Automated Build Studio

by SMARTBEAR

USER MANUAL

Copyright Notice

Table of Contents

Introduction

Overview

Automated Build Studio by SmartBear Software

Automated Build Studio is a powerful **build and release management system** that provides an easy way to automate the build, test and release processes of software projects. Its intuitive interface and unprecedented flexibility allows even inexperienced users to create complex visual macros that can automate repetitive or day-to-day tasks with the single click of a button.

This topic is devoted to the main functions of Automated Build Studio. It provides you with a simple conceptual map of the product. To learn how to use these functions, see *Getting Started*. A good way to learn how to use the product is to explore samples that come with Build Studio. See *Automated Build Studio Samples* for more information on them.

Why Automated Builds?

In the current state of software development, with projects becoming more and more complex, building, testing and releasing of software projects consumes an ever-increasing amount of time and resources. Building the application manually quite often becomes tedious, prone to errors and inefficient in terms of both time and money. A typical build process includes a number of steps:

- Getting latest versions of source files from a version control system,
- Specifying build options,
- Recompiling libraries and packages that are used by the application,
- Building the application,
- Building the installation,
- Notifying co-workers that a new build is available,
- Starting the testing process, and so forth.

It is quite easy to make a mistake when performing these steps. For example, you may specify wrong build version number or compile the application using inappropriate options.

Automated Build Studio lets you optimize the build, test and release processes and allows you to focus on more important issues. Using Build Studio, you spend a certain amount of time on a task only once: on creating a macro for that task. A macro unifies elementary operations, for example, Copy File(s), Get Latest Version From VSS, Compile Visual C# 2010 Project, Compile Delphi XE4 Project, Send E-Mail, etc., that let you perform each step of the process you are automating. After the macro is ready, you can run it either manually, or at certain points in time. That is, you commission Build Studio with all the work and responsibility: it will automatically run the macro whenever you need.

Major Features

Among the many features that make Automated Build Studio the ultimate build and release management tool are:

- **Visually constructed macros**. As we have said above, a macro includes operations each of which is responsible for a separate step of the process you want to automate. Even if you are not familiar with programming, you will find creating macros very easy with the visual macro designer of Automated Build Studio.
- **A wide range of available operations with support for popular modern tools**. Automated Build Studio includes more than 800 built-in operations that let you automate almost any task: taking source files from source control, compiling applications, uploading files to FTP servers, running and stopping virtual and cloud machines and so on. For a list of all available operations, see the *List of Operations* help topic.

  To perform specific actions that cannot be performed by using standard operations, you can write custom script routines in VBScript, JScript and DelphiScript, or you can create custom operations.

  Automated Build Studio offers native integration with other SmartBear's testing profiling and team collaboration products allowing you to fully automate your build, profile, test and deploy processes. You can find these operations in the SmartBear & AutomatedQA Tools category.
- For a full list of compilers, source control systems, test tools and other applications supported by Automated Build Studio, see *Supported Tools* in the Automated Build Studio help.
- **Distributed builds**. Automated Build Studio provides you with the ability to distribute the execution of macro operations among multiple computers. Using this feature, you can dramatically reduce the project build time, perform multi-platform builds, start automated tests on multiple computers in parallel and so on. See *Distributed Builds - Overview*.

  Besides physical computers, you can also use virtual and cloud machines in your distributed builds. Automated Build Studio supports the most popular virtual and cloud environments and provides special operations for managing virtual and cloud computers directly from your macros.
- **Remote macro execution control.** Automated Build Studio includes special features that let you easily run macros located on remote computers and monitor these runs. Using Automated Build Studio Web Interface you can control remote macros through your web browser, even without installing Automated Build Studio on your machine. For complete information, see the *Automated Build Studio Client/Server - Overview* help topic.

  You can also start and control remote builds directly from your macros by using operations of the Remote Macro Execution category.
- **Concurrent execution of operations.** Automated Build Studio lets you organize threads in your macro and run various operations concurrently. This can significantly speed up the macro execution, especially on computers with multiple CPUs. For complete information on using this feature, see *Concurrent Execution of Operations*.
- **Scheduled and event-triggered macro runs**. Automated Build Studio has a built-in Task Scheduler that lets you schedule macro runs at your desire. You can also configure Automated Build Studio so that it automatically runs the appropriate macro when some event occurs, for instance, when a source file of your project is changed under source code control. See *Continuous Integration*.
- **Advanced macro run monitoring and charts**. Automated Build Studio generates detailed logs of macro runs. It automatically compares results of the current macro run with the last successful

run, thus letting you know if something is going wrong. For information on macro logs, see *Logging Macro Runs*.

Automated Build Studio also collects information about macro runs performed on your computer and other computers within your network and displays this information in a graphical form in the **Charts** panel. Charts can help you analyze the ratio of successful and failed runs, the duration of macro runs and the frequency of macro runs over a desired period of time. For more information, see *Charts - Overview* in the Automated Build Studio help.

- **Automatic notifications.** Your macros can automatically notify you about the results of their execution. Automated Build Studio offers a number of operations for sending notifications via the "net send" command, Windows Messenger, MSN Messenger, Windows Live Messenger, ICQ and e-mail. You can also e-mail reports with attached execution logs. See the *Sending Notifications* help topic.
- **Integration with Microsoft Visual Studio.** Automated Build Studio can be tightly integrated into the Microsoft Visual Studio IDE. This integration offers developers full control over Automated Build Studio from within the Visual Studio IDE. You can also run Build Studio macros directly from MSBuild projects and Visual Studio Team Explorer (see *Running Automated Build Studio Macros From MSBuild Projects* and *Running Automated Build Studio Macros From Visual Studio Team Explorer* help topics).
- **Single macro for generating various application versions.** Quite often, it is needed to build several versions of an application from the same sources; for example, the debug, demo and release versions of the product. Using macro configurations, you can have a single macro that will generate all the desired versions. There is no need to create three macros. You can create only one macro and choose the desired configuration before the run. See *Macro Configurations* in this document.
- **Command line and COM automation**. Through the command line, you can control and run Automated Build Studio in a completely hands-off fashion inside a make file or a batch file. The Automated Build Studio installation package includes a console resource-friendly utility, AutomatedMacroPlayer.exe, that simply running of macros created in Automated Build Studio. This executable uses the same command-line arguments which Build Studio uses. For information on the arguments, see *Command-Line Interface* section in the Automated Build Studio help.
- Automated Build Studio can also be automated via COM from third-party applications or scripts defined in other programs. For more information on Build Studio's COM interfaces, see *Working With Automated Build Studio via COM* section in the Automated Build Studio help.
- **Macro debugger.** To help you find and eliminate errors, Automated Build Studio includes a built-in macro debugger that supports breakpoints, stepping over operations, watches, expression evaluation and other debugging features. See the *Debugging Macros in Automated Build Studio* help topic.

Plug-Ins

Automated Build Studio has an open, COM-based architecture. Most of the features you see in the product are supported through plug-ins. When the plug-in is meant to remain for the life of the current version, it is made an internal part of the product executable. When it is possible that the plug-in could be updated, it is left external (such as the database-access plug-ins). Others will be added to SmartBear Web site (http://smartbear.com), and users can write their own plug-ins using the supplied interface documentation and libraries (for instance, user can create plug-ins that will add certain operations to Build Studio).

System Requirements

Supported Operating Systems

- Microsoft Windows 8 (both 32-bit and 64-bit editions).
- Microsoft Windows Server 2012.
- Microsoft Windows 7 (both 32-bit and 64-bit editions).
- Microsoft Windows Vista (both 32-bit and 64-bit editions).
- Microsoft Windows Server 2008 (both 32-bit and 64-bit editions).

  Microsoft Windows Server 2008 R2 is also supported.
- Microsoft Windows XP (both 32-bit and 64-bit editions).
- Microsoft Windows Server 2003 (both 32-bit and 64-bit editions).
- Microsoft Windows 2000 with Service Pack 4 or later.

In order for Automated Build Studio to be able to function on **Windows Server 2008 R2**, the WoW64 component is required. The Server Core installation option for Windows Server 2008 R2 does not install WoW64 by default, so, you should install it yourself.

Minimal System Requirements

- Intel Pentium II 450 MHz or higher.
- Microsoft Windows 2000 with Service Pack 4 or later.
- Microsoft Internet Explorer 5.0 or later.
- Microsoft Internet Information Services ver. 6.0 or later.
- Microsoft .NET Framework 2.0 or later.
- 128 MB of RAM.
- 100 MB hard disk space.
- VGA (640 × 480) or higher resolution monitor.
- Mouse or other pointing device.

Note: Microsoft Internet Information Services should be installed prior to Microsoft .NET Framework.

Recommended System Requirements

- Intel Pentium III 800 MHz or higher.
- Microsoft Windows XP or later.
- Microsoft Internet Explorer 5.0 or later.
- Microsoft Internet Information Services ver. 6.0 or later.
- Microsoft .NET Framework 2.0 or later.
- 256 MB of RAM.
- 150 MB hard disk space.
- SXGA (1280 × 1024) or higher resolution monitor.
- Mouse or other pointing device.

Note: Microsoft Internet Information Services should be installed prior to Microsoft .NET Framework.

Additional Requirements

To install Automated Build Studio Web Interface, the following software needs to be installed on the computer:

- Microsoft Windows 2000 Professional with Service Pack 4 or later.
- Microsoft Data Access Components 2.8 or later.
- Adobe Flash Player 8 or later.

Automated Build Studio's demo version includes all the features available in the registered version, but has a 100-operations-per-macro limit and can be used only during a 30-day trial period.

Getting Support

If you have questions, problems or just need help with Automated Build Studio, contact our **support team** or try to search for the needed information using the help resources located on SmartBear's Web site (forums, blogs, technical papers).

To submit your question to the support team

1. Select **Help | Contact Support Team** from Automated Build Studio's main menu. This will invoke the *Contact Support Team* dialog.
2. (Optional) In the dialog, select the **Attach system information to my support request** check box if you want Automated Build Studio to collect some system information and attach it to your request. To preview the collected system information and make a decision whether you want to send it to our support team, click the **View the system information file contents** link at the bottom of the dialog. This will invoke the **System Information** dialog, in which you can preview the information to be attached to the request.

   If you do not want to attach this information to your request, clear the check box. However, we recommend that you attach this information since it can help our support team solve your problem quicker.

3. Click the **Continue** button in the Contact Support Team dialog. Automated Build Studio will load the web page with the Contact Support Form from the SmartBear Web site to your web browser:

   - **http://support.smartbear.com/message/?prod=Automated%20Build%20Studio**

   Fill in the required fields in this web form. Note that your contact information, the product name and version are specified automatically in the appropriate fields when you proceed to this web page from the Contact Support Team dialog. You only need to describe your problem in the appropriate fields.

   Click **Submit** in the Contact Support Form to submit the request.

   The support team will answer you via e-mail and all further communication will be made via e-mail. However, to start the conversation, please use the Contact Support Team dialog in Automated Build Studio and the Contact Support Form on our web site.

For information on our support policies, please visit our web site **http://support.smartbear.com/**.

More support resources

You can also ask questions, search for answers, exchange comments and suggestions on our **forums**:

- **http://community.smartbear.com/**

You may find answers to your question in the list of the frequently asked questions which is available at:

- **http://support.smartbear.com/faq**

Learn more about using Automated Build Studio from **technical papers** and **blogs** published at:

- **http://support.smartbear.com/articles**
- **http://blog.smartbear.com**

Make sure you regularly visit the SmartBear Web site, **http://smartbear.com**, where you will find:

- News
- More recent support options including frequently asked questions on our products
- Technical papers from SmartBear
- Downloads, such as plug-ins and free tools, from SmartBear
- Hot Stuff contributed by experienced users and the AQA team (hands-on solutions, code, plug-ins, etc.)

Automated Build Studio Samples

A good way to get started using Automated Build Studio is to explore sample macros that are included with the Build Studio installation.

By default, samples are located in the <Automated Build Studio>\Samples folder.

For detailed information on samples and features they demonstrate, see Automated Build Studio help.

Getting Started

The Getting Started topics describe actions you may need to perform to create macros with Automated Build Studio.

Note: Automated Build Studio can be integrated into Microsoft Visual Studio. This integration allows developers to create, configure and run macros without leaving the Visual Studio IDE. For this purpose, Automated Build Studio's panels and toolbars are incorporated into Visual Studio.

This document provides information on how to work with Automated Build Studio running as a standalone application. For information on how to work with Automated Build Studio integrated into Visual Studio, see Automated Build Studio help provided with the Automated Build Studio integrated version.

Basic Concepts

Automated Build Studio is a management tool that allows creating visual ***macros*** to automate the build, test and release processes of software projects. A macro is an analogue of a project in development tools like Microsoft Visual Studio or Borland Delphi. You can save and load macros into Automated Build Studio, insert one macro into another, run macros, debug them, check the results and perform other actions.

Note: Besides editing macros, you can also use Automated Build Studio to modify MSBuild projects. When you open an MSBuild project in Automated Build Studio, the latter switches into a special edit mode that supports only MSBuild-related features.

This document contains information only on creating and editing macros. For information on editing MSBuild projects, see *Working With MSBuild Projects* section in the Automated Build Studio help.

Automated Build Studio macros consist of operations. An ***operation*** performs an elementary action, such as copying files or compiling a project. We use the word "operation" as a general term. For some operations, it would be better to use the term "statement", for others – the term "function" or "procedure".

All the operations that you can use in your macros are displayed in the **Operations** panel. They are organized into categories.

The **Macro** panel holds operations that you added to the macro. For each operation, it displays the operation name, description and two check boxes: the left box specifies if Build Studio will execute the operation; the right check box specifies whether the macro stops running if an error occurs during execution of this operation.

