Quest® ToolBox 3.2.2

User Guide

Legend

■ **WARNING: A WARNING icon indicates a potential for property damage, personal injury, or death.**

! **CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.**

i **IMPORTANT NOTE**, **NOTE**, **TIP**, **MOBILE**, or **VIDEO:** An information icon indicates supporting information.

ToolBox User Guide
Updated -
Software Version - 3.2.2

Contents

1 Overview

Introduction

The ToolBox (formerly “ValidationTool”) is an internal tool designed to assist developers and QC in automating repetitive or error-prone tasks. The motivation for writing the ToolBox was a simple tool similar to ANT or MAVEN which has a simple, easy to learn syntax, that provided a standard model for writing tools during development and which had a minimal learning curve. I wanted a tool that could be used in a variety of situations without having to learn a grammar and syntax like PowerShell. It was imperative that the tool be useful both from a developer’s main development machine as well as the automated build process on TFS.

Installation is as simple as copying the files into a folder – there is currently no installer, though as the number of support assemblies grow, this is something we may consider.

Displaying Basic Information

The ToolBox consists of a command-line client EXE (ToolBox.exe), the core ToolBox functionality (ToolBox.Core.dll), several task assemblies (ToolBox.Tasks.*.dll) and supporting assemblies for the task assemblies. As the tool’s repertoire of tasks has expanded, there are now several supporting assemblies that are part of the package; this will increase as the tool is further expanded.

To display basic information

- `C:> ToolBox`

TOOLBOX v3.1.0.1736

To display basic help information

- `C:> ToolBox -help`

TOOLBOX v1.1.1.115

TOOLBOX -DEPENDENCIES

Checks all task assemblies to ensure dependencies are available

TOOLBOX -DIR

Lists chain-files in the TOOLBOX_SCRIPTS folder

TOOLBOX -EXECUTE <CHAINFILE> [-VARS !(VAR1)=VALUE1 !(VAR2)=VALUE2 ...] [-DEBUG] [-REMOTE HOST:PORT] [-PUSH] [-ALTPATH PATH] [-MAXLOGS COUNT] [-LOG PATH]

Executes the specified chain-file

TOOLBOX -HELP [TASKNAME]

Displays usage information for ToolBox.exe or the specified task

TOOLBOX -LIST [ASSEMBLY1.DLL;...;ASSEMBLYN.DLL]

Displays the list of all tasks or those in the given assemblies

TOOLBOX -MAN [OPERATION]

Provides UNIX-style help for the specified operation

TOOLBOX -NEWCHAIN <CHAINFILE>

Creates a new chain file stub with the given name

TOOLBOX -REGISTER

Registers the tasks contained in any assemblies named, 'ToolBox.Tasks.*.dll'

TOOLBOX -REMOTE_MODE [PORT]

Starts up the ToolBox in remote mode, on the specified port

TOOLBOX -REMOTE_RECONNECT <HOST:PORT> [-MAXLOGS COUNT]

Connects to the specified remote ToolBox and listens for log messages

TOOLBOX -SEARCH <KEY>

Displays all tasks with a name or description containing <key>To retrieve a list of available tasks

- `C:> ToolBox -list`

TOOLBOX v1.1.1.115

[ACTIVEDIRECTORY]

ADContact - The task manages Active Directory Contact

BackupActiveDirectory - Backs up Active Directory data to a SQLite file

EnableActiveDirectoryAccount - Enable/Disable an Active Directory account

JoinDomain - Joins a computer to a domain

LeaveDomain - Removes a computer from a domain

PopulateAdCountryState - Populates ActiveDirectory with Country/State OUs

PopulateAdUsersGroupsComputers - Populates ActiveDirectory with Users groups and ciomputers

PopulateAdUsers - Populates ActiveDirectory with the specified number of users

RenameActiveDirectoryOUs - Renames Active Directory OUs to include numbers

RestoreActiveDirectory - Restores Active Directory Data from a SQLite file

[BUILD]

JavaScript - Runs JavaScript

MsBuild - Performs solution build using MSBuild

NuGet - Performs NuGet operations

PseudoLocalize - Performs resource pseudo localization

SetPkgVersion - Performs search through folders and changes the package versions

[COMPUTER]

RestartComputer - Restarts the specified computer

SetComputerServiceLogonRights - Sets computer service logon rights

SetServiceAccount - Sets the NT Service account and password

StartService - Starts NT Service

TakeScreenshot - Takes a screenshot of the entire screen or a specific window

[DATABASE]

AddUpgradeScriptsToInstaller - Adds upgrade scripts to installer

BackupDatabase - Backs up the specified database

CheckLogErrors - Check Log for Errors

CompareIndexNames - Compares the indices in database X with those in database Y

CompareRolePermissions - Compares the permissions for each table and function in database X with those in database Y

CompareRoutines - Compares the routines in database X with those in database Y

CompareTableContent - Compares the data in the specified tables in database X with the same tables in database Y

CompareTablesColumns - Compares the columns in the specified tables of database X with those in database Y

CompareTables - Compares the tables in database X with those in database Y

CompareViewsColumns - Compares the views columns in the specified views of database X with those in database Y

