ADVANCED WORKFLOW ENGINE (AWE) V10

User Guide
# Table of Contents

The Advanced Workflow Engine .......................................................................................................................... 7

The Concept ......................................................................................................................................................... 8

Architecture ........................................................................................................................................................ 8

Components ....................................................................................................................................................... 8

What's New in the Advanced Workflow Engine v10 ............................................................................................ 9

New and Improved Actions .............................................................................................................................. 12

New Activities ................................................................................................................................................. 18

Features & Benefits ........................................................................................................................................ 21

Installing the Advanced Workflow Engine ....................................................................................................... 23

Task Builder (AWE Interface) ............................................................................................................................ 23

Task Builder - Overview ..................................................................................................................................... 25

Task Builder Quick Tour ................................................................................................................................... 26

User Interface .................................................................................................................................................... 28

Task Builder Ribbon .......................................................................................................................................... 28

Task Builder Actions Panel ............................................................................................................................... 29

Task Builder Steps Panel ................................................................................................................................ 30

Task Builder Status Bar .................................................................................................................................. 33

Debug Panel ..................................................................................................................................................... 34

Debug Panel - Attachments ............................................................................................................................... 35

Debug Panel - Breakpoints ................................................................................................................................. 38

Debug Panel - Comments .................................................................................................................................. 38

Debug Panel - Labels ......................................................................................................................................... 39

Debug Panel - Output ......................................................................................................................................... 41

Debug Panel - Regions ...................................................................................................................................... 42

Debug Panel - Stack .......................................................................................................................................... 43

Debug Panel - Variables .................................................................................................................................... 43

Debug Panel - Watches .................................................................................................................................... 46

Testing Tasks Using Run Options ..................................................................................................................... 47
# Table of Contents

- **Debugging Tools & Techniques** .......................................................... 49
  - Breakpoints .......................................................................................... 50
  - Comments ............................................................................................ 51
  - Regions .................................................................................................. 53
  - Watches ................................................................................................ 55
  - Error Handling ..................................................................................... 56
  - Setting Task Builder Debug Toolbar Options ...................................... 60
- **Task Builder Options** ........................................................................... 61
  - Task Builder Color Options .................................................................. 63
  - Task Builder Debugger Options .......................................................... 65
  - Task Builder Font Options .................................................................... 67
  - Task Builder Formatting Options ......................................................... 69
  - Task Builder General Options ............................................................. 70
  - Task Builder Toolbar Options ............................................................. 73
- **About Creating Tasks** .......................................................................... 75
- **Creating Tasks (Workflows) for Use in Event Rules** .............................. 75
- **Using Advanced Workflows in Event Rules** ........................................ 77
  - Adding Task Steps .............................................................................. 79
- **Creating Variables** ............................................................................ 81
- **Percent Signs in AWE** ........................................................................ 85
- **Task Variables** .................................................................................. 87
- **Using Regular Expressions** ................................................................. 88
- **Using the Expression Builder** ............................................................. 93
- **File Masks & Wildcards** .................................................................... 95
- **Setting Custom Step Description** ..................................................... 97
- **Practical Usage** ................................................................................. 97
- **On Error Tab** .................................................................................... 99
- **Error Causes Tab** .............................................................................. 103
- **Attachments** ..................................................................................... 106
- **Auditing Workflows** ......................................................................... 107
Advanced Workflow Engine v10

Backing Up Tasks ........................................................................................................................................... 107
Bookmarks ..................................................................................................................................................... 108
Copying and Reusing Task Steps ................................................................................................................. 109
Deleting Workflows in EFT ............................................................................................................................. 111
Editing Task Steps ........................................................................................................................................... 111
Finding & Replacing Text ............................................................................................................................... 115
Importing Workflows (Tasks) ......................................................................................................................... 115
Print Task Steps .............................................................................................................................................. 116
AMError ............................................................................................................................................................... 116
AMError.ErrorDesc ........................................................................................................................................ 117
AMError.ErrorNumber ................................................................................................................................. 118
AMError.FunctionName ............................................................................................................................... 118
AMError.IPAddress ....................................................................................................................................... 118
AMError.MachineName ............................................................................................................................... 119
AMError.ParentTaskName ........................................................................................................................... 119
AMError.StepNumber ......................................................................................................................................... 120
AMError.StepText ......................................................................................................................................... 120
AMError.TagLine .............................................................................................................................................121
AMError.TaskName ........................................................................................................................................121

Actions .............................................................................................................................................................. 123

My Actions ..........................................................................................................................................................126
Finding & Organizing Actions ........................................................................................................................... 130
Action Properties ............................................................................................................................................... 133
Examples ............................................................................................................................................................. 134
Sample Workflows .......................................................................................................................................... 135
Example: Downloading Files from a Remote Server .................................................................................. 136
Example: Archiving Files .............................................................................................................................. 137
Example: Using Excel Actions ...................................................................................................................... 139

AML (Markup Language) ............................................................................................................................... 140

AML (Automation Markup Language) ........................................................................................................ 140
# Table of Contents

- Arrays .............................................................................................................................................................. 141
- Shared Arrays ................................................................................................................................................ 143
- Constants ....................................................................................................................................................... 145
- Datasets .......................................................................................................................................................... 147
- Expressions .................................................................................................................................................... 150
- Functions & Extended Functions ................................................................................................................ 153
- Variables ......................................................................................................................................................... 158
- Shared Variables ............................................................................................................................................ 160

## BASIC Scripting

- BASIC Script IDE ............................................................................................................................................... 167
- References Dialog .............................................................................................................................................. 168
- References Overview ........................................................................................................................................ 169
- UserDialog Editor .............................................................................................................................................. 170
- BASIC Scripts & Custom Functions ................................................................................................................ 170
- Regular Expressions ............................................................................................................................................ 172

Index................................................................................................................................................................. 175
The Advanced Workflow Engine

With EFT's Event Rules, you can configure EFT to perform an action automatically when a specific event occurs. You can use AWE Actions to design scripts, batch files, macros, or any other code-intensive process using an easy drag-and-drop interface.

Similar to EFT's Commands, EFT's Advanced Workflows add additional automation capabilities, allowing you to add scripting and variables to workflows, then add these reusable workflows to Event Rules. A workflow is a series of steps that can perform file transfers, batch data processing, application testing, and so on, set to run automatically when started by some event. A workflow can be composed of a single step or hundreds of steps.

You create a workflow, set priorities, security, error handling, and other properties for workflows from within the AWE interface. A workflow can be run, changed, or deleted through the AWE interface.

When you click the Advanced Workflows node in the Server tree, the right pane displays a list of defined workflows sorted in alphabetical order. If no workflows are defined, you can right-click the Advanced Workflows node to create a new workflow. You can edit or remove workflows in the right pane.
You can insert *Expressions* into a workflow to resolve dynamic data at runtime. Expressions can include functions, variables, constants, and operators. EFT reads the text found between percentage signs and then replaces the Expression (including the percentage signs) at runtime. For example, the expression [%USER.LOGIN%] is replaced with the user logon name.

**The basic steps of creating an Advanced Workflow task are simple:**

1. Create a workflow.
2. Add steps to the workflow in the AWE Task Builder.
3. Add the workflow to one or more Event Rules.

**The Concept**

The fundamental concept of The Advanced Workflow Engine (AWE) is the automation of front and back-office business processes. AWE is a software platform for building, managing, and launching automated tasks within EFT Event Rules. Tasks are developed via drag-and-drop without writing a single line of code. Simply drag together each action object in the sequence it should be executed.

**Architecture**

AWE breaks down business processes into "visual" steps with the use of "actions." Multiple steps combined together are called a "task." Actions are used to instruct AWE how to carry out a task. A "trigger" is a condition or event in Windows that essentially directs to execute a task automatically.

Task - The primary and most important object, a task is composed of a series of steps based on actions. Tasks can be designed to any of a very wide spectrum of business processes.

Actions - The available processes that can be used to build the steps to be carried out when a task is run. More than 300 actions are available from which to build tasks. These actions are assembled visually (without the use of code) via drag-and-drop in the Task Builder interface. Each action used to construct the task becomes a step. Upon execution, these steps are performed in a sequential order.

**Components**

The three main components are as follows:

Task Builder - Used to visually assemble actions into sequential task steps to be carried out. Additionally, the Task Builder is fully equipped with a wide variety of debugging tools to easily examine and troubleshoot problem tasks, eventually ensuring maximum reliability when the task is put into production.

EFT Event Rules - The central tool used to administer and customize all aspects of AWE. EFT Event Rules organize tasks, connect to remote installations, and set task properties. In addition, it is used to administer global settings such as the default mail server, default logon properties, database connections, and much more.

Advanced Workflow Action - Launches tasks according to the triggers attached to them.
What's New in the Advanced Workflow Engine v10

Modernized Interface

The Task Builder has been graphically redesigned to make it easier for developers to find and use the full range of features and elements provided while preserving an uncluttered workspace that reduces distraction. The newly designed ribbon contains tabs that are composed of groups, which are a labeled set of closely related commands. This helps users easily find, understand, and use commands efficiently and directly, with a minimum number of clicks and with less need to resort to trial-and-error. The Ribbon also encompasses a Quick Access Toolbar that houses the commands and buttons that are used most often within easy reach.

Improved action and activity library

The Task Builder’s Actions panel now comprises a comprehensive library of more than 600 pre-built actions/activities, eliminates the need for writing scripts and batch files altogether. The bottom portion of the Actions panel contains an intuitive search dialog that enables you to define search criteria and provide more efficient and effective searches. It supports dynamic search filtering, allowing you to enter all or part of an action’s name and view only actions that contain matching text. The list is updated as you type each letter.
Additionally, a new My Actions pane enables easy access to favorite, most recent and most frequently used actions and activities as well as activities designated as snippets.

Snippets
Snippets provide an easy way to implement frequently used code into a task. Instead of re-entering the properties for a specific activity or series of activities every time they’re added as task steps, users can save the steps as a snippet and simply drag and drop the snippet wherever it is needed in any task. A snippet can consist of one or more steps. Once a snippet is added to a task, its contents act as normal steps (e.g., steps can be reorganized or deleted or properties can be modified as needed). By using folders, users can easily organize common or related snippets into categories, creating a cleaner development environment.

Modernized action editor
You set the properties of individual activities via an Action Properties dialog, also known as action editor. The goal for action editor interface design is to make it easier for developers to access the range of properties and parameters supported for each activity as well as reduce redundancy. As a result, the new action editor is redesigned to provide easier and quicker ways to edit a task by focusing on the properties that are of most importance. The left portion of the editor now contains the full list of consolidated activities common to a given action. This enables jumping from one activity to another without the need to re-enter required parameters or re-open the action editor.
Updated Expression Builder

The look and feel of the Expression Builder has been updated to a more modernized theme. In addition, it now encompasses a search filter, allowing users to easily find the object they are looking for without the need to scroll up and down long lists.
Sessions

Many actions now support sessions that contain pre-configured credentials and properties that are shared amongst associated activities within a given session. Linking a combination of related activities to a single session eliminates redundancy. Additionally, a task supports construction and simultaneous execution of multiple sessions, improving efficiency.

New and Improved Actions

PDF

The PDF action allows you to create and share step-by-step activities that make complex or repetitive PDF related tasks easy for virtually anyone to execute consistently. Whether you’re dealing with a single PDF or a large batch of files, the prebuilt PDF activities let you run a sequence of procedures automatically. Activities include the ability to create, sign, stamp, encrypt, decrypt, extract, search and replace PDF files (to name a few).
WMI

The WMI action can monitor and control managed resources on a local or remote computer, thus, improving manageability of computers in a networked environment. For instance, a WQL Query task can initially determine any issues and direct itself to perform proper activities to correct the problem before proceeding to other steps.