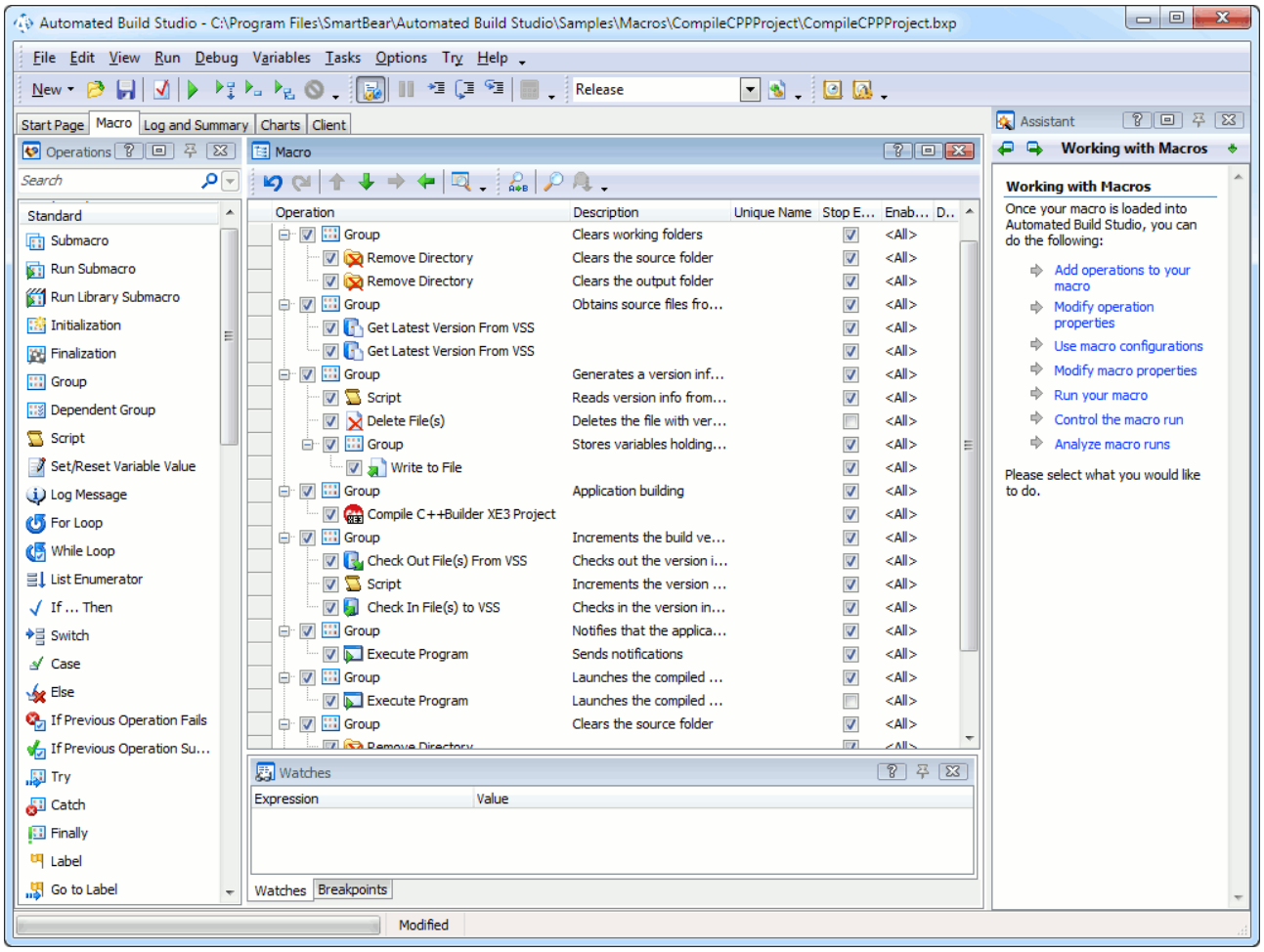
Some operations can have ***child operations***. For example, child operations of the For Loop operation will be executed as the loop "body". Child operations are displayed as child nodes of their parent operation. Child operations can also have child operations, which, in turn, can also have their child operations, etc. That is, the operation list in the macro has a tree-like structure.

By default, Automated Build Studio executes the operations in the same order as they appear in the Macro panel, from top to bottom. This is called serial execution. But you can also configure the macro so that Build Studio executes some operations or groups of operations ***concurrently***. This lets you perform

different tasks at the same time. For more information on how to run macro operations concurrently, see *Concurrent Execution of Operations*.

User Interface Overview

The user interface of Automated Build Studio consists of panels, the main menu and toolbars. The general layout is as follows:



Most of Build Studio's screen area is occupied by panels: **Operations**, **Macro**, **Log**, **Summary** and **Assistant**. Each panel serves a separate purpose in your work with Build Studio:

- The **Operations** panel contains operations you can add to your macro. The operations are organized into categories.
- The **Macro** panel holds operations that you added to the macro.
- The **Log** panel displays results of the macro execution. Automated Build Studio monitors every macro run and logs the name of every operation that was executed as well as the time spent on executing each operation along with all of its child operations.
- Like the Log panel, the **Summary** panel displays results of the macro execution but in the comparative form: you can simultaneously view results both of the current macro run and of the previous run that was finished without failures in the macro. Thus if the macro fails or an error

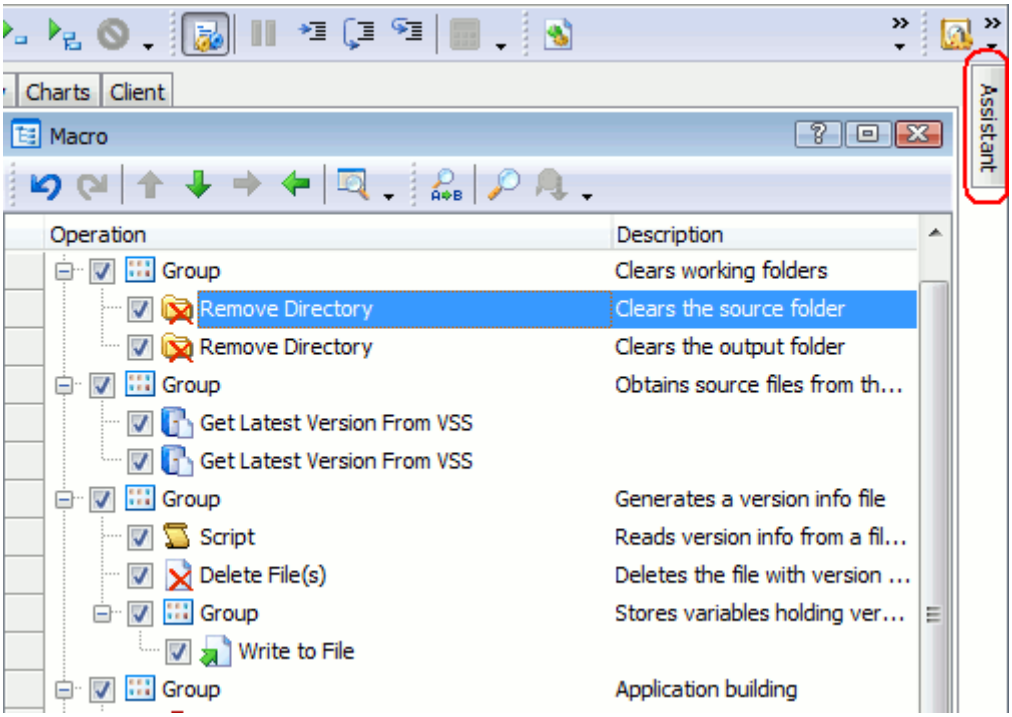
(warning) occurs in an operation, you can quickly see what caused this malfunction in comparison with the run where there were no such problems.

- The **Charts** panel displays statistics on macro runs performed on your computer. For more information on this, see *Charts* in the Automated Build Studio help.
- The **Client** panel is used to control the macro run on remote computers. For more information on this, see the *Automated Build Studio Client/Server* help topic.
- The **Assistant** panel helps you get started with Build Studio's user interface and provides a quick way to perform common tasks that arise when you are working with Build Studio.

Customizing the Panel Layout

The size and layout of panels are not fixed. You can change the size of panels, their position and docking, hide panels and make them visible.

Customizing panel layout in Automated Build Studio is similar to customizing panels in Microsoft Visual Studio. For instance, you can hide panels by clicking the button in the panel caption. This will minimize the panel along one of the sides of Build Studio's window, so only the panel caption remains visible:



The panel pops up when you move the mouse over the caption. For more information on auto-hiding and docking panels, see *Docking* in Automated Build Studio help.

In Automated Build Studio panels, you can also –

- Resize columns by dragging the separator between column headers.
- Swap columns by dragging their headers from one location to another.
- Add and remove columns to the panel.

The general view and behavior of Automated Build Studio panels can be modified in the **User Interface** dialog.

Toolbars and keyboard shortcuts in Automated Build Studio can also be customized at will. You can easily dock toolbars to any other edge of the Automated Build Studio window by dragging them to the desired edge of the window. To remove or add buttons, you can either call the **Toolbar Customization** window or use the *Quick Customization* feature. You can also create your own toolbars. For a complete overview, see *Toolbars Customization* in Automated Build Studio help.

Keyboard shortcuts can be customized via the **Customize Keyboard** dialog. You can define your own shortcuts or select one of the predefined key mapping schemes: *MS Visual Studio IDE* or *Borland IDE*.

Creating a Macro

Your macro is simply your current "work site" in Automated Build Studio. It unifies operations to be executed and variables and constants that will be used by the operations.

To create a new macro, use the **New Macro** command of the **File | New** menu.

After the macro is created, you can add operations to it and modify their properties, and create and modify macro constants and variables, and so on.

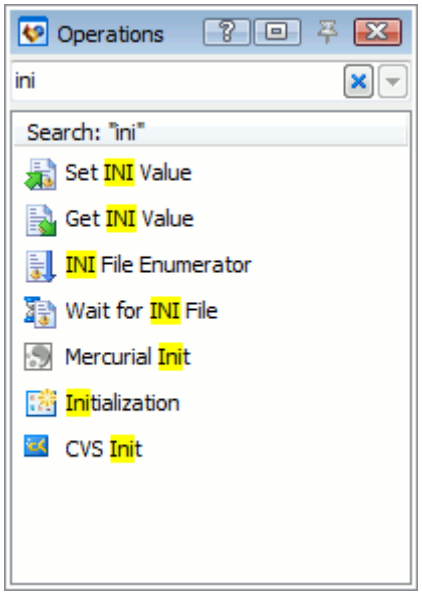
Adding, Removing and Arranging Operations in a Macro

Automated Build Studio offers a lot of operations that can be used in macros. Before adding an operation to a macro, you should locate that operation in the **Operations** panel. You can do this in either of the following ways:

- Expand the category holding the desired operation in the Operations panel. Some categories may be invisible within the panel, so to reach them, you should scroll the panel.

  -- or --

- Search for the operation by typing the operation name in the **Search** box at the top of the Operations panel. Automated Build Studio will search for the operation while you are typing and display operation that match the search pattern in the Operations panel.



To stop the search and restore normal categories display, select **Clear Search**.

Once you have located the desired operation in the Operation panel, you can add it to the macro in any of the following ways:

- Double-click the desired operation in the Operations panel. The new operation will be added to the macro below the currently selected operation.
- Click the desired operation in the Operations panel, then switch to the **Macro** panel. If you place the mouse cursor over an operation in the macro, the cursor will include an arrow that points upwards or downwards depending upon over which part (upper or lower) of that operation the mouse cursor is currently located. If you click that operation, the new operation will be added before or after it according to the arrow direction in the mouse cursor. If you click below the list of operations, the new operation will be added to the end of the macro.

To remove the chosen operation from the macro, either select **Remove Operation** from the context menu or from the **Edit** menu, or press Del. To delete all the operations at once, select **Clear All** from the context menu or from the Edit menu.

Child Operations

Some operations may have child operations. For instance, **While Loop**, **For Loop** or **If…Then**. For loop operations it means that child operations will be run within the loop body. For If…Then, it means that child operations will be executed provided that the *if* condition is True.

To make an operation a child one, simply drag the operation from the left (or right) panel and drop it over another operation you want to be the parent. To change operations relationships, you can also use the **Indent** and **Un-Indent** items of the **Edit** toolbar or the **Edit** menu of Automated Build Studio. In

addition, you can press Ctrl-Right or Ctrl-Left (these shortcuts can be customized in the **Customize Keyboard** dialog).

You can select operations within the macro and drag them along the tree: move the selected operation to another place, copy the operation, or make one operation a child of another.

Order of Operations

Note that by default, operations are executed in the same order they are displayed in the Macro panel of Build Studio -- from top to bottom (except for **Initialization**, **Finalization** and **Submacro** operations). It is also possible to run certain operations or groups of operations concurrently. To learn more, see *Concurrent Execution of Operations*.

To change the order of operations in the macro, use the **Move Up** and **Move Down** items on the Edit toolbar or the Edit menu, press Ctrl-Up or Ctrl-Down, or simply drag and drop an operation to the desired location. The operation being dragged is placed before or after the operation on which it is dropped: the exact new location of the dragged operation is indicated by an arrow in the mouse cursor which is displayed when you hover the cursor over an existing operation before dropping. You can copy the chosen operation to another location: just keep Ctrl pressed when dragging the operation.

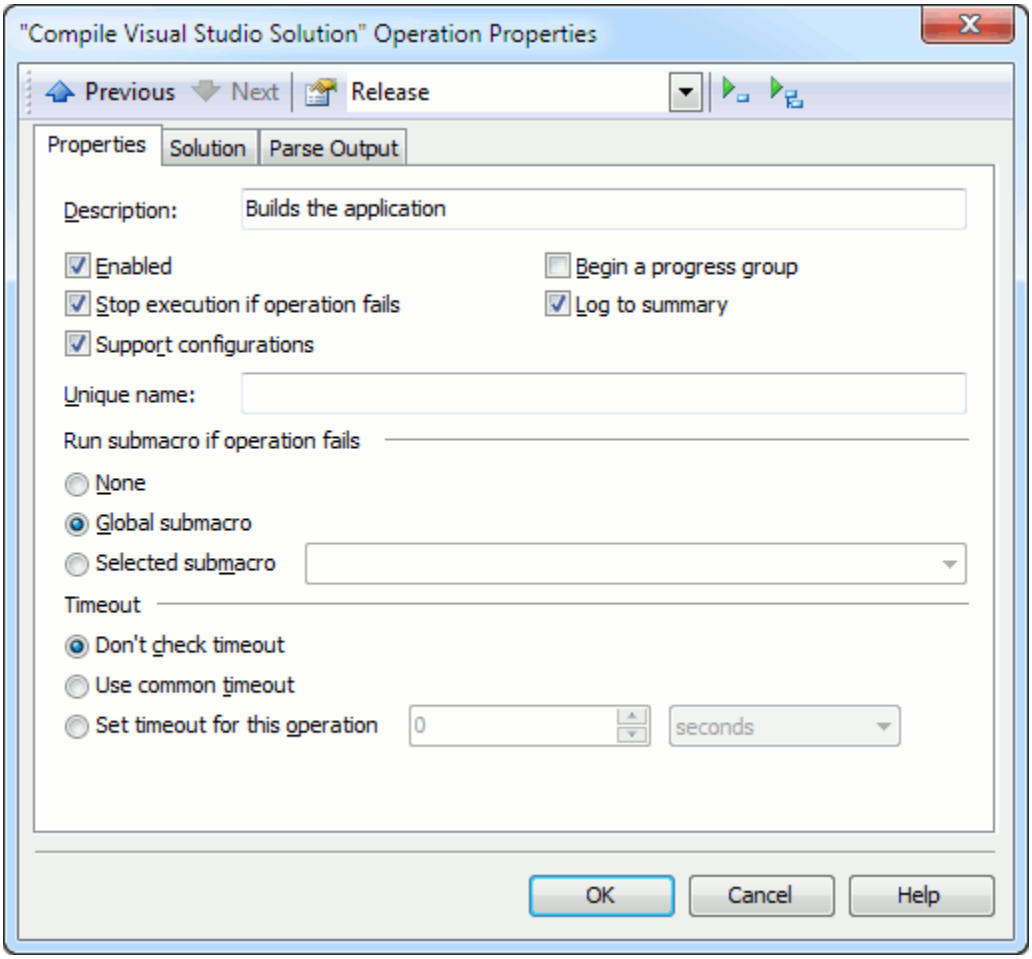
Modifying Operation Properties

Each operation has a number of properties that let you specify the actual actions that the operation performs. For instance, if you want your macro to compile a C# project, you should at least specify the project name in the operation properties. Therefore, once you have added an operation to your macro your next step is to modify the properties of this operation.