CompareViews - Compares the views in database X with those in database Y

CreateDatabase - Creates a new database using dbconfig.exe

CreateLocalDbInstance - Creates a new LocalDb Instance if it doesn't exist

DeleteLocalDbInstance - Deletes a LocalDb Instance

DropDatabase - Drops a database

DumpReporterLogsToDatabase - Dumps the contents of an ER log into the database

EnumerateIndices - Enumerates the indices in the given database

EnumerateRoutines - Enumerates the routines in the given database

EnumerateTables - Enumerates the tables in the given database

EnumerateViews - Enumerates the views in the given database

ExecuteFileQuery - Executes a SQL query from a sql file

ExecuteQuery - Executes a SQL query

ExportDataToCsv - Exports data from a database to a CSV file

ExportDataToSqlite - Exports data to a SQLite database

ExportDataToSqlServer - Exports data from a SQL Server database into another SQL Server database

ExportHttpToSqlServer - Save Graph properties to the database table

ExportJsonToSqlServer - Exports json data into SQL Server database

ExtractLastJobExecutionTime - Extracts Last Job Execution Time from SQLite database

FixDatabaseScript - Generates database creation script

GenerateReportsScript - Generates reports scripts

GenerateResetSecurityScript - Generates reset security script file

GenerateUpgradeScriptStub - Generates an upgrade script stub from version X to version Y

GetMissingTables - Discover missing tables

RestoreDatabase - Restores a backed-up database and gives it a new name

SQLDataCompareProjectResult - Compares two databases using RedGate DataCompare tool

SQLDataCompareProject - Compares two databases using RedGate DataCompare tool

SQLSchemaCompareProject - Compares two database schema using RedGate SQL Schema Compare tool

StartLocalDbInstance - Starts a LocalDb Instance

StopLocalDbInstance - Stops a LocalDb Instance

UpgradeDatabase - Upgrades an existing database

ValidateIndexNames - Examines the specified database to ensure index names match a specified template

ValidateReportIds - Checks report definitions for unique identifiers

ValidateRoles - Ensures the specified database has had appropriate roles applied to each table

[DATAVIZ]

DumpRunInfo - Dumps run data from the main database to a file

VisualiseJobRun - Generates sequence diagrams for tasks of a JobRun

[FILES]

AppendFiles - Concatenates all files together

CompareTextFiles - Compares two text files and outputs lines with different content

ConvertRTFtoText - Converts an RTF document to a plain text document

CopyFilesFromShare - Copies files from remote share source directory to the target

CopyFiles - Copies files from source directory to the target

CopyMatchingFiles - Copies files from source to target directory based on matching directory structure

CreateFolder - Creates a folder if doesn't exist

CreateZip - Creates compressed zip file

DeleteFiles - Copies files from source directory to the target

DeleteFolder - Deletes a folder

FileExists - Check if file exists

FileSize - Enumerates files under a root folder and calculates total size

JsonFileCompare - Compares two Json files for equality

ReplaceTextInFile - Replaces text in a file. It can be text search or regular expression

ReplaceVersionInFileName - Replaces version in a file name

ReplaceVersionInFile - Replaces version in a file

ReSignStrongName - Uses Enhanced Strong Names signing

ResourceUpdate - Updates resource in a binary

SevenZip - Runs 7zip tool

SignTool - Timestamps and digitally signs binaries

SignVerify - Vetrifies digitally signs binaries

UnZip - Unzips zip file

ValidateAssemblyVersions - Examines a set of assemblies and checks for matching file versions

ValidatecsvFormat - Validates that all given csv files are in the correct format and data types are correct

ValidateInstalledFilesDetails - Validates installed files are properly signed

ValidateInstalledFilesSigned - Validates installed files are properly signed

[IMAGES]

IconGenerator - Generates images of various sizes and colours from a folder of SVG files

[POWERSHELL]

ConfigurePowerShell - Configures PowerShell on a computer

CopyFilesFromRemote - Copies files from remote computer using Remote Powershell

CopyFilesToRemote - Copies files to a remote computer using Remote Powershell

ExecutePowerShellCommand - Executes an arbitrary PowerShell command

ExecutePowerShell - Executes an arbitrary PowerShell script

SetPowerShellUnrestricted - Set the Unrestricted PowerShell property

[PROCESSES]

Echo - Echos the input string

ExecuteProcess - Executes an arbitrary process

Sleep - Waits for a specified number of seconds

[SECURITY]

AslrCheck - Performs check for ASLR flag in binaries using Microsoft BinScope tool

HashCode - Generates the hashcode for a string value

[TFS]

TFSAdd - Adds a file to TFS

TFSChangesetAssemblies - Show modified assemblies within a range of TFS changesets

TFSCheckOut - Checks out a file from TFS

TFSQueueBuild - Enqueues a new build task in TFS

TFSSync - Syncs Local Directory Files with TFS Directory

```
[SERVER]
UpgradeServer32 - Upgrades server form <3.2 to 3.2

[VISUALSTUDIO]
FindDeadProjects - Locates unreferenced (dead) projects
LineCount - Performs a basic lines of code count
StaticAnalysis - Examines a set of projects to ensure static analysis tools are properly configured
VSExternalTools - Creates a folder if doesn't exist
VSInstallVsix - Installs a Visual Studio extension

[VMWARE]
VMDeleteSnapshot - Delete VMWare snapshots
VMPowerOff - Powers Off VMWare Computer
VMPowerOn - Powers On a VMWare computer
VMRevertSnapshot - Revert VMWare snapshots
VMRunInGuest - Runs program in guest
VMTakeSnapshot - Take VMWare snapshots

Found 120 tasks
```

2 Working With Tasks

This section provides some examples of how you can retrieve information about specific tasks.

