir.output}]" />
   <mkdir dir="${dir.output}" />
</target>
```

In the above example, we assign the Ant property `dir.output` the value contained in the `BinariesRoot` property if it is passed; otherwise we default it to the value of `${basedir}/output`.

To get JUnit data to be reported into Team Foundation Build, the formatter of the build must be XML, i.e. <formatter type=”xml”> and the JUnit results must be located in `BinariesRoot`. The following is an example of a test target with an XML formatter.

```
<target name="test" depends="init-output">
        <delete dir="${dir.output}/reports" />
        <mkdir dir="${dir.output}/reports" />

        <junit printsummary="on" errorproperty="testsfailed">
                <classpath>
                        <pathelement location="${basedir}/bin" />
                        <pathelement location="${basedir}/junit/junit.jar" />
                </classpath>
                <batchtest todir="${dir.output}/reports">
                        <fileset dir="${basedir}/tests">
                                <include name="**/*Test*.java" />
                        </fileset>
                </batchtest>
                <formatter type="xml" />
        </junit>

        <junitreport todir="${dir.output}/reports">
            <fileset dir="${dir.output}/reports">
              <include name="TEST-*.xml" />
            </fileset>
            <report todir="${dir.output}/reports" />
          </junitreport>
```

```
        <fail if="testsfailed" message="tests failed" />
</target>
```

APPENDIX: TEAMPRISE TEAM FOUNDATION BUILD TARGETS AND PROPERTIES

The Teamprise.Build.Ant.targets file safely inserts the call to Ant into the Team Foundation Build process, by default after the "AfterCompile" target in Team Foundation Build. It also defines a set of targets (marked with an *) suitable for being overridden in the calling TFSBuild.proj file should you want to execute custom MSBuild logic as part of the Team Build process. These custom targets are in addition to the existing MSBuild targets.

	Target Name	Description
*	**BeforeCallAnt**	Override to perform logic before the call to Ant.
	ComputeAntBuildFileLocation	Calculate the local path to the Ant Build File
	CoreCallAnt	Perform the call to the Ant task
*	**AfterCallAnt**	Override to perform logic after the call to Ant
*	**BeforePublishJUnit**	Override to perform logic before the JUnit results are published.
	CorePublishJUnit	Make the call to RunPublishJUnit to actually publish the results with the latest versions of the JUnit results created during the Ant build process.
	RunPublishJUnit	Publish the JUnit results to TFS by calling the PublishJUnit task. This aggregates all the defined JUnit Log results into a single file and then transforms this into a Microsoft Test Results (TRX) file before publishing the results to TFS using the MSTest.exe command line application provided with a team edition of Visual Studio.
*	**AfterPublishJUnit**	Override to perform logic after the JUnit results have been published.

In addition to providing custom targets, there are several build properties that can be defined in the TFSBuild.proj file to control the behavior of the build process.

Property	Description
ANT_HOME	Set the system environment variable ANT_HOME or set ANT_HOME in the calling TFSBuild.proj if you need to specify a particular instance of Ant. By default it will look for a directory under the Teamprise MSBuildExtensions location.
ANT_OPTS	Set default options to be passed to Java when executing Ant. This is most commonly used to set Java heap options, i.e. " –Xmx1024m". Set the system environment variable ANT_OPTS or set an ANT_OPTS propery in the calling TFSBuild.proj if you need to specify any options.
AutoAddAntTestSummary	During the Ant execution step, automatically add test summary to the build report. Note that this should typically be set to 'false' as the test results will later be published using the PublishJUnit task. Defaults to "false".
AutoCallAnt	Defines when Ant should be called. Valid values are "before", "after" or "never" relative to the .NET compilation process. Entering a value such as "never" will mean that the CallAnt target it not inserted by this target and the calling MSBuild file takes the responsibility for ensuring it is called at the correct time. Defaults to "after".
AutoFixupConfigurationList	By default this will be set to "true" to prevent an empty CompilationSummary being added to the build in the case that no .NET build is being performed during the build.
FixupEnvironmentVariableCase	Due to a bug in versions of the .NET framework prior to 4.0, when Team Build starts MSBuild to perform the build, all the environment variables will be in lower case. By default the Ant wrapper will attempt to correct the case by comparing the current processes environment variables with the ones defined at the machine and user level.
JUnitTrxConversionStyle	Path to the stylesheet used to convert aggregated JUnit results into a Microsoft .trx file. Defaults to the path used by the Teamprise Extensions for Team Foundation Build installer: • "$(MSBuildExtensionsPath)\Teamprise\v2\xslt\tpaggregated_trx.xslt" for a TFS2008 build definition or • "$(MSBuildExtensionsPath)\Teamprise\v1\xslt\tpaggregated_trx.xslt" for TFS 2005.
PublishJUnitResults	Defines when the PublishJUnitResults task should be called. The valid values are 'before', 'after' or 'never'. This indicates if the publish step should be called before or after the .NET test steps. For example, if you indicate 'before' then the publish step will be placed before the 'BeforeTest' target in Team Build; if you select 'after' then it will run after the 'AfterTest' target.
SkipJavacBuildSteps	Suppress automatic reporting of JavaC steps in the build report. Defaults to "false".
SkipPublishJUnitResults	Flag to indicate if the PublishJUnitResults step should be performed. 'True' means that it should be skipped, 'false' means that it should not be skipped.
SkipStdErrorBuildSteps	Suppress automatic reporting of StdErr output in the build report. Defaults to "false".

Appendix: Task Reference

The following custom MSBuild Tasks are provided by the Teamprise Extensions for Team Build. Details are provided for each of them should you wish to call them directly.

- Ant Task
- PublishJUnit Task

Ant Task Reference

The Ant task works by calling Java to run Ant. The task first parses the Ant file to locate the name of the project and its description. It then calls Ant and the resulting output from Ant is parsed by the task to locate key information (such as javac and junit tasks) as well as to pass the results into the MSBuild log. Options which are normally available via an Ant launching script are available as additional attributes to the Ant task.

Example Usage