PowerShell

Windows PowerShell is a task-based command-line shell and scripting language designed especially for system administration. PowerShell’s framework allows administrators to perform tasks on both local and remote Windows systems from the command-line. PowerShell - Run script action enables you to automate the management and control for PowerShell processing. With the use of the PowerShell action, you can embed existing PowerShell scripts to a task or point to an external .PS1 file to allow for seamless integration with other actions.
Amazon DynamoDB

Amazon DynamoDB is a fully administered NoSQL database service that provides fast, reliable performance, allowing users to create database tables that can store and access any amount of data to serve any level of traffic without downtime or performance degradation. Developers can automate a variety of DynamoDB operations without the need to write code. Since DynamoDB is a service based on throughput, developers can also create tasks to monitor usage and storage capacity.
Amazon SES Action

Amazon SES provides an easy, flexible, and low-cost way to deliver e-mails from the cloud, while reducing the possibility of legitimate e-mail being classified as spam. It integrates effortlessly with other Amazon Web Service (AWS) products, can send a broad range of business, marketing, and subscription messages, and easily handles large volumes of e-mail. With Amazon SES action, developers can automate the creation and transmission of e-mails as well as retrieve and verify identities for a specific AWS account. In addition, it allows creation of unique sessions used to run specific SES operations. Multiple sessions can exist in a single task allowing simultaneous SES executions to take place.
Images are an essential element of many modern businesses due to their ability to draw attention and create a long-lasting impression, thus, more organizations are learning the benefits of adding a visual component to their social media strategies. Because images are larger in size than textual data, they can eat away at available hard drive space and create bottlenecks in networks. In addition, developers and designers must deal with editing and converting many images by hand, which can be time consuming. Image action consists of activities that can greatly reduce the time it takes to perform such a process. For example, you can create a single task that reduces the size of an image, applies a filter for a particular effect and converts the image file to the desired format. In addition, you can integrate loop related activities to run this task on hundreds or thousands of images.

In many situations, data required for automation are no longer available as text, but rather, only as pixel images. As a result, the Image action also includes a new Image recognition activity, which makes automating tasks in a graphical environment much more accurate. Image recognition allows you to search for images as well as image patterns. It can be used to quickly search for duplicate images within folders and sub-folders that may contain a considerable amount of image file or used as a tool to specify coordinates as a way to precisely identify the location of an image.
SharePoint

With the use of the SharePoint action, organizations have the power to automate the creation, management and implementation of SharePoint sites that are discoverable throughout the organization. By further streamlining and automating SharePoint processes, a company’s resources can be allocated to other areas, which can lead to reduced costs and greater operational efficiency.
OpenDocument Spreadsheet (ODS)

The OpenDocument Spreadsheet action enables creation and management of spreadsheets without requiring Microsoft Excel to be installed on the system. This action not only supports spreadsheet generation and other basic file formatting features, but also supports a number of advanced features that make it much more convenient for developers to manipulate spreadsheet contents, format cells and protect files. More importantly, it can process Excel files dynamically, which enables developers the ability to automate hundreds or thousands of spreadsheets very quickly.

New Activities

Dialog - Custom

The Dialog action supports predefined dialog boxes that can convey specific information or request user input during task execution. However, a custom dialog box may be required that, for example, presents the user with specific options and then runs the appropriate steps based on the option they chose. In such cases, developers can use the new Custom dialog activity to design a custom dialog to fit their needs. A variety of controls are supported, each containing properties that can be viewed or modified via the Dialog Designer (displayed below), an intuitive graphical interface used to easily design custom dialogs.
File System - CSV to dataset

The File system action now contains a new CSV to dataset activity that enables users to automate the movement of tabular data between programs that natively operate on incompatible (often proprietary) formats. For example, a user may need to transfer information from a database program that stores data in a unique format. The database program most likely can export its data in CSV (comma-separated-values) format. AWE can then read the exported CSV data onto a dataset and eventually import the dataset values onto the spreadsheet program.

Excel - Run Macro

A macro is typically a set of instructions produced by recording a sequence of keyboard and mouse actions using the Excel Macro Recorder. If you perform a task repeatedly in Microsoft Excel, you can eliminate repetition by creating a macro that automates the task, however, the macro can only be started manually by some type of user interaction. Excel action contains a new Run macro activity that resolves this issue by allowing such a process to be automated without the need to write a single line of code. You can automatically start existing macros on a set schedule or upon the occurrence of an event or condition, greatly speeding complex spreadsheet operations.

Registry - Get Values

Extra functionality has been added to the Registry action with the addition of the Get registry value(s) activity. With this activity, you can retrieve the values contained within a specific registry key and save the results onto a dataset. You can, for example, determine whether a given value currently exists in a specific registry key and perform other actions as a result.
Loop - Values

The Loop action adds additional functionality to registry operations with a new Loop values activity. This activity loops through the values contained in a registry key and populates a dataset with results. It is useful when you want to operate on a collection registry values or sub-keys, one at a time.

Amazon S3 - Put Object

The Amazon S3 action contains a new Put Object(s) activity. This implementation of the PUT operation adds (uploads) an object to a bucket. You can optionally specify the accounts or groups that should be granted specific permissions on your object(s) with the use of a canned ACL request header. Also, to ensure that data is not corrupted traversing the network, use the Content-MD5 header. This activity also supports the use of a Content-MD5 header to ensure that data is not corrupted traversing the network. When you use this header, Amazon S3 checks the object against the provided MD5 value and, if they do not match, returns an error. Additionally, you can calculate the MD5 while putting an object to Amazon S3 and compare the ETag returned in a dataset to the calculated MD5 value.

Email - Archive messages

Archiving e-mail messages can clean your inbox by getting messages out of your way, and at the same time, protect the data contained in your e-mail. Email action encompasses a new Archive message(s) activity that allows you to easily automate the process of preserving and protecting your e-mail messages (as well as attachments and metadata) so they can be accessed at a later date, should the need arise. This activity contains a filtering function that allows archiving of specific messages based on the specified filter. The benefits of email archiving include the recovery of lost or accidentally deleted messages, accelerated audit response, preservation of the intellectual property and “eDiscovery” in the case of a court trial, lawsuit, legal proceeding or internal investigation.

If - Contains Text

The If - Contains text activity is commonly used to check the contents of a piece of text and perform conditional actions depending on whether it is found. It can determine if the text to look for is contained within the target text and returns TRUE or FALSE depending on the outcome. Optionally, it can determine whether the target text starts with or ends with the text to look for.

Array - Set

The Array - Set activity sets the value of one or more elements in an existing array. In addition, it can optionally add new index/value pairs to an existing array. This activity can be used in conditional-based tasks to add or modify array values on the fly.

Various Text related Activities

Manipulating text strings is one of the most common tasks performed in scripts. The Text action encompasses a handful of new activities that enables a more precise manipulation of text. They include the following:

Convert case - Converts the specified text to upper or lower case letters and populates a variable with the results.
**Reverse** - Reverses a text string and populates a variable with the results.

**Format path** - Formats specific parts of a file path to reveal precise data.

**Base64 encode/decode** - Encodes or decodes a block of text using Base64 algorithm.

**Features & Benefits**

The Advanced Workflow Engine (AWE) is a powerful tool that allows rapid construction of Windows automation routines via drag and drop without any programming. It replaces legacy batch files, scripts and custom application development, enabling complete IT Process automation without writing a single line of code. AWE provides an incredibly easy-to-use, intuitive interface for developing automation applications. The basic building blocks are called actions, which are plain-English, drag-and-drop, fill-in-the-blank tools developers use to build Tasks. Developers simply drag-and-drop actions in Task Builder to build a series of Steps which collectively make up the task.

**Available Actions & Activities**

Includes 100s of actions and activities that involve extensive support for FTP/SFTP, SQL, Email, HTTP, VB Scripts, VMWare Host and VMWare Guest, Services, Environment Variables, Exchange, MSMQ and many more. This list does not begin to address all the possible automated solutions that can be constructed and deployed. A task can contain any number of actions/activities in any order, creating limitless possibilities.

**Intuitive Task Development & Debugging**

The Task Builder is an intuitive interface for developing automation applications. It is used to visually construct and examine the steps that a task should carry out when it is run. To ensure that newly created tasks are working properly before they are put into production, the Task Builder also includes a variety of testing and debugging features that outputs real-time information about a running task. They enable monitoring of each step during task execution, allow inspection of variables, datasets and other dynamic data and aid developers with troubleshooting important aspects of a task as it is being constructed.

**Variable Support**

Includes full support for variables that are used to pass dynamic data during task execution and also supports one and two-dimensional arrays for implementing more complex logic to tasks. Additionally, many actions support the creation of datasets, which can hold multiple rows and columns of dynamic data. This is useful when retrieving data that describes a collection of information such as a database or spreadsheet.

**Replaces Code**

Reduces costs and hassles associated with developing and maintaining code, scripts, and batch files. For developers, increases the speed of development without sacrificing power, saving time and future code maintenance hassles. For non-developers, provides the simplicity to complex processes without the need for code or syntax.
Ease of Use

Provides an incredibly user-friendly, yet intuitive set of interfaces allowing you to visually create, manage, and execute your automated tasks without requiring any programming expertise. Instead of writing scripts and batch files, business and IT processes are created using the Task Builder interface which contains over 300 prebuilt actions and activities that you can easily drag-and-drop together to form powerful automated tasks.

Saves Time & Reduces Errors

Network managers and IT administrators can eliminate time spent on repetitive IT processes so they can focus on other aspects of their jobs. Additionally, automation eliminates errors introduced during manual processes that often result in network downtime, delayed access to data, and user complaints. Streamline inefficient processes and operations with reliable, repeatable, automation and eliminating errors introduced by process delays and repetitive worker activities.

Accelerates flow of information

Drag-and-drop building enables tasks to be constructed rapidly. This accelerates data flows, allowing business users and decision-makers to become more effective and efficient by delivering real-time access to accurate data.
Installing the Advanced Workflow Engine

The Advanced Workflow Engine (AWE) is installed with EFT Enterprise and is included in the 30-day trial of EFT, during which you can create, edit, and execute powerful workflows using EFT’s Event Rule system. After the 30-day trial has expired, you can still create new workflows and edit existing ones; however, you will no longer be able to execute the workflows with EFT’s Event Rules. After you purchase and activate a serial number for the AWE module, you will once again be able to use the workflows in Event Rules. The workflows are not deleted until you delete them.

Refer to the EFT Enterprise EULA for information about Advanced Workflow Engine licensing.

System Requirements

AWE requires Microsoft .NET Framework 4.0 and EFT Enterprise v7.4.5 or later.

Activating/Registering the AWE Module

The AWE module is registered in the EFT administration interface. Please refer to the activation topic in the EFT help for details.

Task Builder (AWE Interface)

The Task Builder is an intuitive project designer used to visually assemble and test the activities that the task should carry out when it is executed (either manually or automatically by a trigger). Task steps are created by visually dragging-and-dropping activities from an actions palette and placing them in the sequence desired for execution, eliminating the need to write code or create batch files. Additionally, the sequence of steps are described in plain English and presented in a way that clearly describes what they do.
The Task Builder includes a variety of testing and debugging features that displays verbose, real time data about a task, which aids a developer in monitoring and debugging tasks as they are being constructed. This ensures that newly created tasks are functioning properly before they are put into production. For instance, a developer can perform a series of “test runs” within the Task Builder and use a variety of intuitive debugging and diagnostic features to determine the cause of failing or problematic tasks.