Searching for Tasks

To search for tasks

- `C:> ToolBox -search create`

```
TOOLBOX v1.1.1.115

[DATABASE]
CreateDatabase - Creates a new database using dbconfig.exe
CreateLocalDbInstance - Creates a new LocalDb Instance if it doesn't exist

[FILES]
CreateFolder - Creates a folder if doesn't exist
CreateZip - Creates compressed zip file

Found 4 tasks
```

Retrieving Help Information for Specific Tasks

To retrieve help information for a specific task

- `C:> ToolBox -help JoinDomain`

```
TOOLBOX v1.1.1.115

TASK: CreateDatabase
--------------------
<?xml version="1.0" encoding="utf-8" ?>
```

```
<TaskChain [failonerror]>

<task name="CreateDatabase" [id] [enabled] [fatal] [runalways] [showalloutput]>

<args>

<arg name="Filepath">[REQ] The path to the database creation executable</arg>

<arg name="SqlHost">[REQ] The SQL Server host machine</arg>

<arg name="DatabaseName">[REQ] The name of the database to be created</arg>

<arg name="DatabaseWizardConfigFile">[REQ] The path to the database wizard
configuration file</arg>

<arg name="SaveServerConfiguration">[OPT] Defines whether we want to save changes to
the server configuration</arg>

</args>

</task>

</TaskChain>
```

TASKCHAIN OPTIONAL ATTRIBUTES:

failonerror="true|(false)" - If true, terminates the chain on any error

TASK OPTIONAL ATTRIBUTES:

id="friendly name" - A unique identifier for a task

TASK OPTIONAL ATTRIBUTES WITH VARIABLE SUPPORT:

enabled="(true)|false" - If true, this task will be skipped

fatal="(true)|false" - If true, any error in this task will terminate the run

runalways="true|(false)" - If true, this task will ALWAYS run even if earlier tasks failed

showalloutput="true|(false)" - If true, shows all output, otherwise just errors

Retrieving a List of Available Tasks

To retrieve a list of available tasks from an extension assembly

- `C:\Users\Ian\Desktop>ToolBox -list ToolBox.Tasks.Build.dll`

TOOLBOX v1.1.1.115

[C:\Program Files\Quest\Enterprise Reporter\Server\ToolBox\Toolbox.Tasks.Build.dll]

[BUILD]

JavaScript - Runs JavaScript

MsBuild - Performs solution build using MSBuild

NuGet - Performs NuGet operations

PseudoLocalize - Performs resource pseudo localization

SetPkgVersion - Performs search through folders and changes the package versions

Found 5 tasks

It is possible to query more than one assembly at a time by separating the names of the assemblies. If the assemblies do not lie in the current path, the names must be fully qualified:

To query more than one assembly at a time

-
  ```
  c:\Projects\ToolBox\bin>toolbox -list
  ToolBox.Tasks.Build.dll;ToolBox.Tasks.Computer.dll
  ```

TOOLBOX v1.1.1.115

[C:\Program Files\Quest\Enterprise Reporter\Server\ToolBox\Toolbox.Tasks.Build.dll]

[BUILD]

JavaScript - Runs JavaScript

MsBuild - Performs solution build using MSBuild

NuGet - Performs NuGet operations

PseudoLocalize - Performs resource pseudo localization

SetPkgVersion - Performs search through folders and changes the package versions

[C:\Program Files\Quest\Enterprise Reporter\Server\ToolBox\Toolbox.Tasks.Computer.dll]

[COMPUTER]

RestartComputer - Restarts the specified computer

SetComputerServiceLogonRights - Sets computer service logon rights

SetServiceAccount - Sets the NT Service account and password

StartService - Starts NT Service

TakeScreenshot - Takes a screenshot of the entire screen or a specific window

Found 10 tasks

Registering Task Assemblies

In order to make use of the tasks within an assembly, ToolBox must first know about the tasks within that assembly. If, for example, you have created a new assembly, or added tasks to an existing assembly, then you need to place that assembly into the same folder as TOOLBOX.EXE and then execute a “toolbox –register” operation. This will force ToolBox to enumerate all assemblies matching the file pattern, “ToolBox.Tasks.*.dll” and interrogate those assemblies for tasks. The result will be an output file containing a mapping named, “taskcatalogue.json” which acts as an index and lookup for the ToolBox when executing tasks.