```
    <Ant TeamFoundationServerUrl="$(TeamFoundationServerUrl)"
         BuildFile="$(SolutionRoot)\java\HelloWorld\build.xml"
         BuildUri="$(BuildUri)"
         AntHome="C:\Java\apache-ant-1.7.0-bin\apache-ant-1.7.0"
         JavaHome="C:\Java\jdk1.6.0_02"
         Flavor="%(ConfigurationToBuild.FlavorToBuild)"
         Platform="%(ConfigurationToBuild.PlatformToBuild)"
         Properties="BinariesRoot=$(BinariesRoot);BuildDefinitionName=$(BuildDefinitionName)
         ;BuildDefinitionUri=$(BuildDefinitionUri);BuildDirectory=$(BuildDirectory);BuildNum
         ber=$(BuildNumber);DropLocation=$(DropLocation);LogLocation=$(LogLocation);SourceGe
         tVersion=$(SourceGetVersion);TestResultsRoot=$(TestResultsRoot);TeamProject=$(TeamP
         roject);WorkspaceName=$(WorkspaceName);WorkspaceOwner=$(WorkspaceOwner)"
         />
```

Task Reference

The following is a complete list of all the attributes supported by the task. Note that many of them are best left to the default values unless a different behavior is explicitly required. Items that are in bold are most frequently used.

Parameter	Required	Description
AntHome	**No**	**Location of Ant on Build Server. If not specified, then the value of the ANT_HOME environment variable will be used.**
AutoProxy	No	In Java 1.5+, use the OS proxies
BuildFile	**No**	**Name of the build file to use; by default this is "build.xml" in the current directory.**
BuildUri	**Yes**	**The team system URI which uniquely represents the instance of the build being run. With-in a Team Build MSBuild script, this is normally available in the MSBuild property $(BuildUri)**
Debug	No	Set to “true” to instruct Ant to print debugging information. By default this is set to “false”.

Parameter	Required	Description
Flavor	**No**	**The "flavor" of the build i.e. Release, Debug etc. This will default to "Release". In the Team Build MSBuild scripts, this is normally available as the global property %(ConfigurationToBuild.FlavorToBuild)**
InputHandler	No	Specifies the Ant class which will handle input requests
JavaHome	**No**	**Location of Java home directory on build server. If not specified then the value of the JAVA_HOME environment variable will be used.**
KeepGoing	No	Instructs Ant to execute all targets that do not depend on failed target(s)
Lib	No	Specifies a path for Ant to search for jars and classes.
Listener	No	Adds an instance of an Ant class as a project listener
Logger	No	Specifies an Ant class to perform logging.
Main	No	Overrides Ant's normal entry point with specified Ant class.
NoClasspath	No	Runs Ant without using CLASSPATH
Noinput	No	Do not allow interactive input in Ant script
NoJavacBuildSteps	No	Set to "true" to suppress the reporting of javac steps to TFS. By default javac steps are added as build steps.
NoUserLib	No	Run Ant without using the jar files from ${user.home}/.ant/lib
Platform	**No**	**The build platform i.e. any CPU, x86, x64. This will default to "Any CPU". In the Team Build MSBuild script, this is normally available as the global property %(ConfigurationToBuild.PlatformToBuild)**
Properties	**No**	**Properties to pass to Ant in "name=value;name2=value2" syntax. When calling Ant, it is often useful to pass through properties from the originating MSBuild script – for example Properties="BinariesRoot=$(BinariesRoot);BuildDefinitionName=$(BuildDefinitionName);BuildDefinitionUri=$(BuildDefinitionUri);BuildDirectory=$(BuildDirectory);BuildNumber=$(BuildNumber);DropLocation=$(DropLocation);LogLocation=$(LogLocation);SourceGetVersion=$(SourceGetVersion);TestResultsRoot=$(TestResultsRoot);TeamProject=$(TeamProject);WorkspaceName=$(WorkspaceName);WorkspaceOwner=$(WorkspaceOwner)"**
PropertyFile	No	Instructs Ant to load all properties from file with -D properties taking precedence
Target	**No**	**Single Ant Target to execute. If not specified then the default target specified in the script will be used.**
Targets	No	Comma separated list of Ant targets to execute.
TeamFoundationServerUrl	**Yes**	**The URL of the Team Foundation Server. In the Team Build MSBuild script, this is often available in the property $(TeamFoundationServerUrl)**
Verbose	No	Set to "true" to instruct Ant to be extra verbose.

PUBLISHJUNIT TASK REFERENCE

The task works by aggregating the passed JUnit XML formatted result files into a single file and then transforming it using the passed stylesheet into the Microsoft Test Results file (TRX).

EXAMPLE USAGE