### Sections of the Task Builder

The Task Builder interface is comprised of various sections (also known as panels or window panes). The table below lists each (in alphabetical order) along with a description.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Builder Actions Panel</td>
<td>Encompasses all of the actions and activities available in used to construct the steps of a task. Also contains My Actions which saves a list of your favorite, recently used and mostly used activities. It also allows creation of snippets, which are copies of specific steps that contain pre-defined properties and settings.</td>
</tr>
<tr>
<td>Task Builder Debug Panel</td>
<td>The Debug panel contains several views which are separated by tabs. Each view is comprised of a different debug tool. A different pop-up menu will appear for each view displaying the options available for that particular debug tool.</td>
</tr>
<tr>
<td>Task Builder Ribbon</td>
<td>The ribbon spans the top of Task Builder, directly below the title bar. It is filled with graphical representations of control elements which are grouped by different functionality. Commands are organized in logical groups, which are collected together under operation related tabs.</td>
</tr>
<tr>
<td>Task Builder Status Bar</td>
<td>Located at the bottom of Task Builder and displays real-time progress of a running task through graphics and text.</td>
</tr>
<tr>
<td>Task Builder Steps Panel</td>
<td>The Task Builder’s Steps panel displays the actions/activities selected from the Actions Panel that will be carried out when the task executes.</td>
</tr>
</tbody>
</table>
Task Builder - Overview

The Task Builder is an intuitive project designer used to visually assemble and test the activities that the task should carry out when it is executed (either manually or automatically by a trigger). Task steps are created by visually dragging-and-dropping activities from an actions palette and placing them in the sequence desired for execution, eliminating the need to write code or create batch files. Additionally, the sequence of steps are described in plain English and presented in a way that clearly describes what they do.

The Task Builder includes a variety of testing and debugging features that displays verbose, real time data about a task, which aids a developer in monitoring and debugging tasks as they are being constructed. This ensures that newly created tasks are functioning properly before they are put into production. For instance, a developer can perform a series of “test runs” within the Task Builder and use a variety of intuitive debugging and diagnostic features to determine the cause of failing or problematic tasks.

Sections of the Task Builder

The Task Builder interface is comprised of various sections (also known as panels or window panes). The table below lists each (in alphabetical order) along with a description.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Builder</td>
<td>Encompasses all of the actions and activities available in used to construct the steps of a task. Also contains My Actions which saves a list of your favorite, recently used and mostly used activities. It also allows creation of snippets, which are copies of specific steps that contain pre-defined properties and settings.</td>
</tr>
<tr>
<td>Actions Panel</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Task Builder Debug Panel</td>
<td>The Debug panel contains several views which are separated by tabs. Each view is comprised of a different debug tool. A different pop-up menu will appear for each view displaying the options available for that particular debug tool.</td>
</tr>
<tr>
<td>Task Builder Ribbon</td>
<td>The ribbon spans the top of Task Builder, directly below the title bar. It is filled with graphical representations of control elements which are grouped by different functionality. Commands are organized in logical groups, which are collected together under operation related tabs.</td>
</tr>
<tr>
<td>Task Builder Status Bar</td>
<td>Located at the bottom of Task Builder and displays real-time progress of a running task through graphics and text.</td>
</tr>
<tr>
<td>Task Builder Steps Panel</td>
<td>The Task Builder’s Steps panel displays the actions/activities selected from the Actions Panel that will be carried out when the task executes.</td>
</tr>
</tbody>
</table>

**Task Builder Quick Tour**

The Task Builder is used to visually construct and examine the steps that an task should carry out when it is run. Task steps are created by visually dragging-and-dropping activities from an actions palette and placing them in the sequence desired for execution. This eliminates the need to write code or create batch files. A task can be composed of a single activity or hundreds of activities, depending on what it is designed to perform.

To ensure that newly created tasks are working properly before they are put into production, the Task Builder includes a variety of testing and debugging features that provide detailed, real time information about a running task, which aids a developer in monitoring and debugging tasks as they are being constructed.
What the Task Builder Accomplishes

Task Builder is the primary interface used to construct and test the steps of a task. It contains a variety of debug tools to allow developers to conveniently examine tasks while they are being constructed. Task Builder operations include the following:

- Simple drag-and-drop task construction
- Edit steps of a task
- Use search feature to find actions/activities
- Print or e-mail task steps
- Test tasks using available Run options
- Viewing debug data via the Debug panel
- Creating variables, arrays and other expressions via the Expression Builder

What the Task Builder Does NOT Accomplish

The Task Builder is used to build and test task specific steps that should be carried out when the task is run (or triggered automatically). However, it does not handle the triggering (such as scheduling the task to run automatically) or system settings. The EFT Event Rule system handles the triggering through Event Rules.

The Task Builder is used to visually construct and examine the steps that a task should carry out when it is run (either manually or automatically with the use of an event/condition). Task steps are created by visually dragging-and-dropping activities from an actions palette and placing them in the sequence desired for execution. This eliminates the need to write code or create batch files. A task can be composed of a single activity or hundreds of activities, depending on what it is designed to perform.
To ensure that newly created tasks are working properly before they are put into production, the Task Builder includes a variety of testing and debugging features that provide detailed, real-time information about a running task, which aids a developer in monitoring and debugging tasks as they are being constructed.

**User Interface**

This section describes the features of the Task Builder interface.

- **Ribbon**
- **Actions Panel**
- **Steps Panel**
- **Debug Panel**
- **Status Bar**

**Task Builder Ribbon**

The Ribbon is a contextual interface designed to increase efficiency and make it easier for users to find features and quickly access popular commands and controls during task construction. The ribbon spans the top of Task Builder, directly below the title bar. It is filled with graphical representations of control elements that are grouped by different functionality. Commands are organized in logical groups, which are collected together under operation-related tabs. The Ribbon brings the most common commands to the forefront, in plain sight, so you no longer have to roam aimlessly through the clutter of menus, sub-menus, and toolbars searching for what you want.

Enlarging or maximizing the Task Builder window auto-expands the ribbon vertically. To add additional space, the ribbon can be collapsed so only the tabs appear. This can be accomplished by clicking the Collapse arrow (on the lower right of the ribbon). Because every option will not fit on the ribbon, drop-down menus are available that are represented by small arrows below certain buttons/commands. Clicking such arrows displays a drop-down menu that contains related sub-commands. For example, if you click the arrow below the Run icon, a drop-down menu will appear displaying related commands such as Run All, Run Selected, Run From Here, and Step (shown below).
Task Builder Actions Panel

A task is something that performs a number of sequential steps that are necessary to reach a desired outcome. The Actions panel houses all of the available actions and associated activities that make up the steps to be performed during task execution. Actions and activities are assembled visually by dragging them from the Actions panel onto the Steps panel (or by simply double-clicking the desired action/activity) and entering its properties. When this is accomplished, the action becomes a task step. This panel also contains a My Actions view that puts your favorite, recently used, and most used activities along with any saved snippets in a convenient location for easy access.

Actions View

The Actions view is comprised of more than 300 built-in actions and activities, which range from starting an application, opening a web page, focusing on a window, uploading or downloading files, sending and receiving e-mail, simulating mouse or keyboard activity and much more. Actions and activities are arranged in a hierarchical format so that users can browse through them conveniently. To view a list of activities (or sub-actions), expand the main action by double-clicking its associated icon or by clicking the small arrow to the left of the action.
Once the desired activity is found, simply drag it onto the Steps panel or double-click it to make it a task step. In addition, the action itself can be dragged into the Steps panel. Doing so will display the properties dialog where an activity can be selected from a list.

**NOTE:** Unlicensed actions and activities will appear opaque. Double-clicking, dragging or adding them to the Steps panel in any other way will generate an error.

---

**My Actions View**

The My Actions view allows easy access to your most-often-used activities via Jump Lists. That way, you don’t have to scroll through numerous actions or type a keyword in the Search bar to find an activity. Simply select the activity from the appropriate list in the My Actions panel.

---

**Task Builder Steps Panel**

The Task Builder’s Steps panel displays the actions/activities that are selected from the Actions panel that will be carried out when the task executes. During task construction, actions and activities are dragged from the Actions panel onto the Steps panel. During this process, a dialog box displaying the properties of the selected action/activity opens and specific parameters and requirements can be viewed or modified. Once parameters are set, the action/activity and its properties are then displayed in the Steps panel and the action becomes a task "step". These steps run automatically in a sequential order when the task is triggered by some event or condition or when started manually by the user.

When creating/debugging external tasks, a single instance of the Task Builder can open multiple tasks, each separated by tabs located at the bottom of the Steps pane. Each tab is labeled with the name of the corresponding task. The steps of the desired task can be viewed by simply selecting the proper tab. When a task is selected, the Steps pane becomes populated with steps of that task and the Status Bar and Debug Pane show data for that task.

By default, a task step is comprised of the following:

- Step number
- Unique icon associated with the action
- Visual or AML description of the step (depending on which view is selected from the Ribbon’s View command group).

Alternatively, step numbers can be omitted, icon sizes can be modified or step descriptions can be fully customized. See Task Builder Editor Options for more details. See Custom Step Description for more information about customizing the default description that appears on the Steps pane.

**NOTE:** If you import a task that includes an unlicensed action/activity, the Steps panel will display the unlicensed step (in Visual view) as currently unlicensed. Running a task with an unlicensed step will always fail at that step. You can bypass the step by disabling it or removing it altogether.
Task Functions/Variables Panel

In order to make the creation and maintainability of tasks easier and safer, now supports variable scoping, variable accessibility as well as external tasks and function declarations and execution. The top portion of the Steps pane contains the Functions & Task Variables drop-down menu.

This drop-down panel is used to create Functions and Variables and display their properties in a clear and concise manner so the user understands the steps being displayed at any given time are associated with the function/variable being modified.