To register task assemblies

-
  ```
  c:\Projects\ToolBox>toolbox -register
  ```

TOOLBOX v1.1.1.115

ToolBox.Tasks.ActiveDirectory.dll (found 10 tasks)

ADContactTask

BackupActiveDirectoryTask

EnableActiveDirectoryAccountTask

JoinDomainTask

LeaveDomainTask

PopulateAdCountryStateTask

PopulateAdUsersGroupsComputersTask

PopulateAdUsersTask

RenameActiveDirectoryOUsTask

RestoreActiveDirectoryTask

ToolBox.Tasks.Build.dll (found 5 tasks)

JavaScriptTask

MsBuildTask

NuGetTask

PseudoLocalizeTask

SetPkgVersionTask

ToolBox.Tasks.Computer.dll (found 5 tasks)

RestartComputerTask

SetComputerServiceLogonRightsTask

SetServiceAccountTask

StartServiceTask

TakeScreenshotTask

ToolBox.Tasks.Database.dll (found 43 tasks)

AddUpgradeScriptsToInstallerTask

BackupDatabaseTask

CheckLogErrorsTask

CompareIndexNamesTask

CompareRolePermissionsTask

CompareRoutinesTask

CompareTableContentTask

CompareTablesColumnsTask

CompareTablesTask

CompareViewsColumnsTask

CompareViewsTask

CreateDatabaseTask

CreateLocalDbInstanceTask

DeleteLocalDbInstanceTask

DropDatabaseTask

DumpReporterLogsToDatabaseTask

EnumerateIndicesTask

EnumerateRoutinesTask

EnumerateTablesTask

EnumerateViewsTask

ExecuteFileQueryTask

ExecuteQueryTask

ExportDataToCsvTask

ExportDataToSqliteTask

ExportDataToSqlServerTask

ExportHttpToSqlServerTask

ExportJsonToSqlServerTask

ExtractLastJobExecutionTimeTask

FixDatabaseScriptTask

GenerateReportsScriptTask

GenerateResetSecurityScriptTask

GenerateUpgradeScriptStubTask

GetMissingTablesTask

RestoreDatabaseTask

SQLDataCompareProjectResultTask

SQLDataCompareProjectTask

SQLSchemaCompareProjectTask

StartLocalDbInstanceTask

StopLocalDbInstanceTask

UpgradeDatabaseTask

ValidateIndexNamesTask

ValidateReportIdsTask

ValidateRolesTask

ToolBox.Tasks.DataViz.dll (found 2 tasks)

DumpRunInfoTask

VisualiseJobRunTask

ToolBox.Tasks.EnterpriseReporter.dll (found 2 tasks)

SetSystemCredentialsTask

ValidateLicenceFilesTask

ToolBox.Tasks.Files.dll (found 26 tasks)

AppendFilesTask

CompareTextFilesTask

ConvertRTFtoTextTask

CopyFilesFromShareTask

CopyFilesTask

CopyMatchingFilesTask

CreateFolderTask

CreateZipTask

DeleteFilesTask

DeleteFolderTask

FileExistsTask

FileSizeTask

JsonFileCompareTask

ReplaceTextInFileTask

ReplaceVersionInFileNameTask

ReplaceVersionInFileTask

ReSignStrongNameTask

ResourceUpdateTask

SevenZipTask

SignToolTask

SignVerifyTask

UnZipTask

ValidateAssemblyVersionsTask

ValidatecsvFormatTask

ValidateInstalledFilesDetailsTask

ValidateInstalledFilesSignedTask

ToolBox.Tasks.Images.dll (found 1 task)

IconGeneratorTask

ToolBox.Tasks.PowerShell.dll (found 6 tasks)

ConfigurePowerShellTask

CopyFilesFromRemoteTask

CopyFilesToRemoteTask

ExecutePowerShellCommandTask

ExecutePowerShellTask

SetPowerShellUnrestrictedTask

ToolBox.Tasks.Processes.dll (found 3 tasks)

```
EchoTask
ExecuteProcessTask
SleepTask

ToolBox.Tasks.Security.dll (found 2 tasks)
AslrCheckTask
HashCodeTask

ToolBox.Tasks.TFS.dll (found 5 tasks)
TFSAddTask
TFSChangesetAssembliesTask
TFSCheckOutTask
TFSQueueBuildTask
TFSSyncTask

ToolBox.Tasks.Upgrade.Server.dll (found 1 task)
UpgradeServer32Task

ToolBox.Tasks.VisualStudio.dll (found 5 tasks)
FindDeadProjectsTask
LineCountTask
StaticAnalysisTask
VSExternalToolsTask
VSInstallVsixTask

ToolBox.Tasks.VMWare.dll (found 6 tasks)
VMDeleteSnapshotTask
VMPowerOffTask
VMPowerOnTask
VMRevertSnapshotTask
VMRunInGuestTask
VMTakeSnapshotTask

Added 15 assemblies containing 122 tasks.
```

Executing Tasks Using Chain Files

The tool provides a number of tasks which can be chained together using what is called a “chain file”; each task in the chain file will be executed, one at a time, in sequence from top to bottom. A chain file is simply an XML file with the following format:

```
<?xml version="1.0" encoding="utf-8" ?>
<TaskChain [failonerror]>
  <task name="Task1" [enabled] [id] [fatal] [showalloutput]>
    <args>
      <arg name="X”>X-value</arg>
    </args>
  </task>
  ...
  <task name="TaskN" [enabled] [id] [fatal] [showalloutput]>
    <args>
      <arg name="X”>X-value</arg>
    </args>
  </task>
</TaskChain>
```

The documentation for each task provides a complete and legal chain file as part of its output. This allows the tool to be self-documenting. Simply copy the complete block of XML from any task into a new text file and then add the appropriate <task> blocks for any other tasks you wish to execute. Once that is done, replace the arguments with your own values and the chain file is ready for use.