You can modify operation properties via Automated Build Studio's user interface or programmatically from scripts. In most cases you will change the operation properties via the user interface.

The operation's properties are set in the **Operation Properties** dialog. If the *Show Properties dialog when adding a new item* option of your macro is enabled, this dialog is called automatically upon adding an operation. Otherwise, you should call the dialog manually. To call the dialog, do one of the following:

- Right-click the desired operation in your macro and then select Properties from the context menu.
- Double-click the operation in the **Macro** panel.
- Select the desired operation and press Enter.



The dialog holds several tabbed pages. The first page, **Properties**, holds properties common for all operations; the other pages hold properties specific to the selected operation. For information on properties that are specific to a certain operation, see the description of that operation.

After you finish editing operation properties, you can select **Run | Check** from the main menu of Automated Build Studio to verify that the properties are set correctly.

Saving and Loading Macros

In Automated Build Studio, to save changes made to your macro, select **File | Save** from Build Studio's main menu or press Ctrl-S (this shortcut can be changed via the **Customize Keyboard** dialog). To save the macro under another name, select **File | Save As**.

Automated Build Studio can save macros in one of the following formats:

Format	Description
.bbp	The Automated Build Studio project file. Stores data in binary form.
.bxp	The Automated Build Studio project file. Stores data in XML format.

To open a macro that already exists, select **File | Open** from the main menu. This will call the standard Open File dialog where you can select the desired file.

Note: If you open a macro created with any of the earlier versions of Automated Build Studio, it will be converted to a new format and you will no longer be able to use this macro in other versions.

To open a macro that you have previously saved, select the **File | Recent** item in the main menu of Automated Build Studio.

To append an Automated Build Studio macro from a file to the currently open macro, use **File | Append From** item in the main menu of Automated Build Studio. Operations of the chosen macro will be added to the end of the current macro. Note that you may not append a macro to the currently open macro, but run it using the **Execute Macro** operation.

Running Macros

Once you have created a macro, you can run it:

- You can launch a macro at certain points in time by passing the macro name to Automated Build Studio via the command line.
- You can run your macro using the Automated Macro Player - a command-line utility that was specifically designed for running macros.
- You can run your macro right in Automated Build Studio.

You can run your macro in Automated Build Studio in one of the following manners:

- Press **Run** on the **Standard** toolbar.
- Select **Run** from the **Run** menu.
- Right-click within the **Macro** panel and select **Run** from the context menu.
- Press F5 (the shortcut for running macros can be changed via the **Customize Keyboard** dialog).

After that Automated Build Studio will start executing the macro. Note that it will execute only those operations that are checked in the Macro panel. Unchecked operations will not be executed. If you uncheck a parent operation, its child operations will not be executed either.

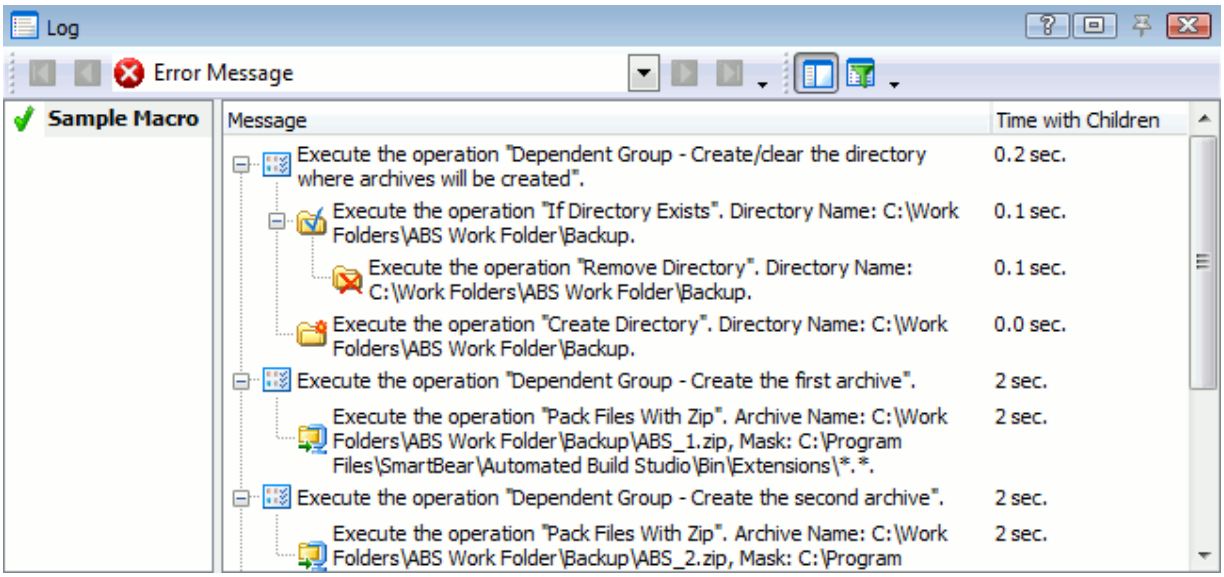
To pause the macro execution, select **Pause** from the Standard toolbar or from the Run menu. Pressing **Stop** will stop the macro execution.

Automated Build Studio logs each macro run and displays the results in the **Log** and **Summary** panels. For more information on this, see *Logging Macro Runs*.

Logging Macro Runs

Automated Build Studio generates full-detail logs of macro runs. These logs help you isolate the errors that might arise during the macro execution.

By default, when Automated Build Studio is executing a macro, it shows the name of each operation being executed in the **Log** panel. If an operation has child operations, they are logged as child nodes of their parent operation's node. If an operation fails to execute, Automated Build Studio will log information about the error that has occurred.

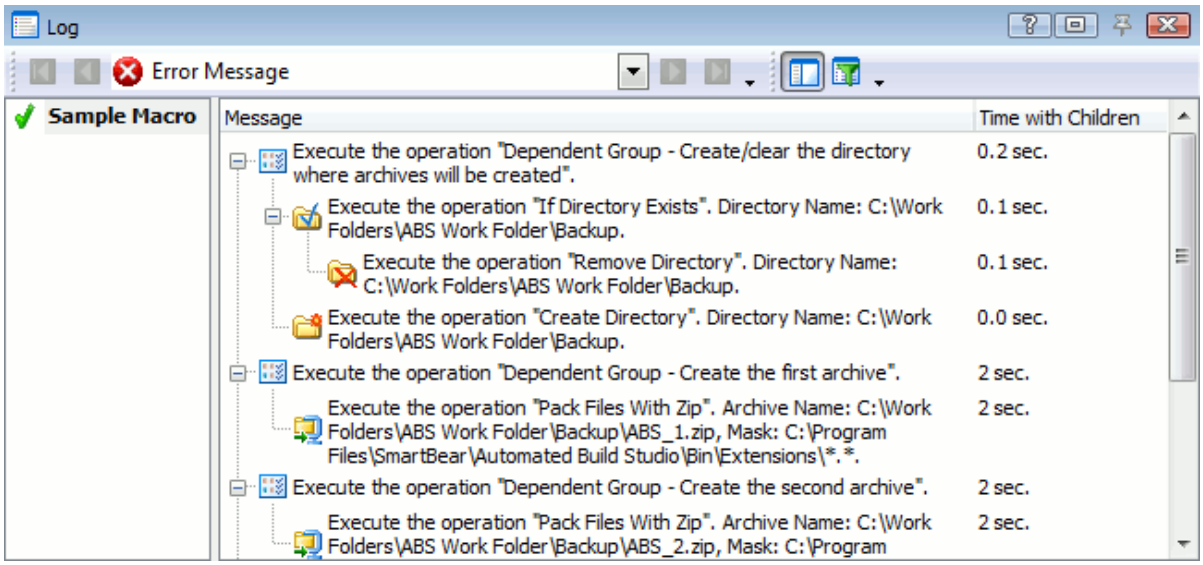


You can post your custom messages to the log. To do this, use the **Log Message** operation. The operation's properties let you specify what kind of message you would like to post: error, warning or an informative message (default).

Automated Build Studio can also post messages about execution of particular operations to the **Summary** panel: simply enable the *Log to summary* property for all the operations you want to trace. The panel is used to compare logs. For this purpose, it displays the logs of two macro runs (the current run and the previous successful run) and displays the differences between the two logs. In addition, this panel always logs all the errors and warnings that occur during the macro run. This lets you quickly find the cause of the problem.

Analyzing Results of Macro Runs

When you run a macro in Automated Build Studio, the results of the macro run are shown in the **Log** panel at runtime and after the run is over.

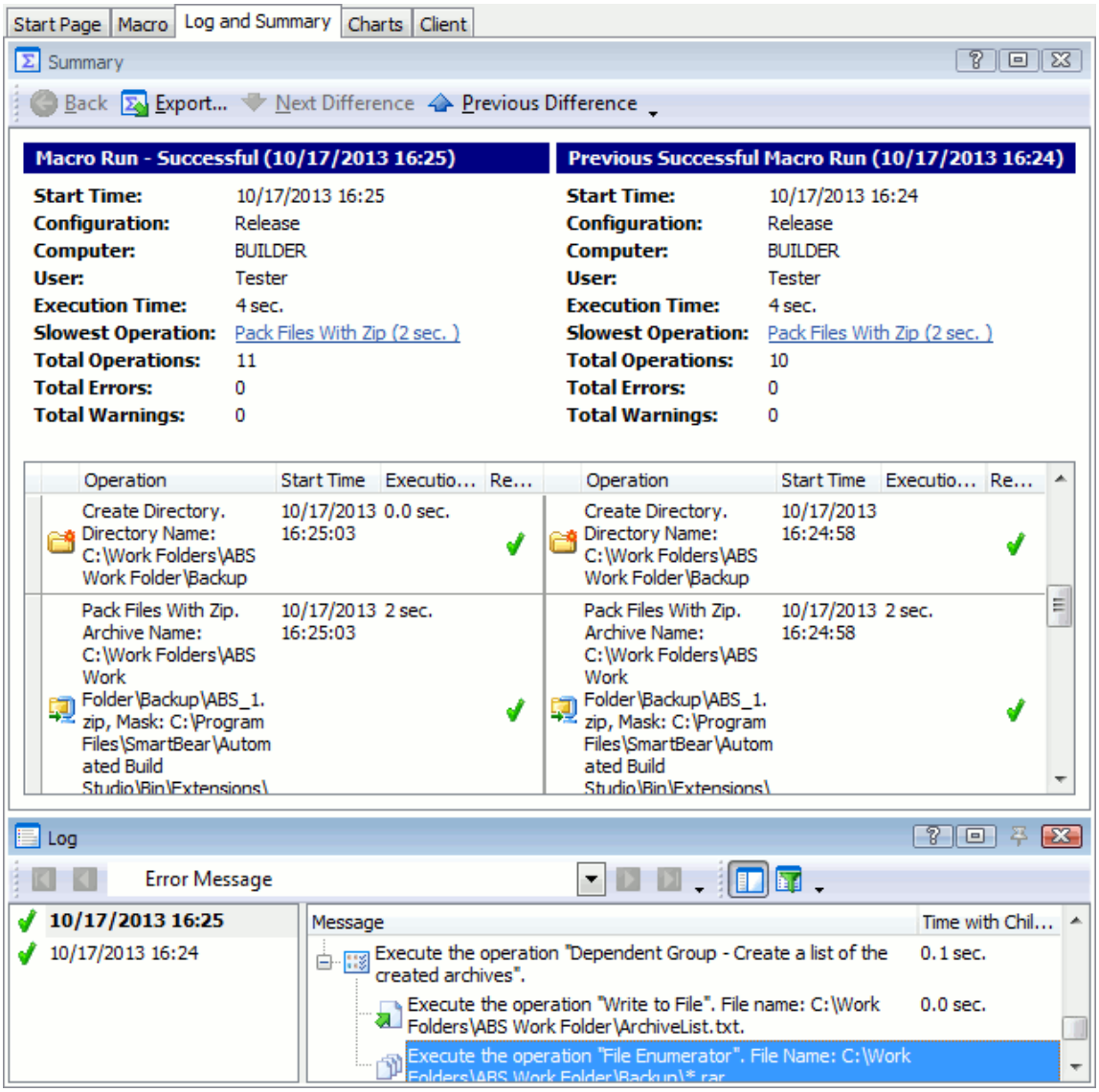


By default, the Log panel shows all messages that were posted to the log during the given run. To view only messages of certain categories, use the Message Filter dialog. To call it, select **Message Filter** from the context menu or from the **Log** toolbar the Log panel toolbar.

Using the **Log Navigation** toolbar the Log panel toolbar you can quickly locate messages of a particular type in the Log panel. This can be useful when the macro is complex and its log is extremely large. To do this, first select the desired type of messages in the **Select Message** dropdown list box on the toolbar (you can select among errors, warnings, informative messages or messages from operations of a particular type).

Then, using the **Go to First Message**, **Go to Previous Message**, **Go to Next Message** and **Go to Last Message** buttons on the toolbar, navigate through messages of the chosen type until you find the message you need.

Instead of investigating the contents of one log in the Log panel you can use a different approach when analyzing macro runs. To do this, switch to the **Summary** panel which displays logs on two different runs at the same time.



The first run is the current run where the macro might fail or an operation might be executed with an error or warning. The second run displayed is the previous run of the same macro where there were no failures. Automated Build Studio automatically compares the contents of the two logs and highlights the differences between them in the Summary panel. This lets you quickly find out why something went wrong in the macro.

Build Strategies

Continuous Integration

Continuous Integration Support

When more than one developer is working on a software project, from time to time the team members need to integrate their changes to the project's codebase and then run a build to see if the renewed code can be built and, if the build succeeds, and if it works without errors. These builds (they indicate the "health" status of the entire project) are called ***integrated builds***.

Normally, they are performed occasionally (for example, right before the deadline) or regularly (for instance, weekly or daily). It is hard to locate errors that arise in integrated builds that are performed rarely, since the number of changes introduced between the previous and current builds may be tremendous. To reduce the number of errors revealed in integrated builds and to make it is easier to find errors, you should perform these builds as often as possible.