Context Menu

Right-clicking one or more lines of code on the Steps pane opens a pop-up menu that displays a list of available options to perform for those steps. The table below contains a list of available Steps pane right-click menu items.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
<th>Hot-Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Cuts the selected step(s). To select multiple steps, hold down CTRL during selection. To select all steps, press CTRL+A or click the Select All button from the Find command group of the Ribbon’s Home tab.</td>
<td>CTRL + X</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies the selected step(s) to the clipboard. To select multiple steps, hold down CTRL during selection. To select all steps, press CTRL+A or click the Select All button from the Find command group of the Ribbon’s Home tab. Copied steps can be pasted in the Steps pane of the current task, another task or an application such as Notepad.</td>
<td>CTRL + C</td>
</tr>
<tr>
<td>Copy Description</td>
<td>Copies the description of the as they appear in Visual view</td>
<td></td>
</tr>
<tr>
<td>Drag steps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Menu Item</td>
<td>Description</td>
<td>Hot-Key</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Paste</td>
<td>Pastes the previously copied or cut step(s). Items are pasted directly below the currently highlighted line in the Steps panel.</td>
<td>CTRL + V</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the selected step(s). To select multiple steps, hold down CTRL during selection.</td>
<td>DEL</td>
</tr>
<tr>
<td>Select All</td>
<td>Selects all steps contained in the Steps panel.</td>
<td>CTRL + A</td>
</tr>
<tr>
<td>Select Step Number</td>
<td>Opens a <em>Select Step Number</em> dialog box prompting you to enter the step number to select.</td>
<td>CTRL + G</td>
</tr>
<tr>
<td>Select Label</td>
<td>Opens a <em>Go to Label</em> dialog box prompting you to enter the label to go to.</td>
<td>CTRL + L</td>
</tr>
<tr>
<td>Disable/Enable</td>
<td>Disables/enables the selected step(s). A disabled step will not execute at runtime.</td>
<td></td>
</tr>
<tr>
<td>Edit</td>
<td>Edits the selected or highlighted step. Only a single step can be edited at a time.</td>
<td></td>
</tr>
<tr>
<td>Clear Regions</td>
<td>Clears the selected or highlighted region(s).</td>
<td></td>
</tr>
<tr>
<td>Clear All Regions</td>
<td>Clears all regions contained in the current task. Note that this will clear all existing regions whether or not they are selected or highlighted.</td>
<td></td>
</tr>
<tr>
<td>Move Steps Up</td>
<td>Moves the selected step(s) one line up.</td>
<td></td>
</tr>
<tr>
<td>Move Steps Down</td>
<td>Moves the selected step(s) one line down.</td>
<td>CTRL+ LEFT</td>
</tr>
<tr>
<td>Increase Indent</td>
<td>Increases indentation of the selected step(s).</td>
<td>CTRL+ RIGHT</td>
</tr>
<tr>
<td>Decrease Indent</td>
<td>Decreases indentation of the selected step(s).</td>
<td></td>
</tr>
<tr>
<td>Add to Watch List</td>
<td>Adds the step to the Watch List. You can view watch list items via the Watches debug pane.</td>
<td></td>
</tr>
<tr>
<td>Add to Favorites</td>
<td>Adds the selected action to your Favorites list. Favorites as well as most often used and recently used actions can be viewed from the My Actions panel.</td>
<td></td>
</tr>
<tr>
<td>Add as Snippet</td>
<td>Adds the selected action to the Snippets list.</td>
<td></td>
</tr>
</tbody>
</table>
**Task Builder Status Bar**

The area at the bottom of the Task Builder comprises an intuitive status bar that displays real-time progress of a running task through graphics and text. When a task is run from the Task Builder, the status bar graphically indicates execution state by means of a progress indicator and activity icons. Textual information such as execution status and runtime details are also displayed. The status bar is a great way to provide useful and relevant information without interrupting execution or breaking task flow, which can be very helpful and convenient for task developers, typically during testing periods.

![status bar diagram]

**Status Bar Controls**

The status bar is divided into sections, each of which displays particular information about the running task. The various elements are detailed below:

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running step/Total steps</td>
<td>Displays the current step being executed as well as the total number of steps the task contains.</td>
</tr>
<tr>
<td>Execution status</td>
<td>Displays the status of the currently running task. For example:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Running</strong> - Task is currently in a running state.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Completed</strong> - Task completed.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Error</strong> - Task generated an error.</td>
</tr>
<tr>
<td>Progress Bar</td>
<td>Displays total progress of the current step being executed.</td>
</tr>
<tr>
<td>Running step action icon</td>
<td>Displays the activity icon associated with the step currently running.</td>
</tr>
<tr>
<td>Step execution details</td>
<td>Displays information about the activity that’s currently running. This section updates step by step information about the task during execution.</td>
</tr>
<tr>
<td>Toggle My Actions view</td>
<td>Click to hide/show My Actions view.</td>
</tr>
<tr>
<td>Toggle Actions view</td>
<td>Click to hide/show Actions view.</td>
</tr>
<tr>
<td>Control</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Toggle Debug</td>
<td>Click to hide/show Debug panel.</td>
</tr>
<tr>
<td>panel</td>
<td></td>
</tr>
</tbody>
</table>
| Zoom slider   | The zoom slider allows you to increase or decrease the size of objects that | The zoom slider allows you to increase or decrease the size of objects that appear in the Steps panel. The zoom value is displayed by percentage smaller/bigger than original size. This feature can be used in the following ways:
- Move the slider left or right until the desired size is reached.
- Click + or - to zoom in or out in increments.
- Right-click the displayed percentage value and select from a drop-down list of predefined values.
**NOTE:** To revert back to the default percentage value (100%), left-click the current value that appears. |
| Zoom level    | The zoom level percentage. To revert back to the default percentage value    | The zoom level percentage. To revert back to the default percentage value (100%), left-click the current value that appears here. |
|               | (100%), left-click the current value that appears here.                     |                                                                             |

**Debug Panel**

To ensure that newly created tasks are functioning properly before they are put into production, the Task Builder’s Debug panel contains a variety of testing and debugging features that enables verbose, real time information to be displayed during task execution. This allows developers to easily monitor and examine many aspects of a task as it is being constructed. With the use of the Debug panel, developers can perform a series of “test runs” within the Task Builder and view a variety of debugging and diagnostic features such as output data, variable values and breakpoints, in order to determine the cause of a problematic task.

The Debug panel contains nine debug views separated by tabs (the active view’s tab is always colored blue). Here, you can select the Variables tab to easily view variables and datasets that are contained in the task as well as their initial and current values, or click the Output tab to view step by step information about the task during execution. Each debug view supplies a context menu containing relevant options(commands) for that particular view.
Debug Panel Views

The available debug tools are listed below in alphabetical order. For more details regarding a particular debugging tool, click the associated link.

<table>
<thead>
<tr>
<th>Debug Panel View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug Panel - Attachments</td>
<td>Lists all the files that are attached to the current task.</td>
</tr>
<tr>
<td>Debug Panel - Breakpoints</td>
<td>Lists all breakpoints contained in the task and their corresponding steps.</td>
</tr>
<tr>
<td>Debug Panel - Comments</td>
<td>Displays each comment in the task along with its step number.</td>
</tr>
<tr>
<td>Debug Panel - Labels</td>
<td>Lists all labels in the task, including the label name and corresponding step number.</td>
</tr>
<tr>
<td>Debug Panel - Output</td>
<td>Shows detailed step by step information about the task during execution.</td>
</tr>
<tr>
<td>Debug Panel - Regions</td>
<td>Displays the list of regions contained in the task, including region name and corresponding step number.</td>
</tr>
<tr>
<td>Debug Panel - Stack</td>
<td>Displays the depth of the currently running task. When a sub-task is running, its immediate parent is listed below it.</td>
</tr>
<tr>
<td>Debug Panel - Variables</td>
<td>Lists variables and datasets that are contained in the task as well as their initial and current values.</td>
</tr>
<tr>
<td>Debug Panel - Watches</td>
<td>Shows all currently set watches and displays the evaluated values for the watched variables and expressions as the task runs.</td>
</tr>
</tbody>
</table>

Debug Panel - Attachments

Attachments are files of any type that are wrapped and accessible from an task. These files can be easily exported or deployed along with their associated task allowing them to be more transportable and manageable. At runtime, attachments are automatically “unwrapped” to a local directory where they can be accessed by the task. The Attachments debug panel displays any files that are currently attached to the task. It also allows you to add a new attachment to the task as well as modify or remove existing attachments.
Parameters

The Attachments debug panel contains the following column headers.

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the attachment.</td>
</tr>
<tr>
<td>Attachment</td>
<td>The filename (including extension) of the attachment.</td>
</tr>
<tr>
<td>Attached On</td>
<td>The date and time when the file was initially attached to the task.</td>
</tr>
<tr>
<td>Description</td>
<td>A user defined description of the attachment (this property is optional, therefore, may appear blank).</td>
</tr>
</tbody>
</table>

Context Menu

Right-clicking anywhere inside the Attachments panel displays a context menu with the following items.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add...</td>
<td>Adds a new attachment to the task.</td>
</tr>
<tr>
<td>Properties</td>
<td>Opens the Attachment dialog box to display the properties associated to the selected attachment.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Refreshes the Attachments list</td>
</tr>
</tbody>
</table>
| Remove    | Removes the selected attachment(s). To select more than one attachment, hold down CTRL during selection.  
**NOTE:** When you remove an attachment, be sure to remove all references to the attachment from the task, otherwise the task may fail to execute properly. |
| Remove All| Removes all attachments that exist in a task.  
**NOTE:** When you remove an attachment, be sure to remove all references to the attachment from the task, otherwise the task may fail to execute properly. |
To attach a file to a task

1. Right-click anywhere inside the Attachments debug panel and select Add from the context menu that appears. This opens a dialog titled Attachment (shown below).
2. Enter the following information:
   - Name - The name of the attachment (this value is what you will use as a placeholder for the file).
   - Attachment file - The full path and filename of the attachment file. Use the folder icon to browse to the file you wish to attach.
   - Description - An optional description of the attachment.
3. Upon completion, click OK. The new attachment and its properties are displayed in the Attachments Debug panel.

**NOTE:** Multiple files of any type and size can be attached to a task, however, be aware that the size of the task file expands with the size and number of attachments, which may affect performance. Multiple or large sized attachments used in a single task may slow down task execution.

To remove one or more existing attachments

1. From the Attachments debug panel, select existing attachment(s) you wish to remove. To select multiple attachments, hold down CTRL during selection.
2. Right-click the attachment(s) and select Remove from the context menu. To remove all existing attachments, select Remove All.

**NOTE:** After removing an attachment, make sure to remove all references to it in any step parameter to ensure that no attachment related error occurs during runtime.

To modify an existing attachment

1. From the Attachments debug panel, right-click the attachment you wish to modify and select Properties from the context menu. This opens the Attachment dialog box.
2. Make the desired adjustments.
3. Upon completion, click OK to save changes and close the dialog box.

**NOTE:** If modifications to the original attachment name is made, make sure to update any step parameters referencing the original name to reflect the new name to ensure that no attachment related error occurs during runtime.
Debug Panel - Breakpoints

Breakpoints provide a means of pausing execution at a specified step in order for developers to inspect certain aspects of the task. In the Steps panel, breakpoints are designated by a red, circled icon located on the left side of the step number. The Breakpoints debug panel is used for viewing and managing breakpoints. It lists the step number and textual description of any step that contains a breakpoint. Using this debug panel, you can easily jump to any breakpoint step, remove a specific breakpoint or remove all breakpoints.

<table>
<thead>
<tr>
<th>Step</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Loop from &quot;1&quot; to &quot;3&quot;. Populate variable &quot;counter1&quot; with index.</td>
</tr>
<tr>
<td>6</td>
<td>Loop from &quot;1&quot; to &quot;3&quot;. Populate variable &quot;counter2&quot; with index.</td>
</tr>
</tbody>
</table>

Parameters

The Breakpoints debug panel includes the following fields (columns).

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>The step number where the breakpoint resides.</td>
</tr>
<tr>
<td>Text</td>
<td>The textual description of the action where the breakpoint resides.</td>
</tr>
</tbody>
</table>

Context Menu

Right-clicking inside the Breakpoints panel opens a context menu with the following items:

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump To</td>
<td>Jumps to the particular step in the Steps panel where a breakpoint resides.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected breakpoint(s). To select multiple breakpoints, hold down CTRL during selection.</td>
</tr>
<tr>
<td>Remove All</td>
<td>Removes all breakpoints contained in the task.</td>
</tr>
</tbody>
</table>

Debug Panel - Comments

The Comments debug panel reveals a list of comments included in the task along with the step number in which they are located. You can use this panel to quickly jump to any step that includes a comment. You can also use it to add, modify, or remove one or more comments.
Parameters

The Comments pane includes the following fields (columns).