```
<PublishJUnit
  Files="@(JUnitLogFiles)"
  TestResultsRoot="$(TestResultsRoot)"
  TeamFoundationServerUrl="$(TeamFoundationServerUrl)"
  BuildUri="$(BuildUri)"
  BuildNumber="$(BuildNumber)"
  TeamProject="$(TeamProject)"
  Flavor="%(ConfigurationToBuild.FlavorToBuild)"
  Platform="%(ConfigurationToBuild.PlatformToBuild)"
  Style="$(MSBuildExtensionsPath)\Teamprise\v2\xslt\tpaggregated_trx.xslt"
  />
```

TASK REFERENCE

The following is a complete list of all the attributes supported by the task. Note that many of them are best left to the default values unless a different behavior is explicitly required.

Parameter	Required	Description
Files	N	Item group indicating the JUnit XML Results files to publish. If not provided or the provided files do not exist, the task will print a message and execute without error.
`TestResultsRoot`	Y	The directory in which to place the intermediate amalgamated JUnit XML file along with the transform TRX file. Typically the $(TestResultsRoot) directory should be used in a Team Foundation Build.
`TeamFoundationServerUrl`	N	The URL to the Team Foundation Server that should be used to publish the build results. If not provided then the results will not be published to a server but the TRX file will still be created in the TestResultsRoot directory.
BuildUri	N	The URI of the build that is currently being executed. This value must be specified if a TeamFoundationServerUrl has been provided.
`BuildNumber`	N	The number of the current build. This value must be specified if a TeamFoundationServerUrl has been provided.
TeamProject	N	The Team Project under which the current build is created. This value must be specified if a TeamFoundationServerUrl has been provided.
`Flavor`	N	The flavor of the current build configuration; defaults to “Any CPU”
`Release`	N	The release type of the current build configuration; defaults to “Release”
`Style`	Y	The XSLT file to be used to transform the amalgamated JUnit XML result file into a Microsoft Test Results (TRX) file.

APPENDIX : REVISION HISTORY

Revision	Description
V1.0.0	Initial version of the Teamprise Build Extensions
V1.1.0	Add PublishJUnit task to convert JUnit XML Log output into Microsoft Test Results file (.trx) format and publish to TFS. Also add ability to specify ANT_OPTS.