Ideally, builds should be performed right after the source code of your project has been modified. This will help you continuously maintain the integrity of your codebase - it will be much easier to find errors that appeared after modifying the source code, because you will always know which source files were changed for this build. It is also recommended that each build be tested and reported appropriately.

By following this strategy, you will always be able to provide developers and users with a working build compiled using the latest versions of the source files. The strategy is called ***continuous integration*** (http://www.extremeprogramming.org/rules/integrateoften.html). It works well if:

- Your code is stored in a central location, preferably a source code control product like Visual SourceSafe. Build Studio macros can access different source control repositories.
- You have automated your build process, say, by writing an appropriate Build Studio macro.
- You have included tests (unit, functional, regression, etc.) in the codebase as part of the project (optional). These tests can be run as part of the build macro.
- Notifications about the build and test results are sent to respective users automatically. You can do this by including the appropriate operations in your macro.

As you can see, Automated Build Studio supports all of the prerequisites of continuous integration. In addition, it includes facilities that allow you to practice continuous integration of your projects in full.

The main facility is a dedicated Windows service called ***Automated Build Studio Service*** (*<Automated Build Studio>\Bin\AutomatedBuildService.exe*). You can configure this service via Build Studio's user interface to track all of the changes in files stored in a definite repository. To reduce the amount of time spent on interaction between the service and the repository and to lower the use of resources, the service does not poll the repository constantly, it does this at certain time intervals.

Continuous Integration Steps

The process of continuous integration provided by the Build Studio can be divided into several stages:

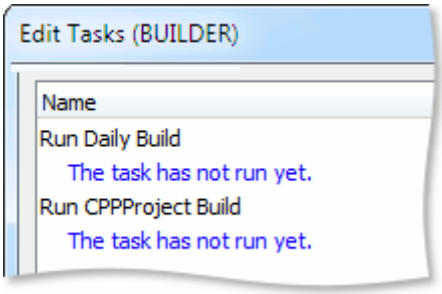
Preliminary steps

First, you need to perform the following preparations:

- Make sure that the Automated Build Studio Service is running.
- Prepare an appropriate macro that will build the project out of the latest source files, run a set of unit, functional, regression and other tests, and then report the build and test results to the respective users and to the log. This macro must be saved to a file. Unsaved macros cannot be used for continuous integration.

Creating a task

Click **Tasks (Continuous Integration)** on the **Tasks** toolbar of Automated Build Studio to call the **Edit Tasks** dialog. The dialog lets you create several tasks that the Automated Build Studio Service should perform. Each of these tasks is intended to run an existing macro in certain conditions. By default, the Edit Tasks dialog lists all the tasks that were created on your computer (the computer name is shown in the dialog caption).



Before creating a new task, select the computer where the task will run.

To be able to manage time- and event-triggered tasks on a remote computer, you must have appropriate permissions on that computer (the easiest way to get them is to add you to the Administrators group on the remote computer).

Since your task will run *Automated Macro Player*, Automated Build Studio must be installed on the remote computer.

Since your task will be performed by the Automated Build Studio Service, the service must be running on the remote computer.

Now let's create a new task:

- Click **Add** in the Edit Tasks dialog. This will call the **Edit Task** dialog.
- In the dialog, specify the task name, the macro that the task will run and other task parameters. For more information, see Automated Build Studio help.

  If your task will run on a remote computer, make sure that the path to the macro file is specific to that computer.

Creating the task's triggers

Now we are ready to specify triggers for our task. Each trigger defines conditions under which the task that will run the macro will be launched.

- Choose the **Triggers** node on the left of the dialog and switch to the details pane.
- Click **Add** to create a new trigger in the task.
- Specify the trigger's name and type. For a complete list of triggers and for information on their properties, refer to the *Tracing Task Triggers* and *Trigger Properties* topics in Automated Build Studio help. Specify the trigger's parameters.

Creating the task's blockers

If you want to block the task execution in some cases (for instance, at night, when your developers do not change the files under Visual SourceSafe), add appropriate blockers (blocking triggers) to the task. To work with the task's blockers, first select the **Blockers** node on the left of the Edit Task dialog. You can manage blockers (create, delete and modify their properties) in the same way you do this with triggers.


When a blocker is fired, all triggers defined for the task are ignored.

Final steps

Information about each task you create via the Edit Tasks dialog is saved to a .task file (the name of the file is the task name) in the *<Automated Build Studio>\Bin\Tasks* folder. This is actually an .xml file with a definite structure. Thus, you can modify the list of tasks and their properties by creating, deleting and modifying these .task files (provided that you keep the valid structure of each file).

To be able to run the prepared tasks, the Automated Build Studio Service must be running. In this instance, Build Studio's executable (*<Automated Build Studio>\Bin\AutomatedBuildStudio.exe*) is not needed until you wish to configure the task list again.

Upon running a task, you can view the status of this run in Windows Event Viewer.

Note: Windows Event Viewer will display those tasks that were executed. If a task is fired by a trigger, it will be only displayed if the trigger fired.

Additionally, the *Automated Build Studio Service* can be configured via the command-line. A full list of arguments that a service can accept is given in the Automated Build Studio help. Only those arguments that relate to task execution are mentioned below:

- `/trace:on` and `/trace:off` - Enables or disables the logging of resource usage for the *Microsoft Visual SourceSafe File Change*, *PVCS File Change*, *StarTeam File Change*, *Subversion File Change*, *Surround SCM File Change*, *Team Coherence File Change* and *Vault File Change* trigger types. Generally the logs contain the command line of the launched process (it is used to check the state of the traced resource), the process response, the request date and time and the latest date and time of the traced resource obtained during the check.
- `/poolsize:N` - Defines a number of tasks that can run simultaneously. The `N` value must range within the 1 - 30 interval, the default value is 5.
- `/exit` - Closes the AutomatedBuildService.exe when it is launched as a stand-alone process and stores the values of the `/poolsize` and `/trace` arguments to the registry.

Note on Team Build Integration

You can use the scheme described above for extending continuous integration features that are provided by Team Foundation Build 2008 and higher.

If Automated Build Studio is integrated with Team Foundation Build, you can use the macros to extend the build targets and specify the custom sequence of actions to be performed during the build run (for more information on Team Build integration and extending build targets with Automated Build studio macros, see *Team Build Integration* in the Automated Build Studio help). Those macros are stored in the source control repository and are registered with the `UsingTask` element in the .proj file.

You can automate the build run as described above to run it every time the macros used for target extending are changed: create a macro that runs the build triggered by macros files changing in the source control repository.

You can also use this scheme to provide the continuous integration feature for Team Foundation 2005 that does not have its own continuous integration support.

Distributed Builds

About Distributed Builds

A process of building an application includes several tasks, such as getting the latest version of source files from a version control system, compiling the source files, creating the installation package and others. Some of these tasks (mostly, compilation of source files) can be divided into smaller tasks, each of which can be run on a separate computer at the same time as other tasks.

These build processes, whose workload is distributed among several cooperative computers within the network, are called ***distributed builds***.

Distributed builds essentially reduce the project build time due to the fact that the build tasks are executed simultaneously on several computers. Using distributed builds, you can expect the build performance to speed up depending on the number of computers you use in the build. In this way, you can run your builds faster and more often, which is an essential part of the ***continuous integration*** strategy.

In Build Studio, distributed builds can be organized by using special operations in the **Remote Macro Execution** category:

- **Remote Group** – Executes a group of operations on a specific remote computer.
- **Remote Execution Stack** – Executes its child operations in parallel, on the specified remote computers that are available at that time.
- **Remote Macro Command** – Lets you execute a specific macro on a remote computer.

Within the **Remote Group** and **Remote Execution Stack** operations, you can use the same operations that are used in typical macros. The only difference is that the properties of these operations (such as file names and paths) must be specified with regard to the remote computer on which they will be run.

The remote operations' logs are also transferred to the main computer once the execution returns to the main macro. They are included in the macro log, which is displayed in the **Log** panel, and taken into account in the macro summary.

Creating Distributed Builds

Macros that perform distributed builds are similar to other build macros, with the difference that some of their tasks are split into smaller tasks that are distributed for running on multiple remote computers. The key

point in creating distributed build macros is to specify a set of computers that can participate in distributed builds and to divide a single build task into several independent tasks:

Configuring Remote Computers

The computers that participate in the distributed build must be configured in a special way.

First, the remote computers must have all of the needed software (such as compilers, build tools and helper utilities) installed and properly configured.

In addition, Automated Build Studio must be installed on these computers and the Automated Build Studio Service must be running on them (this service is used to transmit data between the host computer where the build macro is running and the remote computers where the individual macro tasks are executed).

Connecting to Remote Computers From Macros

Automated Build Studio provides a special **Setup Connections** operation that can be used to specify a set of remote computers for the distributed build and to adjust the connections to these computers. For information on the operation and how to use it to configure connections, see Automated Build Studio help.

Distributing Operations Among Remote Computers

An essential part of creating distributed build macros is to decide which build tasks should be split into smaller ones and to distribute them among remote computers.

Automated Build Studio does not automatically distribute the compile process among remote computers. If you need to distribute the compile process, you should manually create and configure individual projects for building each project module.

You can distribute the remote build tasks among specific computers, or among those specified computers that are available at that time. Again, these computers must have all of the needed software for performing the corresponding operations. For example, if you want to distribute the compile process among several computers, the compiler must be installed on all of them.

When specifying properties of operations that are executed on a remote computer, remember that file names and paths must be specified in the terms of that computer.

Running Operations on a Specific Remote Computer

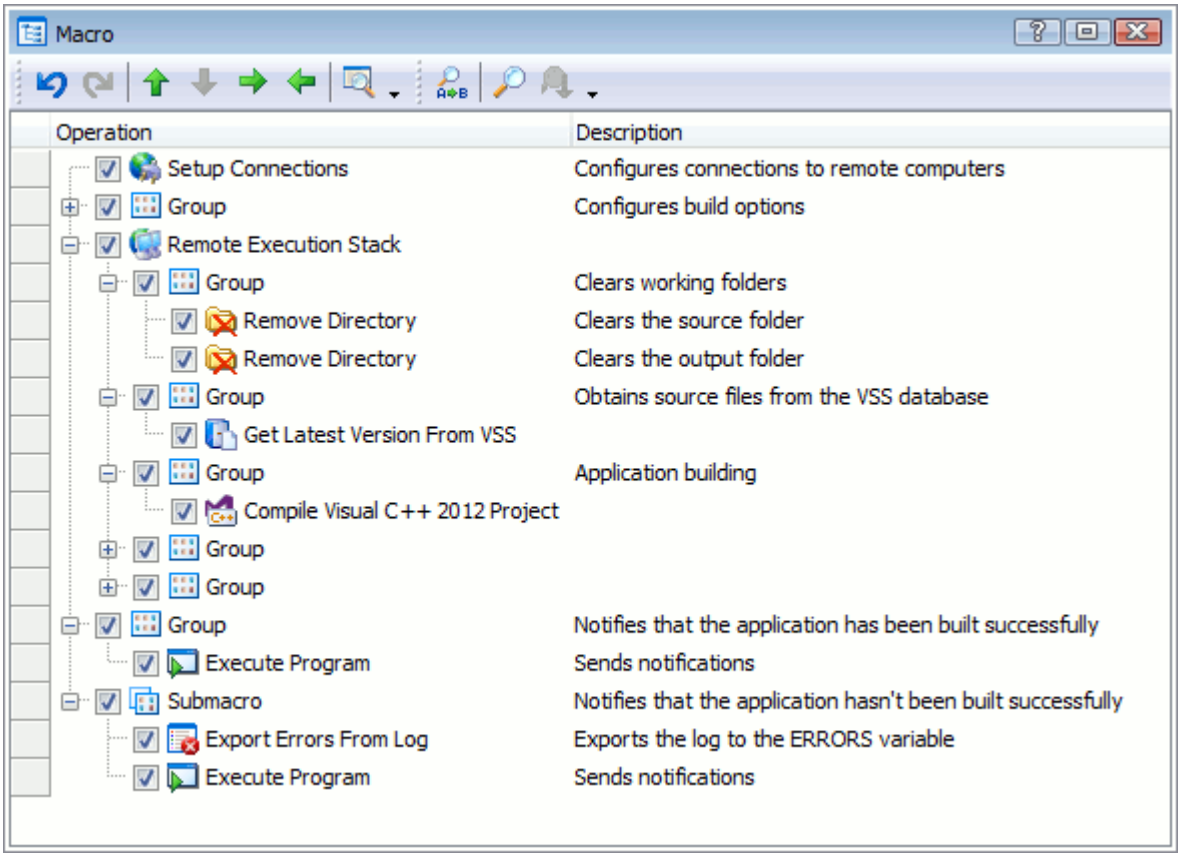
If you wish to execute a group of operations on a specific remote computer, use the **Remote Group operation**. For information on how to configure the operation, see Automated Build Studio help.

Running Operations on Several Remote Computers Concurrently

If you want to execute several operations or groups of operations on multiple remote computers in parallel, you can place several **Remote Group** operations under a concurrent **Group** operation (see *Concurrent Execution of Operations*). In this case, the **Remote Group** operations are run simultaneously on the corresponding remote computers.

However, you may also want to execute an operation or group of operations on any of the computers available for this purpose. This can be achieved by using the **Remote Execution Stack** operation. For information on how to configure the operation, see Automated Build Studio help.

Note that when executing the **Remote Execution Stack**, Build Studio distributes its direct child operations among the available computers. If you need to execute a group of operations rather than a single operation on a remote computer, you should first add the **Group** operation as a child of the Remote Execution Stack operation and then add the desired operations as children of the Group operation, like on the image below:



You can also group operations to be run remotely by using the **Dependent Group** operations. In this case, however, the Remote Execution Stack behavior will be a little different - it will not run all of the groups concurrently, but distribute and execute them in the order that is defined by the group's dependencies. See *Operation Dependencies* for details.

Running Macros on a Remote Computer

Besides executing a group of operations on a remote computer, you can also run a whole macro remotely. For this purpose, use the **Remote Macro Command** operation. For information on how to configure the operation, see Automated Build Studio help.

If a macro that you wish to run on a remote computer has yet to be uploaded to that computer, you should first upload it. In macros, you can do this using the **Upload Macro** operation.

Note: The logs of remote macro runs are stored on the remote computer.

Running Distributed Build Macros

Once you have configured the build servers that will participate in the distributed build and the computer that will host the build, you can run the build macro. Automated Build Studio macros can be run in several ways:

- You can run Automated Build Studio on the host computer and start the macro directly from Build Studio.
- You can run the macro on the host computer from the command line, using either Automated Build Studio or Automated Macro Player.
- You can schedule the macro to run at specific times on the host computer.
- You can start the macro on the host computer from another computer using the Automated Build Studio Client/Server functionality.
- You can run the macro on the host computer via the Automated Build Studio Web Interface.