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>The step number where the comment resides.</td>
</tr>
<tr>
<td>Comment</td>
<td>The text description of the comment.</td>
</tr>
</tbody>
</table>

Context menu

Right-clicking a specific comment or an empty region inside the Comments panel opens a context menu with the following items.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds a comment to the task. To insert a comment at a specific step, first click the desired step in the Steps panel. The comment is added directly above the selected step. If no step is selected, the comment is inserted at the end (last step) of the task.</td>
</tr>
<tr>
<td>Properties</td>
<td>Opens the selected comment’s dialog box allowing you to view or modify the comment. This item is active only if a comment is selected.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected comment(s). To select multiple comments, hold down CTRL during selection. This item is active only if a comment is selected.</td>
</tr>
<tr>
<td>Jump To</td>
<td>Jumps to the step where the selected comment is located. This item is active only if a comment is selected.</td>
</tr>
</tbody>
</table>

Debug Panel - Labels

A Label can be used as a “bookmark” in a task. Labels can be placed at specific steps in a task to mark them as reference points for Goto activities. Labels can be used in step level error handling by setting up a Goto action to be executed if a step error occurs. The Labels debug panel displays labels that are contained in a task, including the name of the label and corresponding step number. This debug tool also allows you to create labels to add to the task. More on Label action
Parameters

The Labels debug panel includes the following fields (columns).

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>The step number where the label resides.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the label (this can be a brief description of the label).</td>
</tr>
</tbody>
</table>

Context Menu

Right-clicking anywhere inside the Labels panel opens a context menu with the following items.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds a label to the task. To add a label to a specific step, highlight the step from the Steps panel, then right-click inside the <strong>Labels</strong> tab and select <strong>Add</strong>.</td>
</tr>
<tr>
<td>Properties</td>
<td>Opens the properties dialog box associated to the label in order to modify its settings. This item is active only when a right-click is performed on a label.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected label(s). To select multiple labels, hold down CTRL during selection. This item is active only when a right-click is performed on a label.</td>
</tr>
<tr>
<td>Jump To</td>
<td>Jumps to the step in the Steps panel where the specified label is located. This item is active only when a right-click is performed on a label.</td>
</tr>
<tr>
<td>Stay On Top</td>
<td>If enabled, after undocking this panel, it will stay on top of all other undocked panels. This option is disabled by default.</td>
</tr>
<tr>
<td>Dockable</td>
<td>If enabled, allows the panel to be docked to its base once it is undocked. If disabled, you will not be able to re-dock the panel once it is undocked. This option is enabled by default. You can undock a panel by holding down and dragging its title bar away from the main window or double-clicking the title bar.</td>
</tr>
</tbody>
</table>
**Debug Panel - Output**

The Output debug panel displays detailed information about a running task and usually provides the best indication of why a task failed. As a task executes, information about each step is logged to the Output panel in real-time, including task step, the action being performed within the step and completion or failure status. If the task fails, extended error information is logged in red for easy detection.

The Output panel is a useful debugging tool because it can contain invaluable information about what a task is doing at any given time. Furthermore, since the data being output is specific to the action being performed, pertinent information unique to each activity’s execution may be logged to the Output panel. For example, the File System - Copy activity outputs the name of each file being copied while it is executing. You can determine if the proper files are being copied by simply viewing the Output panel.

![Output panel screenshot]

**Context Menu**

Right-clicking inside the Output panel opens a context menu that contains the following items.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Copies the selected (currently highlighted) output data to the clipboard. The contents can then be pasted into any application or window that accepts standard text.</td>
</tr>
<tr>
<td>Select All</td>
<td>Selects all data currently displayed in the Output panel.</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears all data currently displayed in the Output panel.</td>
</tr>
<tr>
<td>Save As...</td>
<td>Saves output data as a (.log) file.</td>
</tr>
</tbody>
</table>

**Managing Output Data**

The contents of the Output panel can be handled in several different ways. New output can be appended to the existing data in the Output window, or the window can be cleared before additional data is entered. Output data can be copied to the clipboard, and the contents can then be pasted into any application or window that accepts standard text. You can also save the contents of the Output tab to a standard text file.

**To copy the contents of the Output panel**

1. Right-click anywhere in the **Output** panel and select **Copy**.
2. Open the application of your choice (usually a text file) and use the **Paste** function.
To clear the contents of the Output window

- Right-click anywhere in the Output window and select **Clear**.

  OR

- On the Task Builder Ribbon, click the **Application** button and navigate to **Editor Options**. Then click the **Debugger** tab and ensure **Clear Output window on run** is selected. This will clear the Output window at the start of running any task. This option is enabled by default.

To save output data to a text file

1. Right-click anywhere in the Output window and select **Save As...**
2. Select a file location (usually a text file), enter a file name, and click **Save**. The contents are saved as a standard Windows text file.

**Debug Panel - Regions**

A region is a named section of task steps that can be collapsed as required in order to hide their contents. The use of regions can effectively group a task into sections and can be useful for organizing lengthy tasks and making complex tasks more comprehensible. You can easily expand a region at a click of a button to access steps that you need to work on and collapse the region upon completion. The Regions debug panel lists existing regions within a task, including the name of the region, its starting step and ending step. It also allows you to jump to a particular region, clear a specific region or clear all regions that exist in the task.

<table>
<thead>
<tr>
<th>Region</th>
<th>Starting Step</th>
<th>Ending Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Variables</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Parent Loop</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Child Loop</td>
<td>6</td>
<td>19</td>
</tr>
</tbody>
</table>

**Parameters**

The Regions debug panel generates information divided into the following columns:

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region</td>
<td>The name of the selected region.</td>
</tr>
<tr>
<td>Starting Step</td>
<td>The step where the selected region starts.</td>
</tr>
<tr>
<td>Ending Step</td>
<td>The step where the selected region ends.</td>
</tr>
</tbody>
</table>
Context Menu

Right-click anywhere inside the **Regions** pane to view the following properties.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump To</td>
<td>Jumps to the step where the selected region is located.</td>
</tr>
<tr>
<td>Clear Region</td>
<td>Removes the selected region(s). To select multiple regions, hold down CTRL during selection.</td>
</tr>
<tr>
<td>Clear All Regions</td>
<td>Removes all regions.</td>
</tr>
</tbody>
</table>

Debug Panel - Stack

A stack is a dynamic data structure that stores information about the active sub-tasks contained in a task. The Stack debug panel can be used to display execution depth of the current task and allows you to keep track of the point at which each active sub-task should return control to the main task when it finishes executing.

Menu Items

Right-click inside the Stack pane to view the following menu items.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stay On Top</td>
<td>If enabled, after undocking this panel, it will stay on top of all other undocked panels. This option is disabled by default.</td>
</tr>
<tr>
<td>Dockable</td>
<td>If enabled, allows the panel to be docked to its base once it is undocked. If disabled, you will not be able to re-dock the panel once it is undocked. This option is enabled by default. You can undock a panel by holding down and dragging its title bar away from the main window or double-clicking the title bar.</td>
</tr>
</tbody>
</table>

Debug Panel - Variables

The Variables debug panel is ideal in debugging objects that may contain dynamic data, such as variables, arrays and datasets. When a task runs, this panel generates real-time information about such objects including their initial and current value. You can examine content without inserting additional steps to output the values. Similarly, you can insert breakpoints at certain points in your task to halt execution and view this panel to determine if the current values have been properly set or modified.
Task Builder (AWE Interface)

Parameters

The Variables panel generates information divided into seven columns:

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the selected variable, array or dataset.</td>
</tr>
<tr>
<td>Type</td>
<td>The type of data being displayed. This can be either a variable, array, dataset or field (associated to a dataset).</td>
</tr>
<tr>
<td>Initial Value</td>
<td>Before the task is run for the first time in a Task Builder session, this field contains the initial value of the variable as set in the variable's / array's properties. If no value was set, this field is blank. An array or dataset can contain multiple rows and columns. During runtime, this section will list the initial value of the rows and columns associated to each array/dataset.</td>
</tr>
<tr>
<td>Current Value</td>
<td>This field reflects the current value of the variable, array or dataset during task execution. <strong>NOTE:</strong> If Real-Time Variable Insight is enabled in the Debugger Preferences, the value is updated after each step executes. If Real-Time Variable Insight is not enabled, the variable Current Value field is updated only when the task is paused, when a breakpoint is encountered or when the task ends.</td>
</tr>
<tr>
<td>Description</td>
<td>A description of the variable or array as set in the associated Create Variable or Create Array properties dialog. The description does not affect variable/array performance at runtime.</td>
</tr>
<tr>
<td>Access</td>
<td>Whether the object is public or private. Controls how variables are accessed externally.</td>
</tr>
</tbody>
</table>
| Scope         | Specifies whether the variable's scope is Local or Task:  
  - **Local** - Specifies that the variable is local to the current context or scope. Usually, this means the procedure or function you are currently executing.  
  - **Task** - Specifies that the variable is a task variable which is considered global to the entire task. |

Context Menu

Right-click anywhere inside the Variables panel to view the following menu items.
### Item Description

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>Opens the properties dialog of the selected object for viewing or editing purposes.</td>
</tr>
<tr>
<td>Inspect</td>
<td>Opens a dialog box allowing you to further inspect the selected object.</td>
</tr>
<tr>
<td>Jump to</td>
<td>Jumps to the step number in the Steps panel associated to the object.</td>
</tr>
<tr>
<td>Sort</td>
<td>Sorts the list of objects in the following order:</td>
</tr>
<tr>
<td></td>
<td>- Declaration (default) - Sorts by order of declaration.</td>
</tr>
<tr>
<td></td>
<td>- Ascending - Sorts in ascending alphabetical order.</td>
</tr>
<tr>
<td></td>
<td>- Descending - Sorts in descending alphabetical order.</td>
</tr>
<tr>
<td>Add Variable</td>
<td>Adds a variable to the current task (identical to selecting Create Variable activity from the Actions panel).</td>
</tr>
<tr>
<td>Add Array</td>
<td>Adds an array to the current task (identical to selecting Create Array activity from the Actions panel).</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the selected object.</td>
</tr>
<tr>
<td>Delete Locals</td>
<td>Deletes all local variables.</td>
</tr>
<tr>
<td>Delete All</td>
<td>Deletes all objects listed in the Variables panel.</td>
</tr>
</tbody>
</table>

### Managing Variables

Variables can be added, edited, or deleted directly from the Variables debug panel. When you edit variables in this way, Create Variable steps are added, changed, or deleted in the task.

**To edit an existing variable**

1. In the Variables Debug panel, double-click the variable or right-click the variable and select Properties from the context menu that appears.
2. Edit the desired properties in the dialog box and click OK when finished.

**To add a variable/Array**

1. Right-click anywhere inside the Variables debug panel and select Add Variable or Add Array from the context menu that appears.
2. Enter properties for the variable/array in the properties dialog box and click OK when finished.

**To delete a variable/Array**

- In the Variables debug panel, right-click the variable or array and select Delete or select the variable/array then press the Delete key on your keyboard.
- To delete only local variables, right-click and select Delete Locals.
- To delete all variables/arrays, right-click and select Delete All.
Debug Panel - Watches

The Watches debug panel can store several variables and expressions that you want to view over the course of the debugging session. You can set a watch on a variable used in the task or you can enter an expression to create a more complex watch. You can use the Watches debug view to add, remove, or change watches. After each step executes, any variables and expressions listed are re-evaluated and updated with their current values. Watches go beyond the Variable window by providing an intuitive means of testing variable or task states while a task is executing.