To execute a chain file from the command line

- `C:> ToolBox –execute chainfile.xml`

To execute a chain file using drag and drop

- Drag and drop the file on to the ToolBox.exe file.

Executing a Chain File in Debug Mode

A new execution mode has recently been added. We call this debug mode and it executes a chain file in exactly the same way as the standard Execute Mode but it dumps the contents of all variables (declared, injected and task-local) prior to executing each task. This can be useful for tracking down errors.

To execute a chain file in debug mode

- `C:> ToolBox –debug chainfile.xml`

How ToolBox Locates Chain Files

There are two ways that the ToolBox will locate chain files:

1 Specify a path to the file:

```
C:> ToolBox –debug c:\myscripts\chainfile.xml
```

2 Look in the current working directory and then look in the directory specified in the environment variable TOOLBOX_SCRIPTS:

```
    C:> ToolBox –debug chainfile.xml
```

Note that this is the order in which the tool will search. If there is a file named, “chainfile.xml” in both the current working directory AND in the directory referenced in TOOLBOX_SCRIPTS, then it will be the script in the WORKING DIRECTORY that will be executed. In the event a file cannot be located, then ToolBox will make a best-guess based on a Levenshtein Distance calculation for all files in the current working directory and the TOOLBOX_SCRIPTS directory.

A Real Example

Follow is a real example of a chain-file.. The actual file contains many more files, but this stripped-down version is sufficient to explain all the details of a chain file.

```
<!--This file is named ‘SampleValidateProjectDetails.xml’ -->
<?xml version="1.0" encoding="utf-8" ?>
<TaskChain>
    <vars>
        <var name="!(serverpath)">C:\Program Files\Quest\Enterprise
Reporter\Server</var>
        <var name="!(configpath)">C:\Program Files\Quest\Enterprise
Reporter\ConfigurationManager</var>
        <var name="!(reportpath)">C:\Program Files\Quest\Enterprise
Reporter\ReportManager</var>
        <var name="!(asmversion)">3.1.0.0</var>
        <var name="!(fileversion)">3.1.0.1736</var>
    </vars>

    <!-- Server -->
    <task name="ValidateProjectDetailsTask" id="server" fatal="false"
enabled="false">
        <args>
            <arg name="AssemblyVersion">!(asmversion)</arg>
            <arg name="FileVersion">!(fileversion)</arg>
            <arg name="CompanyName">Quest</arg>
            <arg name="Files">
                !(serverpath)\Category.dll,
                !(serverpath)\DatabaseWizard.exe,
                !(serverpath)\DBConsole.exe,
                !(serverpath)\DbLibrary.dll
            </arg>
        </args>
    </task>
    <!-- Configuration Manager -->
    <task name="ValidateProjectDetailsTask" id="configuration manager"
fatal="false">
        <args>
            <arg name="AssemblyVersion">!(asmversion)</arg>
            <arg name="FileVersion">!(fileversion)</arg>
            <arg name="CompanyName">Quest</arg>
            <arg name="Files">
                !(configpath)\CollectorUI\Quest.Reporter.Consoles.Discovery.
CollectorViews.dll,
                !(configpath)\ConfigurationManager.exe,
                !(configpath)\Quest.Common.AMConnector.dll,
                !(configpath)\Quest.Reporter.Configuration.PowerShell.dll
            </arg>
        </args>
    </task>

    <!-- Report Manager -->
    <task name="ValidateProjectDetailsTask" id="report manager" fatal="false">
        <args>
            <arg name="AssemblyVersion">!(asmversion)</arg>
            <arg name="FileVersion">!(fileversion)</arg>
            <arg name="CompanyName">Quest</arg>
            <arg name="Files">
                !(reportpath)\Category.dll,
                !(reportpath)\Quest.Common.AccessManagerHelper.dll,
                !(reportpath)\Quest.Common.AMConnector.dll,
                !(reportpath)\Quest.Common.SqlParsing.dll
            </arg>
        </args>
    </task>
</TaskChain>
```

XML Comments

The chain file may contain XML comments. A chain-file is simply an XML file, so anything that is valid inside an XML file is valid in a chain file.

Task ID

Each task has an “id” attribute. This makes it possible to distinguish the exact usage of a task within a chain file. Looking at the example above, the chain file has used the same “ValidateProjectDetailsTask” three times. If an error were to occur, it would show that it had occurred in a task of type “ValidateProjectDetailsTask” but it might be difficult to identify exactly which call was causing the error. It is possible to narrow down an error to the specific usage of a task using the unique identifier on each of the tasks. The ID, “env” is reserved for the system to use when importing system environment variables.