Operation Dependencies

About the Dependent Group Operation

To let you set dependencies between macro operations, Automated Build Studio provides a special **Dependent Group** operation. The purpose of this operation is to separate a group of operations that need to be executed prior to or after another operation group.

The Dependent Group's properties include a list of the group's prerequisites, that is, groups that need to be finished in order for the given group to start. In a way, these dependencies are analogue to target dependencies in *makefiles* used by the `make` tools. However, unlike *makefiles*, Build Studio macros do not have targets, so all of the dependent groups in the macro (unless some of them are disabled) will be executed when you run the macro. Dependencies only define the groups' execution order.

When you run the macro containing dependent groups, Build Studio analyzes the dependencies between them, builds the macro flow graph and executes dependent groups according to this graph. Where possible, groups are executed in parallel, in order to improve the macro performance and speed it up.

Typically, the Dependent Group operations should be the top-level operations in your macro (so that the macro "consists" of dependent groups). Optionally, they can be grouped by using the **Group** operations; this grouping does not affect the execution order of dependent groups.

Any other macro operations, except for Dependent Groups, their parents and children and operations mentioned in the **Remarks** section, are ignored.

The order in which you arrange the Dependent Group operations in your macro does not matter and does not affect their execution order, which is completely determined by the group's interdependencies.

How Dependent Groups Work

When Automated Build Studio (or Automated Macro Player) runs a macro that consists of dependent groups, it first forms the macro dependency graph and ensures that there are no circular references between dependent groups. If Automated Build Studio finds a circular reference, it displays an error message that notifies you about the found loop. In this case, you will need to correct the dependencies to be able to run the macro.

If the macro structure is consistent, Build Studio picks out those **Dependent Group** operations that do not have dependencies specified, and runs them concurrently. Once the execution of a particular Dependent Group finishes, Automated Build Studio starts executing groups that depend on the finished group. If a

specific Dependent Group operation depends on several groups, it is only executed after all of the latter groups have been completed.

If dependent groups are children of the **Remote Execution Stack** operation, the situation is similar. In this case, Automated Build Studio automatically distributes dependent groups among remote build servers according to the operation dependencies.

If the number of groups to be run at a certain moment exceeds the *Maximum thread count* property of the macro, then only the first *Maximum thread count* groups are started; the rest are queued. For example, if the *Maximum thread count* is set to 1, dependent groups will be executed serially, one after another (of course, according to their dependencies).

Remarks

There are a few limitations applied to macros that consist of dependent groups:

- Circular references between dependent groups are not allowed. Automated Build Studio cannot run a macro if it contains two or more Dependent Group operations that depend on each other.

  To check the macro consistency, you can use the **Run | Check** command of Automated Build Studio's main menu. If Automated Build Studio finds any circular references between dependent groups, it will display an error message notifying you about the found loop "path".

- If the macro contains dependent groups, any macro operations other than Dependent Groups (including their parents and children) are ignored. However, there are a few exceptions to this rule:
  - The **Initialization** and **Finalization** operations are executed before and after all of the dependent groups, respectively.
  - The **Submacro** operation defines a submacro that can be run from within a dependent group through the Run Submacro operation.
  - The **Setup Connection** operation is used to establish connections to remote computers to be used during the build.
- The  Run From Selected command cannot be used for macros that consist of dependent groups. The  Run to Cursor command, however, is still available.

Concurrent Execution of Operations

With Automated Build Studio, you can organize **concurrent** execution of macro operations and thus perform different tasks at the same time.

Concurrent (or multithreading) execution lets you improve the performance of Automated Build Studio macros and reduce the macro execution time. When macro operations are run in a series, they are executed within the same thread, so the next operation only starts after the previous operation is finished. When groups of operations are run concurrently, Automated Build Studio starts the next operation in a separate thread, without waiting for the previous operation to complete. As a result, the CPU is used more effectively and less time is needed to execute macros.

Concurrent execution of operations can greatly improve performance in the following cases:

- A macro is executed on a multiprocessor computer. A processor can only execute one thread at a time. So, on a computer with a single processor, only one thread can be executed at a time; whereas a computer with ten processors can execute ten threads at a time.
- A macro contains operations that require a lot of time (for example, the Get Latest Version operations for large projects, file uploads and downloads, database queries and so on). When this

type of an operation is being executed, other operations cannot be started until that operation is finished. Putting an operation in a concurrently running group will allow other operations to run simultaneously with this operation, and it will not block them.

- A macro contains operations that perform data transmission over a network (for example, operations of the FTP, HTTP and other groups). When running concurrently, each operation uses a separate connection so that the overall bandwidth is greater than the serial operation's execution, when all operations use the same connection.

Generally, almost all operations provided by Automated Build Studio can be run concurrently (however, there are technical and logical restrictions described below). Here are some examples of the operations that should be run concurrently:

- To split long-running tasks into smaller tasks. For example, an operation that obtains source files from a Source Control System (SCS) can be divided into several same operations, each obtaining a certain part of the source files. Running concurrently, these operations will take less time than a single operation that performs the whole task.
- Run the operations, that do not depend on each other and do not use the same resources, concurrently. For example, if your macro contains the Remove Directory operation that cleans temporary folders and the Upload File(s) to FTP operation that uploads one or several files to an FTP server, you can group them so that they run concurrently.

Usually, the best performance is achieved when a multi-threaded macro is run on a multiprocessor computer. In this case, Automated Build Studio automatically schedules macro threads across available processors, which Automated Build Studio is allowed to run on, in the most effective way.

Organizing Concurrent Execution of Operations

There are two ways to organize concurrent execution of macro operations:

- You can configure your macro so that Automated Build Studio automatically parallelizes the operations' execution.
- You can directly specify which operations should be run in parallel.

The former approach requires re-structuring your macro file and specifying the dependencies between different operation groups. When running this macro, Automated Build Studio (or Automated Macro Player) automatically determines the macro execution flow and, where possible, parallelizes the execution of certain operation groups.

While the former approach affects execution of the whole macro, the latter one lets you adjust the execution of specific operations. These operations should be grouped into a **Group** operation with the *Run child operations asynchronously* property turned on. Unlike the rest of the macro, where operations run step-by-step, these grouped operations will be executed in parallel.

Possible Issues with Concurrent Execution of Operations

There are a number of things that you should keep in mind when organizing concurrent execution of operations in your macro.

Reasoning from the macro execution logic, you should not run concurrently those operations that depend on each other. For example, an operation that prepares resources for a certain task and an operation that performs this task should not be run concurrently, because when the latter operation starts, it may turn out that the resources are not prepared for it.

Issues may occur if you have concurrent operations that use the same resources (for example, if the operations save their results to the same file or Build Studio variable). In this case, a resource may be blocked, or it may be accidentally changed by another operation. This can cause errors during the operation's

execution. To avoid these errors, do not concurrently run those operations that use and modify the same resources. Another solution is to use "thread-relative" variables. These variables allow you to have different values for different threads.

The issue described above, results in another restriction: if an operation's API requires exclusive access to a resource, then the operations implemented via the same API cannot run concurrently. Currently this restriction applies to Team Coherence operations from the **Issue Tracking** and **Team Coherence** categories. These operations should not be executed concurrently, otherwise an error may occur.

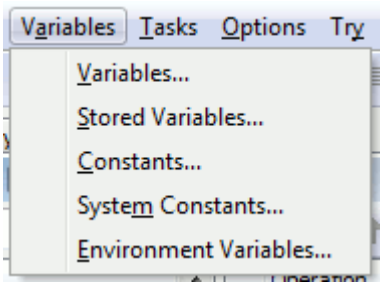
In addition, a great number of concurrently running operations require a lot of operating system resources in order to keep track of all threads. When switching between the threads, the system saves the context of the current thread and restores the context of the next thread. These actions can take a significant part of the processor's time, which will eliminate the benefits of concurrent operation execution. Automated Build Studio lets you limit the number of threads that can be started for a group of operations. An optimal number of concurrent operations depends on the configuration of your computer.

Using Variables and Constants in Macros

About Variables and Constants

Macros in Automated Build Studio can contain ***variables*** and ***constants*** that you can use to set values for operation properties.

To create or check values of variables and constants, use commands of the **Variables** menu.



Note that both variables and constants can be used in scripts.

About Variables

Variables let you make your macros more flexible and powerful. They can be used to specify values of operation properties (such as file names, build numbers, and so on) as well as to store the operation result and output. They are also accessible in scripts and serve as ordinary script variables.

There are four types of variables that can be used in macros:

- ***Global variables*** - Are accessible from any operation in your macro. It is possible to provide default values for global variables – the variables will be assigned with these values at the beginning of the macro run. Additionally, global macro variables can have two attributes:
  - ***Environment*** – This attribute determines how a variable is used. A variable that has its environment attribute enabled is passed to the process launched by the macro and serves as an environment variable for that process.
  - ***Thread-relative*** – This attribute is useful if the variable is supposed to be used in several concurrently executed operations. If the variable has this attribute set, each macro execution thread gets its own copy of the variable, so that the variable can be referred in different threads without causing a resource conflict.
- ***Local variables*** - As follows from their name, are available only in a particular local scope. They can be defined in the **Group**, **Dependent Group** or **Submacro** operation and will be "visible" only to operations that belong to that group (submacro). Outside the group (submacro) where a local variable is defined, it does not exist and any references to it will cause an error.

  Similar to global variables, local ones can have default values, which you can specify in the properties of the Group or Submacro operation. Variables are initialized with these values when Automated Build Studio executes operations in which local variables are defined.

- ***Stored variables*** - Are very similar to global variables. The only difference is that the values of stored variables are saved between macro runs. Stored variables are useful, for instance, to keep the build number, which could easily be incremented in any of the next runs.

  The values of stored variables can be saved either in the macro file, or in a separate .bvars file (this file will be created in the same folder that contains the macro file). The latter can be useful, for instance, if you need to have computer-dependent values of stored variables.
- ***Environment variables*** - Environment variables contain information about the system environment and the currently logged on user. These variables are defined by Windows, programs and users and cannot be created in Automated Build Studio. Build Studio loads the list of environment variables when it starts.

  You can change values of environment variables in your macros, however, these changes will only be "visible" to Automated Build Studio and applications started by your macro. Also, Build Studio will restore original values of environment variables at the end of the macro run.

About Constants

Constants are very similar to variables, however, their values cannot be changed during the macro run. There are two types of constants:

- ***Ordinary constants*** - Specify arbitrary values that are supposed to remain unchanged during the whole macro run. Examples of such constant values are: paths to the folder containing the application's source files and to the output folder, your company information to be written to the application modules, and so on.

  Ordinary constants can have the *Environment* attribute, which defiles how a constant is used. All constants that have this attribute set will be passed to the applications launched by the macro.
- ***System constants*** - Specify paths to applications installed on the computer and provide information about the current macro file. Automated Build Studio loads values of system constants at startup. These values can neither be changed in Build Studio, nor at runtime.

  The main purpose of system constants is to make macros easier and portable. For example, if your macro somehow utilizes the path to the folder where the macro file resides, you can use the ABSOPENEDPROJECTDIR constant to obtain this path, so that you will not have to change anything in your macro in order for it to be successfully run from another folder.

Remarks

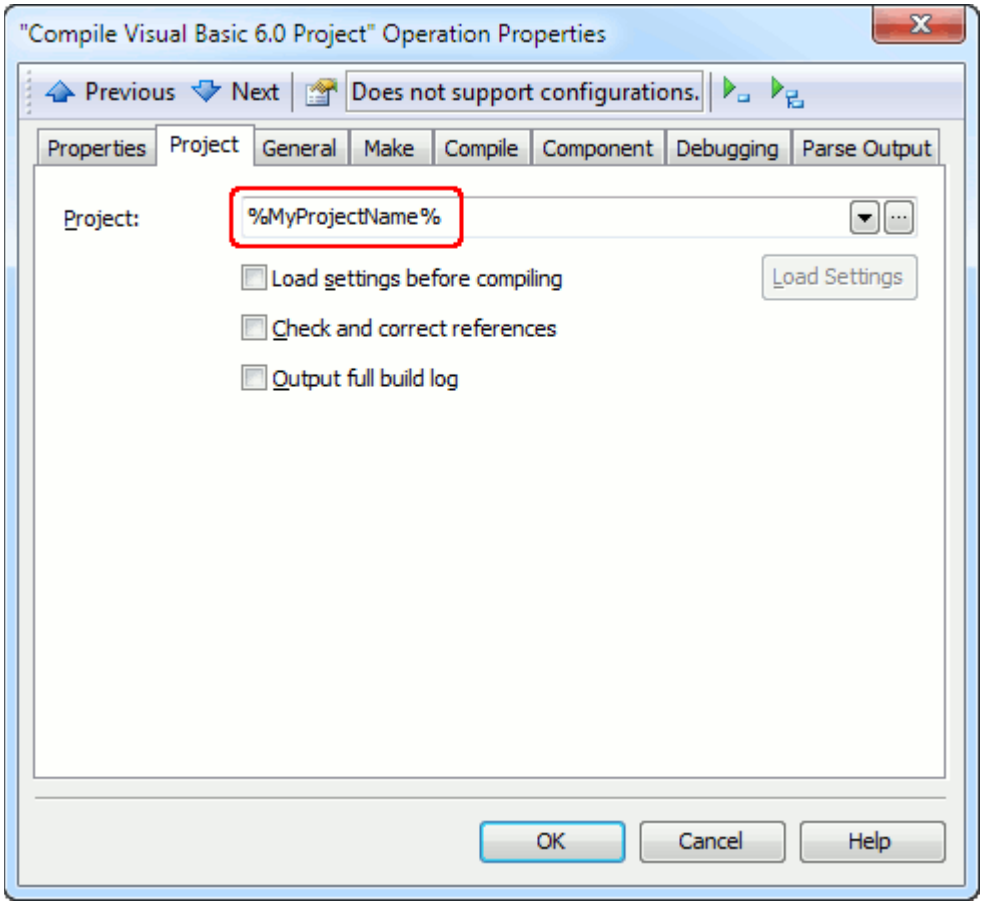
Three things you should keep in mind about variables:

- Since variable names are used to address variables in scripts, the variable name must match the naming rules used in the scripting language you chose for the script. To make the variable name suitable for any supported scripting language, follow this simple rule when naming a variable: a name can include only letters, digits and underscores and must begin with a letter.
- Variables and constants use the same namespace, so you cannot create a global, local or stored variable that has the same name as an environment variable, system constant or an existing ordinary constant.
- Neither global, local, stored nor environment variables can be used as loop variables in scripts.

Using Variables and Constants in Operation Properties

You can use variables and constants in almost any operation property. The way you specify a variable or constant name depends on the purpose of the variable (constant).