Parameters

The Watches panel includes the following fields (columns).

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>The syntax of the watch. This can be simply a variable name, or a complex expression.</td>
</tr>
<tr>
<td>Value</td>
<td>The result of the watch expression. This is updated with each step execution while the task is running.</td>
</tr>
</tbody>
</table>

Context Menu

Right-click inside the Watches panel to view the following menu items.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change</td>
<td>Renames the selected variable or modifies the selected expression.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected item. To select multiple items, hold down CTRL during selection.</td>
</tr>
<tr>
<td>Remove All</td>
<td>Removes all items listed.</td>
</tr>
</tbody>
</table>
Testing Tasks Using Run Options

Task execution may happen so rapidly that it is difficult for a user to determine the exact cause of a problematic task. In such cases, the Task Builder encompasses various Run options (displayed below) to handle this issue. The Run Selected option runs only the selected steps, which is a useful way to ignore execution of steps that non-problematic. The Run From Here option runs the task from the selected step onward. The Step option suspends task execution on a step by step basis, allowing you to examine what the task is doing during execution of each step as well as view important runtime information populated within the various Debug panel views.

![Run Options](image)

NOTE: When suspended, the task is still running but paused between actions. variables and datasets retain their values, and the Task Builder's Output panel displays information about the step that is currently running.

Available Run options

To run the entire task

- Click the main Run All button or press F5. This will run all of the steps of the task sequentially from beginning to end (or up to a Breakpoint if one is added).

To run one or more selected steps

1. Select the step(s) you wish to run. To select more than one step, hold down CTRL during selection.

2. Select Run Selected or press CTRL+F5. This will run the specified steps in sequence from the lowest to highest numbered step.

To run the task from a specific step

1. Select the step that should start the execution process.

2. Select Run From Here or press SHIFT+CTRL+F5. The steps will run sequentially starting from the specified step.
To run the task step by step

1. Click the Step button. The first (or selected) step of the task runs. At this time you can:
   - Examine the contents of your variables by looking at the Variable View of the Debug Window
   - Examine the states of any watches you have set, using the Watches View of the Debug Window (see below for more information on watches)
   - Observe what the previous step may (or may not) have done by reading the contents of the Output View.

2. Click Step to execute the next line.

3. Continue clicking Step to execute the next line until you reach the end of the task or until you have determined the issue.

4. You can also run the rest of the task at regular speed from this point forward by clicking the Run from here button. Click the Reset button at any time to stop the step process.

More on "Stepping"

"Stepping" is the process of executing a task one step at a time. When clicking the Run button, task execution may happen so quickly that it’s too complicated to determine what is causing the problem and where that problem takes place. The Step button is ideal for such a situation because clicking this button will execute one step, then wait, providing a chance for the developer to examine what the previous step may (or may not) have done or observe the contents generated in the Debug window in order to better diagnose the task. The Step button can be accessed via the keyboard by hitting the F9 key.

NOTE: Most Task Builder commands and controls contain a corresponding shortcut key (or hot-key). To determine the appropriate shortcut for a control, simply hover the mouse over it.

Control running tasks

You can manually pause or stop a currently running task at any time during its execution.

To suspend a running task

- Click the Pause button or press F6. Upon doing so, the task delays execution at the current line. Click Pause or press F6 again to resume execution.

To stop a running task

- Click the Stop button or press CTRL + F2. The task immediately stops at the current line and resets itself.

NOTE: The Pause and Stop buttons are usually disabled and are active only during task execution.
Debugging Tools & Techniques

An intuitive drag-and-drop Task Builder interface allows users to easily construct the steps that a task should perform during runtime. While this solution may eliminate construction time, maintenance, effort and common errors, it does not make tasks immune from inevitable "bugs" which may still occur, causing them to behave in unintended manners or produce incorrect or unexpected results. That is why the Task Builder encompasses a wide variety of troubleshooting and debugging features as well, to aid users in properly examining tasks, detecting anomalies and determining discrepancies. This article discusses a number of Task Builder debugging tools and common techniques that can facilitate in troubleshooting and resolving problematic tasks.

Examining Debug Pane Data

The Task Builder’s Debug Pane generates an assortment of real time information about a running task, allowing you to easily monitor and examine many aspects of your project as it is being constructed. For example, the Output Pane displays verbose, real-time information about the task during its execution. For more information, see Task Builder Debug Window.

Stepping & other Run Options

When clicking the Run button, task execution may happen so quickly that it may be too difficult to pin-point exactly which task step is causing the problem and why. The main Run button’s default behavior is to run all steps of the task sequentially, from start to finish. Alternatively, clicking the down arrow located below the Run button supplies other run options (as illustrated below). The Run Selected button runs only selected steps, the Run From Here button runs the task from a specified step and the Step button runs the task one step at a time. For more information, see Testing Tasks Using Run Options.

Setting Breakpoints

Breakpoints are very useful for larger tasks where stepping from the beginning of the task to a point of interest may be tedious and time consuming. Instead, you can tell the task to pause at a specific step by creating a breakpoint at that step. When the breakpoint is encountered, will pause task execution. From here, you can follow the recommendations above, including stepping from this point forward by clicking the Step button, or continuing normal task execution by clicking the Run button. See Breakpoints for more information.

Setting a Watch

Watches provide a powerful means for watching the state of variables and expressions in your task. In larger tasks, where Script steps, multiple Set Variable steps, or other more complex task steps are involved, watches are invaluable for seeing where and how variable values change. A watch can be simply a variable name to be examined, or it can be a complex expression that is reevaluated as the step executes. See Watches for more information.
The **Watches Debug Pane** displays a list of all watches and updates the value of each watch as the task runs. Watches can be added, removed, or modified using the Watches Tool. Watches are primarily used for debugging, thus, they are ignored when a task is executed as a result of a trigger or executed manually from the Task Administrator.

### Minimizing Task Builder on run

For tasks that make use of Interactivity Actions, or otherwise interact with various windows, set an option to minimize Task Builder as the task runs. Task Builder is minimized as the task runs, and restored when the task ends or encounters a breakpoint. This allows Interactivity and Window actions to work unobstructed.

**To minimize Task Builder on run**

1. On the Task Builder **Tools** menu, click **Editor Options**.
2. Click the **Debugger** tab, select **Minimize on run** and click **OK**.

### Breakpoints

When a task runs, each step’s total execution time may only amount to 1 or 2 seconds, making it difficult to examine certain steps that you suspect are not running properly. You can try stepping through each step, however, this can be tedious and time consuming, especially with long, multi-step tasks. Breakpoints may be the most ideal solution in this situation. They enable you to suspend task execution where and when you need to. More generally, a breakpoint is a means of acquiring knowledge about a task during its execution. During the interruption, you can inspect the test environment to find out whether the task is functioning as expected.

A breakpoint consists of one or more conditions that determine when a task’s execution should be interrupted. Breakpoints can be set at any step and can be used as many times as needed in a single task. They take effect only when a task is run from Task Builder and are ignored when the task is triggered or executed outside of the Task Builder. The **Breakpoints Debug Panel** can be used to examine and manage breakpoints within a given task. Using breakpoints can speed up the debugging process enormously. Without this feature, it would be very difficult to debug a large task. Breakpoints are indicated by a red, circled icon located on the left side of the step number in the Steps panel (as shown below).
Behavior

When a running task reaches a breakpoint step, the Task Builder pauses the task, at which point, you can verify proper completion of a particular step or examine the data produced by the assortment of debugging tools located in the Debug panel. For example, you can view step details generated by the Output panel or view the Variables panel to determine the current value of a certain variable or dataset. In addition, you can manage existing breakpoints or easily jump to a breakpoint step via the Breakpoints panel.

Depending on what you determine from the data displayed, you can then select Run to continue the task, Step to continue the task step by step, or Stop to halt task execution in order to edit the steps. When paused due to a breakpoint, the task is still running but suspended between actions. Variables and datasets retain their values and the Output view continues to display detailed information about the task that is currently running.

NOTE: Breakpoint symbols are positioned in the same location as bookmark symbols on the Steps panel. If a specific step comprises both, the breakpoint symbol takes precedence over the bookmark symbol, thus, you will see only the icon indicating the breakpoint and not the bookmark icon.

To set a single Breakpoint:

- From the Steps panel, go to the step where you want to add a breakpoint and left-click just to the left of the step number.

To set multiple breakpoints simultaneously:

1. From the Steps panel, select the steps in which to place a breakpoint. To select more than one step, hold down CTRL during selection.
2. From the Ribbon control, go to the Home tab and click Breakpoint from the Step command group.
   OR
   Press F8 on your keyboard.

Comments

A Comment is used to embed user readable annotations in the steps of a task, usually with the purpose of making the task easier to understand. Comments can be used to summarize the task, explain the intent of a specific step, document instructions, enter notes, reminders or other important information. Comments are ignored by during runtime. For debugging convenience, the Comments Debug Panel displays each comment embedded in a task and their associated step number. For easy detection, the default color for an added comment is green, however, this color can be modified via the Task Builder Color Options.

To add a comment

1. From the Steps panel, highlight the step in which to place the comment.
2. From the Ribbon’s **Insert** command group, select **More -> Add Comment** or press CTRL + ALT + C. A dialog box titled **Comment** will appear as shown below.

![Comment Dialog Box](image)

3. Enter the desired comment in the **Comment Text** field.

4. To automatically add this comment at the end of the task, enable the **Add to bottom of task** option. Otherwise, the comment will be placed in the step currently highlighted.

5. Click **OK** when finished. The new comment is then displayed in the Steps pane (as shown below circled in red).

![Steps Pane with Comment](image)

**NOTE:** Comments can be added from the Debug panel as well by clicking the **Comments** tab, right-clicking inside the panel and selecting **Add**... from the context menu.
**Regions**

A region is a uniquely named section of a task that can be collapsed in order to hide its contents. It can be used to separate segments of a task, thus, making it less cluttered and more manageable. Regions have no effect on task execution. They are mainly used to modify visual aspects of a task during construction. When a task runs from Task Builder, regions expand automatically so the developer can view each step as it executes. Upon completion, regions are re-collapsed to their former state.

Regions can be created to categorize blocks of steps in a task based on common functionality or actions. When a region is collapsed, their contents are hidden from view and they appear as a single line in the task. A unique name can be entered for a region as a way to ascertain its contents when it is collapsed. If a particular step needs to be viewed or modified, simply expand the region that contains that step while the rest of the regions stay collapsed and hidden from view. supports nested regions (region within a region) to further expand their capabilities.

**Creating and Populating Regions**

You can create an empty region and later add steps to it or create one pre-populated with selected steps. When creating a new region, its default name is always ‘New Region.’ It is important that you provide a unique name for newly created regions in order to avoid confusion. When you are finished constructing your task, you can opt to remove all existing regions with the **Clear Regions** command. All regions are ignored when the task is triggered or ran manually outside the Task Builder.

**To create a region**

1. Highlight the step(s) in the **Steps panel** you wish to place in a region. To select multiple steps, hold down CTRL during selection (selected steps must be sequential).
2. Click the **Create Region** button located in the ribbon’s **Step** command group or right-click and select **Region** -> **Create Region**. A new region is added to the Steps panel populated with the step(s) you initially selected.
3. Left-click the newly created region to rename it.
To add a new action/activity to an existing region

1. Expand the region you wish to add an action to by double-clicking the region name or clicking the expand symbol to the left of the region name.
2. Drag the desired action/activity from the Available Actions pane onto a selected part of the region. A red line will appear signifying the section of the region in which the action will be placed.