Enabled

BOOLEAN - By default, ToolBox will run all tasks found within a chain-file. Sometimes, it is convenient to be able to prevent a task from running. In such cases, it is not necessary to resort to removing the task or commenting it out as the enabled attribute can be used to keep a specific task or hierarchy of tasks (see Advanced Topics) from running.

Show All Output

BOOLEAN - By default, ToolBox displays only error messages. By specifying a value of true for this attribute, all output will be displayed (both success and errors).

Fatal

BOOLEAN - This attribute determines whether or not the task should terminate the run; the default is to terminate the run whenever a task encounters an error. This behavior can be over-ridden by marking non-essential tasks as non-fatal.

Passing Multiple Arguments

Each “Files” argument contains multiple filenames separated by commas. Standard rules for parsing CSV files apply. A future enhancement will allow multi-value arguments to separate items within an <item></item> tag.

3

Variables

Variable names are case-insensitive and come in four different flavours:

Declared Variables

Declared variables are placed into a <vars></vars> block at the head of a chain file.

```
<?xml version="1.0" encoding="utf-8" ?>
<TaskChain>
    <vars>
        <var name="!(questroot)">C:\Program Files\Quest\Enterprise Reporter<var>
        <var name="!(serverpath)">!(questroot)\Server</var>
        <var name="!(configpath)">!(questroot)\ConfigurationManager</var>
        <var name="!(reportpath)">!(questroot)\ReportManager</var>
        <var file="c:\globalvariables.xml"/>
    </vars>
        …
</TaskChain>
```

Chain files may contain variables in a <vars> section at the head of the file. This allows using these variables within the chain file to avoid typing repetitive text. In the example above, they have been used to represent directories off a common root path. Using this approach not only makes it easy to replace a common root path but makes the script file easier to read and reduces the likelihood of introducing errors by missing paths in need of replacement.

Declared variables may make use of system environment variables by specifying a value in the form of a variable with the “env.” prefix. In the example below, !(windows) will be set to the value of the environment variable, “WINDIR”.

```
     <var name="!(windows)">!(env.windir)<var>
```

Variables defined in other chain files can also be included.

```
     <var file="c:\globalvariables.xml"/>
```

Declared variables are GLOBAL and visible to ALL tasks in a chain file.

Injected Variables

Variables in a chain file may be injected from outside simply by specifying the replacement value on the command-line. For example, in the chain file shown above, the variable !(questroot) is defined as C:\Program Files\Quest\Enterprise Reporter.

If a new value is passed to this file during execution, using a command-line such as that shown below, the value in the file will be over-written in favor of the new value.

```
C:> ToolBox -execute SampleValidateProjectDetails.xml "!(questroot)=D:\Quest"
```

In this case, !(questroot) would be redefined as D:\Quest and the other variables would be updated accordingly.

One of the primary cases for using injected variables is when using the ToolBox in conjunction with TFS build. TFS defines various variables that can vary from one build machine to the next. By injecting variables the exact same chain file can be re-used and simply over-write those variables that need to be changed.

Injected variables are GLOBAL and visible to ALL tasks in a chain file.

Task-Level Variables

When defining a task, values for the arguments associated with that task often must be provided. Arguments are, themselves, a form of variable. The primary difference (and the reason we distinguish between task arguments and variables) is that arguments to tasks are visible ONLY to the specific task (and instance of that task) that declares them and for which values are provided. Variables may be used in conjunction with task arguments as in the example below:

```
<?xml version="1.0" encoding="utf-8" ?>
<TaskChain>
  <vars>
    <var name="!(message)">This is a test message</var>
  </vars>

  <task name="Echo" id="msg">
    <args>
      <arg name="Message">The message variable contains: !(message)</arg>
    </args>
  </task>
</TaskChain>
```

The output of this task is:

```
TOOLBOX 3.1.0.1736
Identity:  HALILITTLEWOOD\Ian
Host:      HALILITTLEWOOD
Name:      HALILITTLEWOOD
Exe:       c:\Projects\Toolbox\bin\ToolBox.exe
```

```
Script:    c:\Projects\testecho.xml
Arguments: -execute c:\Projects\testecho.xml
--------------------------------------------------------------------------------
START [Echo (msg)]
The message variable contains: This is a test message
END [Echo (msg)]    OK                                00m 00s 006ms
--------------------------------------------------------------------------------
SUMMARY
Date:     May 10, 2018
Duration: 00m 00s 006ms
[Echo (msg)]   OK    01:39:12.197 - 01:39:12.204 (00m 00s 006ms)

SUCCESSFUL RUN
```

Output Variables

Output variables allow a task to produce named variables that can be consumed by other tasks.