If you use a variable or constant to specify a property value, you should enclose the variable (constant) name with the % signs. Otherwise, Automated Build Studio will treat the variable (constant) name as a string value. For instance, on the following figure we use the MyProjectName variable to specify the Visual Basic project to be compiled:



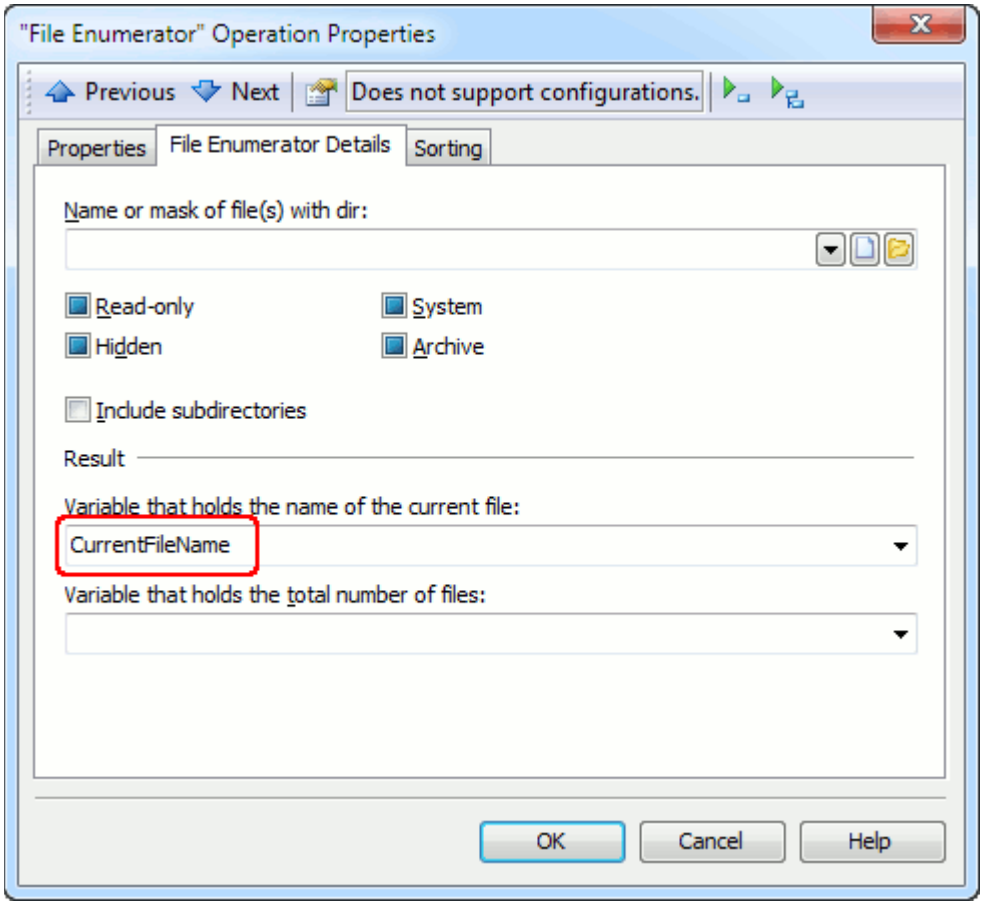
Note: To specify the % character in the property value, use the *%%%* string.

To concatenate values of two or more variables or constants, simply put them one after another. For instance, if one variable holds the file name and another one holds the file path, the following string will return the entire file name:

```
%File_Path_Var%%File_Name_Var%
```

You can concatenate two or more constants in the same way you concatenate variables. You can also concatenate constants with variables and strings.

If you use a variable to save an operation result (for example, to save a file name returned by the File Enumerator operation), you should not use the % signs:



Using Variables and Constants in Script Operations

You can set the value of a variable in your macro either by using the **Set\Reset Variable Value** operation in your macro, or by writing a script.

From the script's point of view, macro variables are public variables that are visible within the entire macro. To address a variable, use the following syntax:

```
Variables.variable_name = value
```

Using the `Variables.variable_name` format when referring to the macro's variables allows you to avoid confusing variable names if your script holds local variables that are declared using the same names.

To address a constant, use the `Constants` object:

```
LocalVar = Constants.constant_name
```

Macro variables cannot be used as loop variables in scripts. A loop variable should be a local script variable.

Also, macro variables cannot be used to store object references. They can only store ordinary values: numbers, strings, dates and so on.

To change the value of a macro variable, you can either assign the desired value to the variable directly, or use the `ABSUtils.SetVariableValue` method:

```
ABSUtils.SetVariableValue("variable_name", value, for_all_configs)
```

The difference from this method and the usual assignment operation is that, depending on the last parameter (*for_all_configs*) it only changes the variable value in the current configuration or in all of the macro configurations at once. The latter can be useful when modifying values of stored variables.

A similar method, `ABSUtils.ResetVariableValue`, lets you reset a macro variable to its default value. As you may notice, these two methods are scripting analogues of the Set/Reset Variable Value operation.

Specifying Operation Properties

Specifying Operation Properties - Overview

One of the ways to specify operation properties is to use the **Operation Properties** dialog. In the dialog, you can type operation properties' values or specify them in the following ways:

- Using special dialogs - You call these dialogs by clicking the ellipsis button to the right of a property's edit box. In general, these dialogs are used to specify files and directories. These can also be dialogs that allow you to create a list of files that will be specified as a property value or select the desired value from the available list.
- Using variables and constants - You can specify a constant or a variable that holds the desired value. You can specify the desired variable or constant manually or select it using the Variables and Constants dialog. You can also type the % symbol in the property field and select the desired variable from the appeared window.

  When using variables and constants, keep in mind the following:

  - If a variable or constant is used to specify a property value, the variable (constant) name must be enclosed in % symbols, for instance, *%Value_Var%*.
  - If the operation saves a property value to a variable, do not enclose the variable name in % symbols.

  For more information, see *Using Variables and Constants in Operation Properties*.
- Using scripts - To specify string properties, you can also use single-line script statements. For instance, if you want to specify an operation property as the return value of the DateToStr built-in script function, type the `Script: DateToStr(Date)` string in the property field.

  Note: A script line must always start with the `Script` prefix.

For more information, see *Using Scripts in Operation Properties*.

Using Scripts in Operation Properties

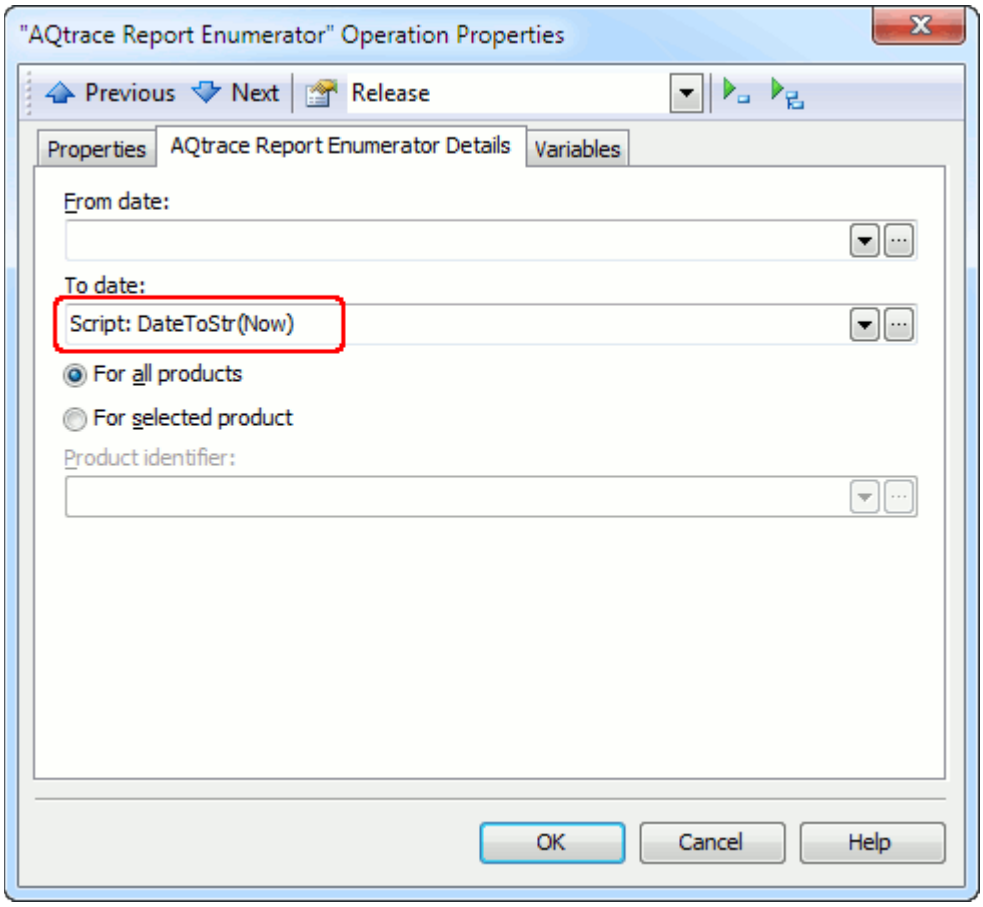
You can use single-line script statements to specify operation properties that can have a string value. If you specify a script statement in the property edit box, Automated Build Studio will execute it during macro execution and will assign the return value to the operation property.

The rules of script writing are the same that are used in the Script operation. A script line can be written in one of the three supported scripting languages: DelphiScript, VBScript or JScript. All three supported scripting languages use only OLE-compatible data types and cannot use pointers. For more information, see *Script Writing*.

When you use a script in operation properties, the following requirements must be met:

- You can specify only a single-line script statement.
- The statement must return a string, otherwise, errors will occur during the operation run.

- A script line must start with the `Script` prefix. Otherwise, Automated Build Studio will treat the script line as a string value. For instance, in the figure below, you can see that we use the *Script: DateToStr(Now)* string to specify the end date of the analyzed period:



In operation properties, you can specify script lines that:

- Evaluate simple mathematical expressions. For instance, *Script: 100 + 200*. In this case, the operation property will be set to *300*.
- Call built-in script functions. For instance, *Script: DateToStr(Date)*. An operation property value will be specified as a string containing the specified date.
- Call any methods and properties of internal and external objects. For instance, *Script: Variables.MyVariable*. An operation property value will be specified as the value of the *MyVariable* variable.

You can also concatenate values of two or more script statements using the language-specific syntax. For instance, the following string specifies the current date and time:

```
[VBScript]
Script: DateToStr(Now) & " " & TimeToStr(Now)
```

```
[JScript]
Script: DateToStr(Now) + " " + TimeToStr(Now)
```

```
[DelphiScript]
Script: DateToStr(Now) + ' ' + TimeToStr(Now)
```


Macro Configurations

About Macro Configurations

Why Use Configurations?

Often, when you make a build or an installation by using Automated Build Studio, you need to create different versions of the same build (installation): for instance, a debug (checked) version, a demo version or a release version.

Typically, macros that generate these versions slightly differ from each other: there are just some changes in the logic, operation properties and in the values of constants and stored variables. You can create an individual macro to generate each version independently, but this is inefficient, because these macros will have lots of identical parts. Since all of these versions pertain to the same product, the most reasonable solution is to have a single macro that would generate all of the versions. This is achieved via ***macro configurations***.

Macro configurations allow you to re-use the same logic with different data sets. This is like passing different parameters to a function to get different results. For example, you can create the *Release* configuration that builds your product with optimizations turned on, and the *Debug* configuration that builds it with debug information included and with no optimizations.

To sum up, using macro configurations has the following benefits:

- You can re-use the same logic and the same macro for building several versions of your product, or even several closely related products.
- Whenever you need to change the common logic of your build processes or fix a problem in them, you modify a single macro. (If you had several identical macros, you would have to update each of them).

What Data Configurations Include

A macro configuration includes the following:

- Property values of all operations included in the macro,
- Values and *Environment* attributes of all variables, stored variables and constants defined in the macro.

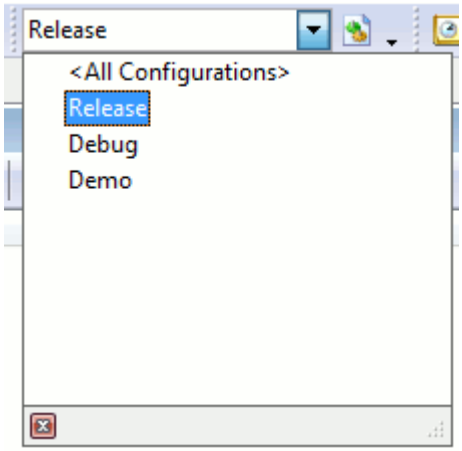
Note that the **set of macro operations, variables and constants is shared among all configurations**. So, if you add or remove a macro operation, variable or constant in one configuration, it will be added or deleted in other configurations as well.

All information about configurations (both the configuration names and configuration-specific values) is stored in the macro file. Stored variables, however, can store their values in a separate file -- this can be useful if these values need to be computer-dependent (for example, working folder paths on the development and production machines).

Managing Macro Configurations

Each Automated Build Studio macro has at least one configuration. The default configuration name is *Release*. You can create any number of additional configurations if necessary, each with a unique name. You can also remove the configurations that are no longer needed in your build.

All configurations defined in the current macro are displayed in the **Select Active Configuration** list on the **Configurations** toolbar. You can quickly change the active configuration by selecting the desired configuration from the list.



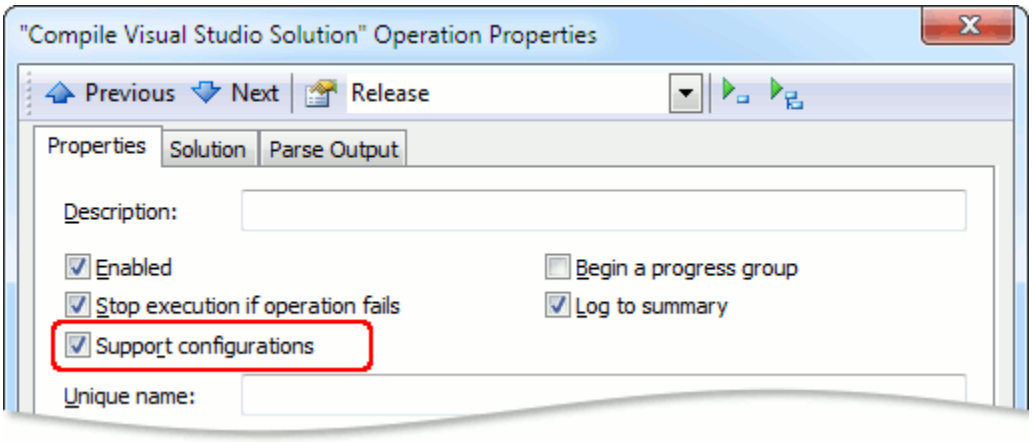
When a specific configuration is active, changes made to the operation properties are applied to that configuration only (see *Specifying Configuration-Dependent Values*).