To add existing steps to a region

1. Expand the region you wish to add steps to by double-clicking the region name or clicking the expand symbol to the left of the region name.
2. Select one or more existing steps from the Steps pane (selected steps must be sequential in order).
3. Drag the step(s) to the desired part of the region. A red line will appear signifying the section of the region in which the step(s) will be placed.

To create a nested region (region within a region)

1. Expand the region you wish to add a sub-region to.
2. Highlight the steps within that region that you would like to place in the sub-region.
3. Click the Create Region button from the Task Builder Ribbon’s Step command group or right-click and select Create Region. A new sub-region is added inside the parent region.
4. Click the newly created sub-region to rename it.

Managing Regions

To expand all regions simultaneously

1. Select Expand All Regions from the Ribbon’s Step command group.
   Or
2. Press SHIFT + E

To collapse all regions simultaneously

1. Select Collapse All Regions from the Ribbon’s Step command group.
   Or
2. Press SHIFT + C

To clear one or more regions

1. Select the region you wish to remove. To select multiple regions, hold down CTRL during selection.
2. Right-click and select Clear Regions.
3. To clear all existing regions, right-click and select Clear All Regions.
**Watches**

A Watch is a debugging feature used for examining the state of variables and other expressions within a running task. A watch can simply be the name of a variable. For example, to watch the value of a variable named `TotalCount`, add a watch that contains simply `%TotalCount%`. Embedded expressions can also be used within a watch, using the same syntax as a standard embedded expression. For example, to watch the length of an variable named `myName`, add a watch `%Len (myName) %`.

The **Watches Debug Panel** allows you to add, remove or modify items to watch. It displays the list of added watches and updates their value as the task runs. This way, watches can be re-evaluated after each task step is executed. Watches are mainly used for debugging, therefore, they are ignored when a task is executed as a result of a trigger or executed manually from the Task Administrator.

**Using Watches**

The Watches debug panel is a place where you can enter variable names and expressions that you want to watch during a debugging session.

**To watch a variable**

- Enter the variable to watch in one of the following ways:
  - In the **Steps panel** of the Task Builder, right-click the **Create variable** step associated with the variable you want to watch and select **Add to Watchlist**.
  - In the Watches debug panel, type the name of the variable in the provided entry box and click **Watch**.

**To watch an expression**

1. In the Watches debug panel, enter the expression to watch in one of the following ways:
   - Type the expression into the entry box. For example, to evaluate the length of an variable named `myName`, enter `Len (myName)`.
   - OR
   - Click the **Gear** button to the right of the Watches entry box. Then use the **Expression Builder** to create an expression to watch. Click **Insert** when you are finished.

2. Click **Watch** to add to the watch list.

3. The expression and its current value (if applicable) is added to the **Watches** debug panel.

**To modify a watch**

1. In the **Watches** debug panel, right-click the watch and select **Change**.

2. Type the new variable or expression for the watch and press **Enter**.
To remove a watch

- Select the watch to be removed and press the **Delete** key, or select **Remove** from the right-click menu.

- To remove all watches, select **Remove All** from the right-click menu.

**NOTE:** Using watches can affect runtime performance in the debugger on slower machines. If debugging speed becomes a factor, try removing watches that are no longer needed. This does not apply to runtime performance outside the debugger, as watches are ignored when the task is executed outside Task Builder.

**Error Handling**

A three-tiered approach to error handling provides a variety of methods in which to troubleshoot problematic tasks as well as supply alternative techniques to allow a task to recover gracefully from unexpected errors. Error handling can be set via the step, task and system level. The flow of error handling takes precedence from step to task to system. For instance, if an error occurs during task execution, first adheres to any exception handling parameters set for the step that caused the error (if any), followed by error handling set for the task level, and conclusively, any system wide error handling parameters.

**Step Level Error Handling**

Step level error handling handles task exceptions on a per step basis. Normally, any failure of a step within a task causes a step error. What causes a step to fail is, however, dependent on the step and its parameters. These parameters can be adjusted under the **Error Causes** tab of any step (as shown below). For additional details, see **Error Causes**.
If a step fails, the user can select from a number of actions to be carried out. These actions are located in the On Error tab of any specific step (as shown below). For additional details, see On Error.
Step level error handling takes precedence over task and global level error handling. When a step level error occurs, a task can alert a user, request user intervention, or with proper exception handling, it can recover on its own. For further information, see Step Error Handling.

**Task Level Error Handling**

As part of "tri-level" error handling, errors may be handled on a per task basis. This is ideal if the user wants to perform specific exception handling procedures for a managed task. A task level error occurs when a step within a task fails and generates an error. This causes the task to notify the Task Service of the failure, which allows the task’s error level handling events to occur. Task level error handling can be set individually for any task and can be configured in the Task Administrator by accessing a task’s Managed Task Properties and selecting On Error. Doing so will open the On Task Error page (as shown below). For further details, see On Error.
In most cases, error information is written to a log file so the user can easily track down illusive errors or identify obscure program behavior.

**System Level Error Handling**

AWE can handle errors globally for all tasks that run on a system. If any system level error handling parameters are enabled, will act on ALL managed tasks that fail with an error. The settings for System Error Handling can be configured in the Task Administrator by navigating to **Options -> Default Properties -> On Task Error**. Doing so will open the global On Task Error page (as shown below). For more details, see **On Task Error**.
Setting Task Builder Debug Toolbar Options

When you test a task that includes interaction with a window, Task Builder minimizes to allow the steps to proceed. To aid in testing the task, the Debug toolbar displays in a separate window while the task runs. A set of shortcut keys can be used when testing a task. These are listed in the Debug toolbar options page, and are active only while a task is being run/tested in Task Builder. You can change the shortcut keys to any desired key combinations not already in use, and you can change the position of the Debug toolbar on the screen.

To set Debug toolbar options

1. Do one of the following:
   - On Task Builder Ribbon, click the Application button located on the top left corner of the Task Builder window and click the Preferences button. From the Preferences window that appears, click the Debug Toolbar tab (as shown below).
   OR
   - On the Tools menu, click Editor Options. From the Preferences window that appears, click the Debug Toolbar tab (as shown below).

2. To hide the debug toolbar, clear the Show toolbar at check box.

3. To change the display position of the toolbar, ensure the check box is selected and select a position from the drop-down menu below.
4. To control whether the toolbar remains on top, select or clear the **Always on top** check box.

5. To change any of the toolbar shortcut keys, select the desired toolbar hot-key and press the desired keys.

6. Click **OK** when finished in order to save changes.

### Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| Show toolbar at           | If enabled, specifies the location of the screen in which the debug toolbar (shown below) should appear when a task is run from the Task Builder. The toolbar is a convenient debugging tool that allows you to control (e.g., stop, pause, step through) a task during its execution. If disabled, the toolbar will be hidden from view. The available options are:  
  • upper left
  • upper right
  • lower left
  • lower right                                                               |
| Always on top             | If enabled, the debug toolbar will always be on top (or in front) of other windows on the desktop during task execution. If disabled, other windows may steal focus, thus, hiding the toolbar from view. |
| Toolbar Hot-keys          | Allows selection of hot-keys to run, stop, pause or step through a task. The available options (along with their default hot-key parameters) are:    
  • **Run** - CTRL + ALT + F5
  • **Stop** - CTRL + ALT + F2
  • **Pause** - CTRL + ALT + F9
  • **Step** - CTRL + ALT + F6                                                |

### Task Builder Options

The Task Builder contains its own set of options that can be configured to meet a user’s specific needs. They include color, icon and font settings, viewing options, debug preferences and formatting properties. Task Builder options can be accessed via **File -> Options** (circled below).
Available Options

Task Builder options are grouped into six separate categories, each encompassing specific preferences and options. Below lists the available categories, with links to the topics covering each.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Options</td>
<td>Contains options that determine the color to be used for the Task Builder interface or allows adjustments to be made to the current color scheme.</td>
</tr>
<tr>
<td>Debugger Options</td>
<td>Contains options that affect Task Builder’s behavior while running and debugging tasks.</td>
</tr>
<tr>
<td>Fonts Options</td>
<td>Contains options to modify the font name, size and style to be used for the Visual view and the AML view.</td>
</tr>
<tr>
<td>Formatting Options</td>
<td>Contains options that deal with how steps should be formatted in the Steps pane.</td>
</tr>
<tr>
<td>General Options</td>
<td>Contains options that determine certain behavioral characteristics of Task Builder. Also determines the default managed task location.</td>
</tr>
<tr>
<td>Task Builder Toolbar Options</td>
<td>Contains options that determine the look and feel of debugger toolbar.</td>
</tr>
</tbody>
</table>
Task Builder Color Options

The Task Builder can be configured to use color schemes, which are collections of colors used for various steps, columns and conditions that appear in the Steps panel. A color scheme allows different users of the Task Builder to create their own custom “look and feel” when creating, editing and debugging tasks. You can select from a list of pre-defined color schemes or create your own scheme. Color options is a subset of Task Builder Options.

![Options dialog box with color scheme settings]

Parameters

The following table describes available color parameters from left to right starting from the top row.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Scheme</td>
<td>Default</td>
<td>The name of the color scheme presently being viewed. The default color scheme is named ‘Default’. To create a new color scheme, select the desired color for each element, click Save As and enter the desired name. To delete a scheme, select the name from the drop-down list and click Delete.</td>
</tr>
<tr>
<td>Steps</td>
<td>Black</td>
<td>The color of an active, recognized step. The default color is black.</td>
</tr>
<tr>
<td>Step Number Text</td>
<td>Gray</td>
<td>The color of the step numbers. Step numbers appear in a column along the left-hand side of the Steps panel. The default color is gray.</td>
</tr>
<tr>
<td>Error</td>
<td>Red</td>
<td>The background color to use when highlighting a step that has caused an error. When debugging a task, this color will show the step that caused the task to fail. The default color is red.</td>
</tr>
</tbody>
</table>
### Task Builder (AWE Interface)

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highlight</td>
<td>Navy</td>
<td>The background color of a highlighted, or currently selected, step. This is typically a color that contrasts with the highlight text color. The default color is navy.</td>
</tr>
<tr>
<td>Step Number Column</td>
<td>238.238.238</td>
<td>The color of the step column that contains the step numbers. Step numbers appear in a column along the left-hand side of the Steps panel. The default color is RGB color code 238.238.238.</td>
</tr>
<tr>
<td>Highlight Text</td>
<td>White</td>
<td>The text color of a highlighted, or currently selected, step. This should be a color that contrasts with the highlight color. The default color is white.</td>
</tr>
<tr>
<td>Background</td>
<td>White</td>
<td>The background color of the Steps panel. The default value is white.</td>
</tr>
<tr>
<td>Breakpoint</td>
<td>Teal</td>
<td>The text color of a step that contains a breakpoint. Task execution pauses at a breakpoint step when running the task from the Task Builder. The default color is teal.</td>
</tr>
<tr>
<td>Region Text</td>
<td>Blue</td>
<td>The color of region text. The default color is blue.</td>
</tr>
<tr>
<td>Comment</td>
<td>Green</td>
<td>The color of a comment. Comments, or remarks, are ignored by at runtime, and provide a means for documenting or placing notes into your task. The default color is green.</td>
</tr>
<tr>
<td>Running</td>
<td>Green</td>
<td>The background color of the currently running step during task execution. This color allows easy tracking of the step currently being processed while debugging a task in the Task Builder. The default color is green.</td>
</tr>
<tr>
<td>Disabled</td>
<td>Gray</td>
<td>The color of a disabled step. Disabled steps are not executed at runtime. The default color is dark gray.</td>
</tr>
<tr>
<td>Breakpoint Column</td>
<td>221.221.221</td>
<td>The color of the column that contains Breakpoints. The default color is color code 221.221.221.</td>
</tr>
<tr>
<td>Block Shading</td>
<td>238.238.238</td>
<td>The color of a block shade - The default color is color code 238.238.238.</td>
</tr>
<tr>
<td>Region Background</td>
<td>238.238.238</td>
<td>The background color of a region. The default color is color code 238.238.238.</td>
</tr>
</tbody>
</table>

**To switch existing color schemes:**

1. In the **Color Scheme** parameter, click the down arrow.

2. Select the name of the scheme you want to switch to from the drop-down list that appears. The selected scheme's name appears in the **Color Scheme** parameter and automatically becomes the default scheme.
To create a new color scheme:

1. Select the desired color for each element.
2. Click Save As... and enter a name for the new color scheme in the dialog box that appears.
3. Click OK to save changes. The newly created color scheme is automatically saved as the default scheme and its name appears in the Color Scheme parameter.