Example: Using Output Variables

```
<?xml version="1.0" encoding="utf-8" ?>
<TaskChain>
  <task name="Echo" id="msg01">
    <args>
      <arg name="Message">This is a test message</arg>
    </args>
    <outargs>
      <outarg name="MessageLength" description="The length of the message"/>
    </outargs>
  </task>

  <task name="Echo" id="msg02">
    <args>
      <arg name="Message">Message msg01 is !(msg01.MessageLength) characters
long</arg>
    </args>
    <outargs>
      <outarg name="MessageLength" description="The length of the message"/>
    </outargs>
  </task>

  <task name="Echo" id="msg03">
    <args>
      <arg name="Message">Message msg02 is !(msg02.MessageLength) characters long</arg>
    </args>
  </task>
</TaskChain>
```

This chain file produces the following output:

```
TOOLBOX 3.1.0.1736
Identity:  HALILITTLEWOOD\Ian
Host:      HALILITTLEWOOD
Name:      HALILITTLEWOOD
Exe:       c:\Projects\Toolbox\bin\ToolBox.exe
Script:    c:\Projects\testecho.xml
Arguments: -execute c:\Projects\testecho.xml
-------------------------------------------------------------------------------
START [Echo (msg01)]
This is a test message
END [Echo (msg01)]    OK                              00m 00s 007ms
-------------------------------------------------------------------------------
START [Echo (msg02)]
Message msg01 is 22 characters long
END [Echo (msg02)]    OK                              00m 00s 000ms
-------------------------------------------------------------------------------
START [Echo (msg03)]
Message msg02 is 35 characters long
END [Echo (msg03)]    OK                              00m 00s 001ms
-------------------------------------------------------------------------------
SUMMARY
Date:     May 03, 2017
Duration: 00m 00s 020ms
[Echo (msg01)]   OK    01:42:34.053 - 01:42:34.060 (00m 00s 007ms)
[Echo (msg02)]   OK    01:42:34.066 - 01:42:34.066 (00m 00s 000ms)
[Echo (msg03)]   OK    01:42:34.072 - 01:42:34.073 (00m 00s 001ms)

SUCCESSFUL RUN
```

Example Output Explained

These tasks contain an <outargs> block. This is optional in the chain file but should generally be left in place purely for documentation purposes. The output variable will be produced even if the <outargs> block is not included.

Each task also has the optional id attribute specified. This is required so that each output variable can be uniquely annotated. Consider what would happen if the attribute were omitted; the task with ID “msg01” would produce an output variable named “!(MessageLength)” but the next task would execute and over-write the variable with a new value. This can be avoided by prefixing the id of the task that produces the output variable so that the values can be preserved. This means that a task with id=”msg01” will produce a variable named, “!(msg01.MessageLength)”

Output variables are GLOBAL and visible to ALL tasks in a chain file.

Example: Simple Chain File

Following is a very simple example of a chain file. This chain file waits for 1 second, then 5 seconds, then 10 seconds:

```
<?xml version="1.0" encoding="utf-8" ?>
<TaskChain>
  <task name="Sleep" id="1s">
    <args>
      <arg name="TimeInSeconds">1</arg>
    </args>
  </task>
  <task name="Sleep" id="5s">
    <args>
      <arg name="TimeInSeconds">5</arg>
    </args>
  </task>
  <task name="Sleep" id="10s">
    <args>
      <arg name="TimeInSeconds">10</arg>
    </args>
  </task>
</TaskChain>
```

The output for this chain file looks like this:

```
TOOLBOX 3.1.0.1736
Identity:  HALILITTLEWOOD\Ian
Host:      HALILITTLEWOOD
```

```
Name:     HALILITTLEWOOD
Exe:      c:\Projects\Toolbox\bin\ToolBox.exe
Script:   c:\Projects\testsleep.xml
Arguments: -execute c:\Projects\testsleep.xml
--------------------------------------------------------------------------------
START [Sleep (1s)]
Sleeping for 1 seconds
END [Sleep (1s)]    OK                              00m 01s 008ms
--------------------------------------------------------------------------------
START [Sleep (5s)]
Sleeping for 5 seconds
END [Sleep (5s)]    OK                              00m 05s 001ms
--------------------------------------------------------------------------------
START [Sleep (10s)]
Sleeping for 10 seconds
END [Sleep (10s)]    OK                              00m 10s 002ms
--------------------------------------------------------------------------------
SUMMARY
Date:     May 03, 2017
Duration: 00m 16s 022ms
[Sleep (1s)]    OK    01:47:23.633 - 01:47:24.642 (00m 01s 008ms)
[Sleep (5s)]    OK    01:47:24.648 - 01:47:29.649 (00m 05s 001ms)
[Sleep (10s)]   OK    01:47:29.653 - 01:47:39.656 (00m 10s 002ms)

SUCCESSFUL RUN
```

In the output, the first thing is the name of the chain file that was executed ("sleepychain.xml").

Next, there are three START/END blocks showing that a task has started (START), which task has started (Sleep) and the identifier of that task (1s). If the task produced any output, it would appear between the START and END blocks. Finally, an END block shows the associated task (Sleep), the status of the task on completion (SUCCESS) and the total run-time of the task.

Finally, there is a summary of the entire chain file.