You can also switch between different macro configurations while modifying operation properties, macro variables or constants in the appropriate dialogs. These dialogs have the **Configuration** list, so, you can quickly change the active macro configuration if necessary. Note, however, that the active configuration selected in these dialogs is preserved only while the dialog is open. After you close the dialog, the active configuration previously selected in the macro is restored.

The name of the active configuration is saved to the macro file, so, the next time you open a saved macro, the last used configuration will be active.

Specifying Configuration-Dependent Values

In order for a macro operation to support different property value sets in different macro configurations, its *Support configurations* property must be enabled:



Macro variables and constants are configuration-aware by default.

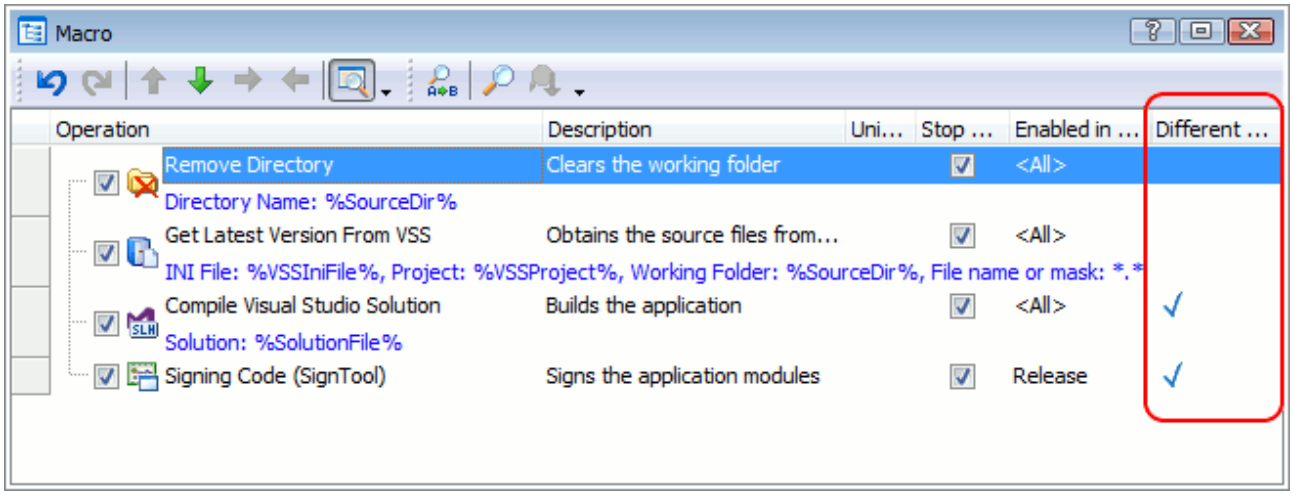
To set values specific for a configuration:

- First, activate the desired configuration.
- Then specify the needed values for operation properties, variables and constants.

  The changes you make will be applied to the selected configuration only, and will not affect the other configurations.

If you need to specify or modify a certain value in all the configurations at once, select *<All Configurations>* from the configuration list and make the desired changes.

Operations that have different property values in different configurations are indicated with a ✓ tick in the **Different Configuration Settings** column of the **Macro** panel. As a result, you can quickly identify operations that have configuration-specific property values and those having the same property values in all the configurations.



Automated Build Studio Client/Server and Web Interface

Automated Build Studio lets you start and monitor macros working on other computers in the network.

The remote computers where the macro is running are called ***servers***. The computers where you control the remote macro execution are called ***clients***.

Automated Build Studio is not divided into Client and Server parts. Once you installed Build Studio on your workstation, it can function like a server and a client. The server provides information needed for remote macro runs and the client sends commands to the server.

You can manage remote macros directly from the Client panel of Automated Build Studio. You can also manage macro runs without installing Automated Build Studio - the Automated Build Studio Web Interface installed on the server lets you control macro runs through your web browser.

To transfer data and commands between the server and client computers, the ***Automated Build Studio Service*** is needed. This service is shipped with Build Studio and is installed by the Automated Build Studio installation program. So, after you installed Automated Build Studio on a computer, this computer automatically becomes a *server*, which can be controlled by a remote instance of Automated Build Studio.

Automated Build Studio Client/Server Functionality

To start and monitor macros working on other computers in the network, use the **Client** panel in Automated Build Studio:

Connecting to the Server

To run a macro on a remote computer, you have to first connect to this computer. To do this:

- Switch to the Client panel.
- In the **Location** box displayed on the **Macro** page, specify the connection which you will use. You can either select an existing connection from the dropdown list, or create a new connection by clicking the Location box's ellipsis button and using the subsequent **Connections** dialog. In this dialog you can choose the desired connection type and specify the user name, domain and password for the connection.

Registering Macros

After you have connected to a server, you can choose the macro, which you will run on the remote computer. To do this:

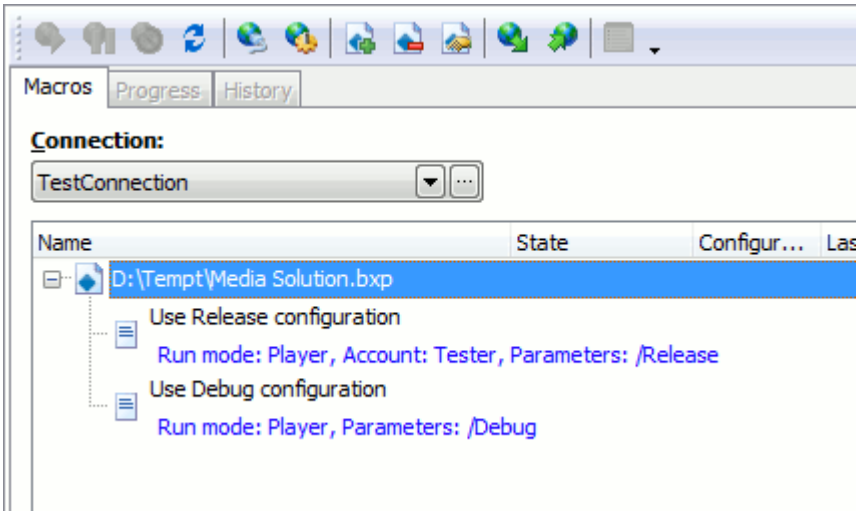
- Click  **Add Macro** on the toolbar or the Client panel, or right-click somewhere within the panel's Macro page and choose **Add Macro** from the context menu. This will call the **Add Macro** dialog.
- On the **General** tabbed page of the dialog, you can specify the macro file name, description, the utility to be used to execute the macro (you can choose either Automated Build Studio or Automated Macro Player) and the command-line arguments for the utility.

Note that the macro file name is specified in terms of the remote computer. There is no need to share the folder on the remote computer where the macro is located in order to access the macro file.

- On the **Identification** page you can specify the user name, domain and password that will be used to connect to the remote computer.

  Note that the specified account must have permissions to run executables on the server computer.
- On the **Permissions** page you can specify users or user groups, which are allowed to execute the macro and monitor the macro run. These users and groups must be registered on the server with the **Server Settings** dialog.
- Click **OK** in the Add Macro dialog to save changes and add the macro to the Client panel.

If you add the same macro several times, Automated Build Studio will create ***macro presets*** and display them as child nodes of the macro node. The presets differ from each other by the parameters specified in the Add Macro dialog.



Automated Build Studio also supports auto-registering of macros. With the *Automatically register all macros that are run on the server* option enabled, the tool will register any macro that will be started on the remote computer.

After you have added the macro to the panel, you can run it on the remote computer and monitor the macro run.

Running Macro and Monitoring the Run

To run a macro added to the Client panel:

- Choose the desired macro in the Macro page of the panel.
- Right-click the selected macro and select **Run** from the context menu or click the **Run** button on the panel's toolbar.

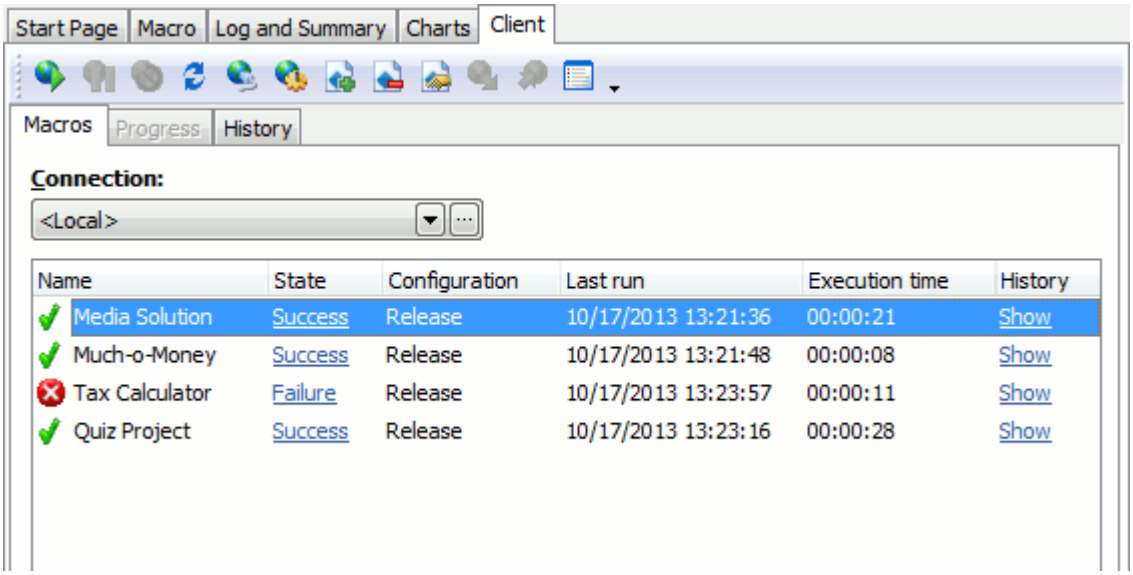
Automated Build Studio will start running the macro on the remote computer. The **State** column of the Macro page will indicate the run status.

To view detailed information on the macro's execution progress, select the desired macro in the Macro page and switch to the **Progress** page. This page indicates the execution progress using progress group defined in the macro being run. If the macro does not contain progress groups, the page will be empty.

Note: The Client panel will monitor all macro runs: those that will be initiated from the Client panel and those that will be initiated outside of it. In other words, you will see macro runs that will be started by a user who will log on the remote computer.

Viewing Results

After the macro execution is over, the **State** column of the Macro page will indicate the macro run results (*Failed* or *Success*). The **Configuration** column will show the configuration used to build the macro, **Last run** will indicate the date and time of the run start, **Execution time** - the run duration.



To view detailed information on macro run results, click the word (*Failed* or *Success*) displayed in the State column. Automated Build Studio will load the run results to the Log and Summary panel.

To view the history of macro runs, choose the desired macro in the Macro page and then click **Show** in the **History** column of that page. Build Studio will retrieve the history of the macro runs from the remote computer and then it will display the History page with the retrieved data.

Automated Build Studio Web Interface

Automated Build Studio's Client/Server Web Interface lets you control macros running on the server computer through your Internet browser (that is, without installing Automated Build Studio on client computers). The Web Interface module is installed on the server computer and generates web pages that you access through your Internet browser.

To view a page generated by the Web interface, open your Internet browser and navigate to the following page:

```
http://ServerName_or_IP/AutomatedBuildStudioWeb
```

As you can see, you can specify the server computer using either its name or IP address.

AutomatedBuildStudioWeb is the virtual directory holding the pages generated by the Web Interface module.

After you enter the specified address, you will see the **Login** page asking you to specify the authentication parameters for login. Specify the user name and password to log in as a local Automated Build

Studio user. Alternatively, you can enable the **Authenticate by OS** check box to use your current Windows logon settings.

The next page after the Login page is very similar to Automated Build Studio's Client panel. You can use it in the same manner as you use the Client panel. For more information, see *Using the Client/Server Functionality* and *Automated Build Studio Client/Server - Common Tasks* (in Automated Build Studio help).

Build Requests

Automated Build Studio provides users with the possibility to ***request*** build runs. This enables the users who cannot start builds directly due to insufficient permissions, trigger builds along with authorized users, just in a different way.

Users can submit build requests via a special form in Automated Build Studio's ***Web Interface***.

In this form, the users can choose one or several build macros (or macro presets) that they want to run, specify the request description and priority. The description can, for instance, include the list of bugs fixed, new features implemented or specify a reason why the user needs a new build. The request priority is analyzed by certain build triggers, which require higher-priority requests to start the build than lower-priority ones.

All submitted requests are collected by the server and processed by build triggers. Once the trigger condition is met, the build is started. You can monitor and manage the build execution in the same way as if you have directly started the build. For example, you can pause the build and resume it at a later time, view the build progress and so on.

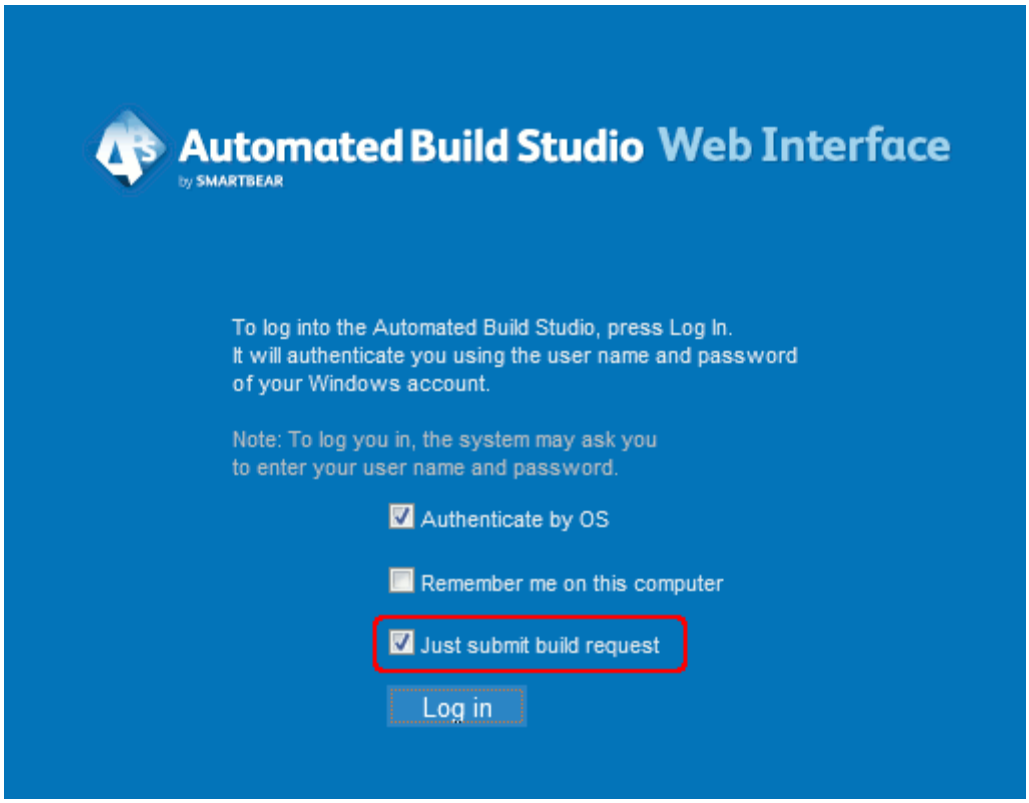
To submit a new build request

The way you can submit build requests depends on whether you are already logged into Automated Build Studio Web Interface or not.

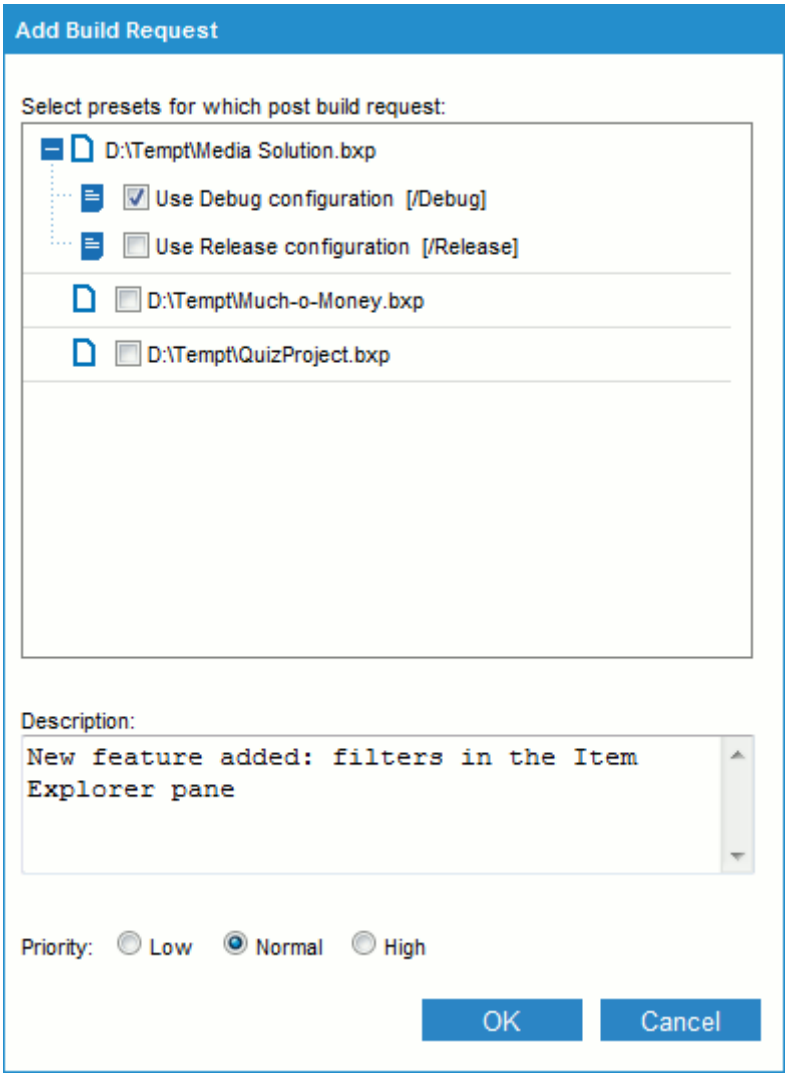
If you are not currently logged into Automated Build Studio Web Interface, follow these steps:

1. Launch your web browser and go to the Automated Build Studio Server Web Interface login page (the default address is *http://ServerName_or_IP/AutomatedBuildStudioWeb/*).
2. Enter your user name and password or check **Authenticate by OS** to log in using your Windows account.

3. Check the **Just submit build request** option (see the image below) and log in:



4. In the ensuing **Add Build Request** dialog, check the macros (macro presets) whose run you want to request, enter the request description and choose the request priority - *Low*, *Normal* (default) or *High*.



Click **OK** to submit your request.

5. To submit another request, repeat step 4.
6. After you have submitted all the needed requests, click **Cancel** to log off the Web Interface.

If you are already logged into Automated Build Studio Web Interface, do the following:

1. In the **Macros** section, select the macro (or macro preset) whose execution you want to request.
2. Switch to the **Requests** tab and click **Add Request**. The **Add Build Request** dialog appears.
3. In the dialog, check the macros (macro presets) whose run you want to request, enter the request description, choose the request priority and click **OK**.

   Two notes:

   - The request description is required; it is not possible to submit an empty request.
   - If the macro list is empty, it is most probably that you do not have the Run macros permission and you have enabled the *Hide non-available macros in build request dialog* option in the **Client Settings**. Disable this option and try submitting your request again.

Script Writing

General Information

The **Script** operation lets you include scripts in your macros. A ***script*** is a single script-language procedure or function.

A script can be written in one of the three supported scripting languages:

- DelphiScript
- VBScript
- JScript

All three supported scripting languages use only OLE-compatible data types and cannot use pointers.

DelphiScript support is built in Automated Build Studio. **VBScript** and **JScript** are supported by DLLs (of the same name) supplied with Windows or Internet Explorer. To run VBScript or JScript in Automated Build Studio, Internet Explorer 5.0 or higher must be installed on the machine. If you do not have IE 5 or higher, you can install the Microsoft scripting components directly.

For detailed information on scripting languages, see:

- VBScript - http://msdn.microsoft.com/en-us/library/ms950396.aspx
- JScript - http://msdn.microsoft.com/en-us/library/ms950396.aspx
- DelphiScript - DelphiScript Description in Automated Build Studio help

What can your scripts do in Automated Build Studio? They are used mainly to solve the following problems:

- To perform specific actions that cannot be executed with existing operations.
- To assign values to macro variables.

  Note: Macro variables cannot be used as loop variables. A loop variable should be a local script variable.

- To call operations existing in the macro and access their properties.
- To work with internal and external objects. Automated Build Studio's **Object Model** can help you better understand how you can use scripts to work with different objects.

Object Model

In most cases, scripts deal with internal objects of Automated Build Studio and external objects of other applications (via OLE/COM interfaces), process variables defined in the macro, call operations that exist in the macro and access properties of these operations. Automated Build Studio treats each variable, constant or operation as an object.

Each object can have one parent and one or several child objects. Automated Build Studio contains a number of top-level objects, which you can address in scripts directly. To get access to other objects you should use methods and properties of the top-level objects.

The top-level objects defined in Automated Build Studio are –

Object	Description
`ABSUtils`	Contains methods that perform operations specific to Automated Build Studio and provides scripting access to the all of the macro operations.
`BuiltIn`	Lets your scripts call routines that are built into Automated Build Studio. See *List of Built-In Routines* in the Automated Build Studio help.
`Constants` and `Variables`	Provides access to variables and constants defined for a macro in Automated Build Studio. See *Using Variables and Constants in Macros*.
`Log`	Use methods of this object to post custom messages to Automated Build Studio's **Log** panel from scripts.
`Operations`	Provides access to Automated Build Studio operations from scripts. In order for a script to call an operation, this operation must have a unique name.
`Utilities`	Makes certain functions and variables of the Borland VCL SysUtils unit available to your scripts. See *Utilities Object - List of Supported Functions* and *Utilities Object - List of Supported Variables and Constants* in the Automated Build Studio help.

Automated Build Studio Command-Line and Exit Codes

Command-Line Arguments

Automated Build Studio Command Line

Automated Build Studio (<*Automated Build Studio*>*Bin**AutomatedBuildStudio.exe*) uses the following command line arguments, which are case-insensitive:

file_name

Launches Build Studio and loads the specified macro (.bbp or .bxp file) into it.

/ns

Opens Automated Build Studio without displaying the splash screen.

/Run or /R

Must be specified after *file_name*. This will run the specified macro upon loading Build Studio. After the macro finishes, Automated Build Studio will stay open.

/RunAndExit or /RE

See */Run*. This option does the same as */Run* but closes Build Studio after the macro finishes.

/Multirun

If you run a macro from the command line (that is, if the command line contains the /Run or /RunAndExit arguments), Automated Build Studio will check whether the macro is already running. If it is, Build Studio will block the run from the command line.

To change this default behavior and enable running several instances of the macro concurrently, use the *Multirun* command-line argument. If this argument is specified, Automated Build Studio does not check whether the macro is running or not.

/SilentMode, /ErrorAndWarningMode or /ErrorOnlyMode

You can use one of these arguments to specify which message boxes and dialogs Automated Build Studio will display during a macro run:

- */SilentMode* - If this argument is specified, Automated Build Studio will neither display any dialogs nor inform you about any errors and warnings that occurred during a macro run.
- */ErrorAndWarningMode* - If this argument is used, Automated Build Studio will not display dialogs, but it will show error and warning messages.
- */ErrorOnlyMode* - If this argument is used, Automated Build Studio will not display any dialogs and message boxes except for error messages.

/variable_name=variable_value or -variable_name=variable_value

Initializes the specified variable with the given value. The variable must exist in the macro you are running. If the variable value holds spaces, the entire value must be enclosed in double-quotes. This command line argument is valid only if it is used with the */Run* or*/RunAndExit* arguments. To initialize several variables, you can use this argument several times in the same command line.

/configuration_name

Runs the macro in the specified *configuration*. If the configuration name includes spaces, it must be enclosed in double-quotes: */"configuration name"*. To batch-run the macro in several configurations at once, use this argument several times in the same command line. If this argument is not specified, the active configuration which is saved in the macro will be used.

/@all

Runs a macro using all existing macro *configurations*.

Note: To determine which parameters were passed to Build Studio in its command line, add a *Script* operation to you macro and use the `ParamStr` and `ParamCount` functions in the script.

Here are some examples of running Automated Build Studio with command line arguments.

- Use the following command line to run Automated Build Studio, load the specified macro (My.bbp) in it and start the macro execution for the *Release* configuration.

```
"C:\Automated Build Studio\Bin\AutomatedBuildStudio.exe" "C:\Wor
k\Macros\My.bbp" /Run /Release
```

- The following line commands Automated Build Studio to run without the splash screen, to load the specified macro (My.bbp), to start its execution, to initialize the *ProjectDir* variable and to exit when the macro execution is over.

```
AutomatedBuildStudio.exe "C:\Work\Macros\My.bbp" /ns /RunAndExit
 /ProjectDir="C:\My Projects"
```

Automated Macro Player Command Line

The installation of Automated Build Studio also includes Automated Macro Player. This is a console application (<*Automated Build Studio*>*\Bin\AutomatedMacroPlayer.exe*) that is used to run the macro you have constructed using Build Studio.

When calling the Macro Player via the command line, you can pass it the name of a file that holds a macro and the Macro Player will start running this macro immediately, using the active *configuration*. Because it is easier and faster than Build Studio, the Macro Player is a good solution to run the macros you have already checked thoroughly and proved to be reliable.

In addition to the macro file name, you can pass the Macro Player all the other parameters that are supported by AutomatedBuildStudio.exe except for /ns, /Run and /RunAndExit, which are simply ignored by the Macro Player.

Automated Build Studio Service Command Line

Automated Build Studio's installation also includes an *Automated Build Studio Service*. This is a Windows service (<*Automated Build Studio*>*\Bin\AutomatedBuildService.exe*) that is used to provide client-server functionality and continuous integration support. It dispatches data and commands between client and server machines, stores results of macro runs, traces task triggers and starts a task when the trigger events or conditions are accomplished.

AutomatedBuildService.exe accepts the following command-line arguments:

/trace:on and /trace:off

For some triggers, the Automated Build Studio Service can post logs on request to resources they trace. All logs are collected in the *<Automated Build Studio>\Bin\Extensions* folder, except for the *Microsoft Visual SourceSafe File Change* triggers' logs, which are posted to the VSSLog subfolder of this folder.

The log files' format depends on the trigger type, and it usually includes the entire command line of the launched process (it is used to check the state of the traced resource), the process response, the request date and time and the latest date and time of the traced resource obtained during the check. New records are appended to the log file that is stored for each trigger.

The */trace:on* argument enables logging of triggers' activity.

The */trace:off* argument disables trigger logging.

/poolsize:N

This argument sets the number of tasks (it can range between 1 and 30) that can be run simultaneously by AutomatedBuildService.exe. By default, this number is 5.

/service

Registers the AutomatedBuildService.exe process as a Windows service (if it is not yet registered) and runs this service. Before the service is run, it reads the values passed in the */poolsize* and/or */trace* command-line arguments.

/restart

Starts the Automated Build Studio Service if it is not running and restarts it if it is already running.

/stop

Stops the Automated Build Studio Service.

/exit

Use this argument with the */poolsize* and */trace* arguments to pass their values to AutomatedBuildService.exe when it is launched as a stand-alone application, not as a service. The argument passes values but does not run an instance of AutomatedBuildService.exe with newly set parameters.

Exit Codes

Both Automated Build Studio and Automated Macro Player provide the following exit codes, which report on the results of the last run:

Exit Code	Description
0	The last run was executed successfully.
1	An error occurred in a macro during the last run.

Normally, these codes are of use when Automated Build Studio or the Automated Macro Player is launched from a batch file.

More Information About Automated Build Studio

To get more information on advanced management features of Automated Build Studio or get a detailed description of all available operations, please refer to the following sections in Automated Build Studio Help.

Operation References
Provides a detailed description of all operations that exist in Automated Build Studio. Descriptions of operations are grouped by categories to which appropriate operations belong.

Scheduling Macro Runs
Describes how to schedule macro runs to trigger them at a specified time.

Script Writing
Provides detailed information on how to create scripts in macros.

Handling Errors in Macros
Describes how you can handle errors occurring during macro runs.

Debugging Macros and Scripts
Describes how to trace macro execution and find and fix errors in macros and scripts.

Exporting Macro Results
Describes how to export macro results displayed in the Log and Summary panels to a file.

Macro Libraries
Describes how to organize frequently used macros, submacros and operations into macro libraries and use these libraries in macros.

Creating Custom Operations
Describes how to create custom operations for Automated Build Studio. Custom operations will help you perform tasks that cannot be accomplished with standard operations.

Integration With Microsoft Visual Studio and MSBuild
Describes how Automated Build Studio integrates into Microsoft Visual Studio and extends various parts of this product.

Working With MSBuild Projects
Describes how to view and modify MSBuild projects in Automated Build Studio.

Working With Automated Build Studio via COM
Describes how to connect to and work with Automated Build Studio's subsystems via COM.

Working With Source Control System
Describes how to work with macros stored in a source control system directly from the Automated Build Studio IDE.

Index