4

Advanced Topics

Force Tasks to Run in Parallel

One ToolBox features provides the ability to wrap a set of one or more tasks in <parallel>…</parallel> to run the tasks in the set in parallel. Parallel has a single attribute, "threads" which accepts an integer detailing the maximum number of threads that should be running concurrently when executing the task. The default behavior is to immediately spawn each task in its own thread.

```
<parallel threads="n">
```

Force Parallel Tasks to Run Sequentially

The ToolBox also provides the ability to wrap a set of tasks in <sequential>…</sequential>. At first glance, this may seem to be redundant; after-all, a Task-Chain is, by definition, a set of tasks to be executed sequentially. However, <sequential/> forces certain tasks within a <parallel/> block to execute in sequence while other tasks in the same block process entirely in parallel.

```
<sequential>
```

Example: A Chain-File with Sequential Tasks

This first example is a traditional chain-file. Each task will execute in sequence. Total run-time will be approximately 30 seconds.

```
<?xml version="1.0" encoding="utf-8" ?>
<TaskChain>
  <task name="Echo" id="Start">
    <args>
      <arg name="Message">Start</arg>
    </args>
  </task>

  <task name="Sleep" id="A">
    <args>
```

```
      <arg name="TimeInSeconds">10</arg>
    </args>
  </task>

  <task name="Sleep" id="B">
    <args>
      <arg name="TimeInSeconds">10</arg>
    </args>
  </task>

  <task name="Sleep" id="C">
    <args>
      <arg name="TimeInSeconds">10</arg>
    </args>
  </task>

  <task name="Echo" id="Stop">
    <args>
      <arg name="Message">Start</arg>
    </args>
  </task>
</TaskChain>
```

Graphically, this execution looks like so, where time runs down the vertical:

Table 1. Chain-file sequential task execution

Start
A
B
C
Stop

Example: A Chain-File with Parallel Tasks

This slightly re-written example will cause all tasks to be executed in parallel. Total run-time will be approximately 10 seconds.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<TaskChain>
  <task name="Echo" id="Start">
    <args>
      <arg name="Message">Start</arg>
    </args>
  </task>

  <parallel>
    <task name="Sleep" id="A">
      <args>
        <arg name="TimeInSeconds">10</arg>
      </args>
    </task>

    <task name="Sleep" id="B">
      <args>
        <arg name="TimeInSeconds">10</arg>
      </args>
    </task>

    <task name="Sleep" id="C">
      <args>
        <arg name="TimeInSeconds">10</arg>
      </args>
    </task>
  </parallel>

  <task name="Echo" id="Stop">
    <args>
      <arg name="Message">Stop</arg>
    </args>
  </task>
 </TaskChain>
```

Graphically, this looks like so, where time runs down the vertical:

Table 2. Chain-file parallel task execution

Start

Table 2. Chain-file parallel task execution

A	B	C
	Stop	

Example: A Chain-File with Both Sequential and Parallel Tasks

This example shows how sequential blocks can be useful:

```
<?xml version="1.0" encoding="utf-8" ?>
<TaskChain>
  <task name="Echo" id="Start">
    <args>
      <arg name="Message">Start</arg>
    </args>
  </task>

  <parallel>
    <task name="Sleep" id="A">
      <args>
        <arg name="TimeInSeconds">10</arg>
      </args>
    </task>

    <task name="Sleep" id="B">
      <args>
        <arg name="TimeInSeconds">10</arg>
      </args>
    </task>

    <sequential>
      <task name="Sleep" id="C">
        <args>
          <arg name="TimeInSeconds">2</arg>
        </args>
      </task>

      <task name="Sleep" id="D">
        <args>
```

```
         <arg name="TimeInSeconds">2</arg>
       </args>
     </task>

     <task name="Sleep" id="E">
       <args>
         <arg name="TimeInSeconds">2</arg>
       </args>
     </task>
   </sequential>
 </parallel>

 <task name="Echo" id="Stop">
   <args>
     <arg name="Message">Stop</arg>
   </args>
 </task>
</TaskChain>
```

Graphically, this looks like so, where time runs down the vertical:

Table 3. Chain-file with sequential and parallel task execution

	Start	
A	B	C D E
	Stop	

How is this useful? Well, one of the build tasks is used to build the Enterprise Reporter codebase. Some solutions are dependent on others, but others are independent. For example, each of the collectors takes approximately 1 minute to build. The collectors have dependencies on some solutions and other solutions have dependencies on the collectors, but the collectors have no dependencies on each other, so they can be built in parallel.

About us

Quest provides software solutions for the rapidly-changing world of enterprise IT. We help simplify the challenges caused by data explosion, cloud expansion, hybrid datacenters, security threats, and regulatory requirements. We are a global provider to 130,000 companies across 100 countries, including 95% of the Fortune 500 and 90% of the Global 1000. Since 1987, we have built a portfolio of solutions that now includes database management, data protection, identity and access management, Microsoft platform management, and unified endpoint management. With Quest, organizations spend less time on IT administration and more time on business innovation. For more information, visit www.quest.com.

Technical support resources

Technical support is available to Quest customers with a valid maintenance contract and customers who have trial versions. You can access the Quest Support Portal at https://support.quest.com.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request.
- View Knowledge Base articles.
- Sign up for product notifications.
- Download software and technical documentation.
- View how-to-videos.
- Engage in community discussions.
- Chat with support engineers online.
- View services to assist you with your product.