To delete an existing color scheme:

1. In the Color Scheme parameter, click the down arrow.
2. Select the name of the scheme you wish to delete from the drop-down list that appears and click Delete.
3. A dialog appears verifying whether or not to delete the color scheme. Click Yes to complete the removal process.

Task Builder Debugger Options

The Task Builder includes settings that affect its behavior while running a task. In some cases you may want to change these settings to better fit your debugging needs. For instance, you can opt to minimize the Task Builder window during execution to better examine an interactive task, automatically clear the Output Debug Panel after each run, or play a custom sound when an error occurs. Debugger options is a subset of Task Builder Options.

![Debugger Options](image-url)
The following table describes available **Debugger** options.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save task on run</td>
<td>Disabled</td>
<td>If enabled, Task Builder automatically saves changes made to a task when clicking the Run button. If the task has not been saved yet, will prompt the to save before running. This parameter is disabled by default.</td>
</tr>
<tr>
<td>Minimize on run</td>
<td>Disabled</td>
<td>If enabled, the Task Builder window minimizes to the system tray when execution starts. When the task ends, the Task Builder restores itself and highlights the current step or displays <strong>Finished</strong> as the step status in the status bar.</td>
</tr>
<tr>
<td><strong>NOTE:</strong> The default behavior of the Task Builder is to stay on the desktop (does not minimize or hide itself) while a task is running. When testing tasks that include interactive actions (e.g., Send Keystrokes, Move Mouse to Object actions) or actions that interact with other windows (e.g., Maximize Window, Minimize Window actions), if set to default, the Task Builder window may interfere with the actions being performed by a task. This may result in a Send Keystrokes step, for example, failing to send keystrokes to the proper window or application. Enabling the <strong>Minimize on run</strong> option will remedy such an issue.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restore on pause and/or breakpoint</td>
<td>Enabled</td>
<td>If enabled, when task execution is manually paused (when a user clicks the Pause button) or pauses at a breakpoint, and the Task Builder window is currently minimized, it restores itself and highlights the current step. When execution continues, the Task Builder window, once again, minimizes to the system tray.</td>
</tr>
<tr>
<td>Clear output window on run</td>
<td>Enabled</td>
<td>If enabled, clears the contents of the <strong>Output debug panel</strong> before the task starts its execution. If disabled, new debugging output is appended to existing output accumulated from a previous task run.</td>
</tr>
<tr>
<td>Real-time variable insight</td>
<td>Enabled</td>
<td>The Task Builder maintains a list of variables in the current task which can be viewed in the <strong>Variables Debug Panel</strong>. At runtime, this panel also shows the current value of these variables. If this option is enabled, the panel is updated after each step is executed to show the current variable value. If disabled, the panel is updated only when the task is paused (by clicking the Pause button or when a breakpoint is encountered), while stepping through the task with the use of the <strong>Step</strong> button, and when the task ends.</td>
</tr>
<tr>
<td><strong>NOTE:</strong> Enabling this option could cause tasks to execute somewhat slower while debugging (it does not affect execution speed when not debugging, however).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Property Default Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspect entire dataset</td>
<td>Disabled</td>
<td>Datasets contain a number of properties that are not usually displayed in the Variable Window. These properties are used, for example, to view the date/time the dataset was created, the total number of rows in the dataset, etc. If enabled, this option displays these extended dataset properties in the Variables Debug Panel when the task is run.</td>
</tr>
<tr>
<td>Show initial variable value</td>
<td>Enabled</td>
<td>If enabled, causes the Variables Debug Panel to include a column for <strong>Initial Value</strong> in addition to the <strong>Current Value</strong> column. If this is disabled, only the current value is shown.</td>
</tr>
<tr>
<td>Ignore task timeout</td>
<td>Enabled</td>
<td>If enabled, the task timeout value set under the Error Causes properties of the task is ignored.</td>
</tr>
<tr>
<td>Play sound on error</td>
<td>Error.wav</td>
<td>Selects whether to play a sound when the task stops as a result of an error. Regardless of this setting, when a step fails, it is always colored using the Error color set in the <strong>Color Options</strong> (default is red). Also, task execution stops, and the error is written to the Output Debug panel. The selections are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Never</strong> - Do not play a sound when an error occurs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Use default sound (default)</strong> - Use the default system error sound, as specified in Window’s Control Panel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Use this sound</strong> - Specify a valid path and filename of a .wav file to use.</td>
</tr>
</tbody>
</table>

### Task Builder Font Options

Task Builder font options allow you to select a customized font to be used for the visual view and the AML view of the Task Builder’s Steps Panel. Font options is a subset of Task Builder Options.
Parameters

This table lists the fonts for both Visual and AML modes. To modify font preferences, click Change to open a standard windows Font dialog. Make the desired changes and click OK.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual View Name</td>
<td>Segoe UI</td>
<td>The name of the font to be used when Visual mode is selected from the View menu. An easy to read font for this view is suggested, such as Segoe UI or Tahoma, since Visual mode is meant to provide a clear, natural-language description about the task's steps.</td>
</tr>
<tr>
<td>Visual View Size</td>
<td>9</td>
<td>The Visual view font size.</td>
</tr>
<tr>
<td>Visual View Style</td>
<td>Regular</td>
<td>The Visual view font style.</td>
</tr>
<tr>
<td>AML View Name</td>
<td>Courier New</td>
<td>The font to be used when AML mode is selected from the View menu. A fixed-width font, such as Courier New, for AML view is suggested, since this type of view is typically used for fine, programmatic refinements to task steps.</td>
</tr>
<tr>
<td>AML View Size</td>
<td>10</td>
<td>The AML view font size.</td>
</tr>
<tr>
<td>Property</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>AML View Style</td>
<td>Regular</td>
<td>The AML view font style</td>
</tr>
</tbody>
</table>

**Task Builder Formatting Options**

Task Builder formatting preferences allow you to modify how activities that represent a block of steps, such as activities contained inside a Loop or If statement, are formatted in the Steps panel. You can choose to automatically add closing steps to those that require one (e.g., Loop activities should automatically be followed by End Loop activities or If activities should automatically be followed by End If activities) or you can choose to add such steps manually as you construct the task. You can also decide upon whether those blocks of steps should be indented and what the indentation value should be. Formatting options is a subset of Task Builder Options.
Parameters

The following table describes Task Builder’s available **Formatting** options.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrap step text</td>
<td>Enabled</td>
<td>If enabled, the text that appears in the Steps panel will automatically wrap itself so as not to run off the right edge of the window. This enables all text to be viewed without needing to scroll to the right. If disabled, the text that appears in the Steps panel will run off the right edge.</td>
</tr>
<tr>
<td>Step level auto-completion</td>
<td>Enabled</td>
<td>If enabled, any activity that starts a block of steps will be followed by the activity used to end the block of steps. For example, if enabled:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Any If statements added to a task will automatically be followed by an End If Statement. If disabled, you must manually add an End If step.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Any Loop actions added as a step to a task will automatically be followed by an End Loop action. If disabled, you must manually add an End Loop step.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Any Case statements added to a task will automatically be followed by an End Case Statement. If disabled, you must manually add an End Case step.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Select/Case structures will be built automatically.</td>
</tr>
<tr>
<td>Automatically enclose proceeding</td>
<td>Enabled</td>
<td>If enabled, proceeding steps are automatically added when starting a block of steps.</td>
</tr>
<tr>
<td>step when starting a block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auto-Indent</td>
<td>Enabled</td>
<td>If enabled, allows steps to be automatically indented. The default indent value can be entered in the <strong>Indent</strong> text box.</td>
</tr>
<tr>
<td>Indent</td>
<td>5</td>
<td>Specifies the indentation value. This parameter is set to 5 by default and only available if the <strong>Suggest Indent</strong> parameter is enabled.</td>
</tr>
</tbody>
</table>

**Task Builder General Options**

Task Builder contains general options that enables control over a variety of interactive aspects. You can choose whether double-clicking an action/activity from the **Actions panel** should add it to the beginning or end of the task. Also, you can dictate what task should appear the next time the Task Builder is open, whether it be a new task, all tasks from the previous session or selected task from the previous session. General options is a subset of **Task Builder Options**.
Parameters

The following table describes Task Builder’s available General options.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On startup, open</td>
<td>New task</td>
<td>Dictates the Task Builder’s startup behavior. This option is relevant only when the Task Builder is started manually from Windows Start menu or by the executable file via Program Files directory (e.g., C:\Program Files (x86)\10\AMTB.exe). The previous task is never opened when Task Builder is started for a managed task through Task Administrator.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>New task</strong> - A new task will open.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Previously opened tasks are ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Selected task from last session</strong> - Automatically re-opens the task that was last open in the Task Builder. If multiple tasks were previously open, the active/selected task is the one that will re-open.</td>
</tr>
<tr>
<td>Property</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Double-clicking adds to end of task</td>
<td>Enabled</td>
<td>If enabled, double-clicking an action/activity from the Actions panel automatically adds it to the Steps panel as the last step. If disabled, the action/activity is automatically inserted in the Steps panel above the currently selected (highlighted) step. In either case, a new step can be inserted at the desired position by simply dragging the action/activity from the Actions panel into the position in the Steps panel.</td>
</tr>
<tr>
<td>Default task location</td>
<td>C:\Users[username]\Documents\My Tasks</td>
<td>Indicates the default location that an unmanaged task should be saved to. This location can vary depending on the operating system. By default, tasks are saved to a location accessible by the user that initially installed the program. This location is different from the default managed task location. See Managed &amp; Unmanaged Tasks for more details. <strong>NOTE</strong>: If specifying a network location as the default task location, it is critical that mapped drive letters not be used. Folders should always be specified using UNC (Universal Naming Convention) paths. For example: X:\pathname should be \servername\pathname. This is because mapped drives are available only when a user logs on and this does not occur for LocalSystem services.</td>
</tr>
<tr>
<td>Include error handling descriptions</td>
<td>Only when different from default</td>
<td>Controls when the visual view in the Steps pane displays descriptive text about each step’s error handling along with descriptive text about the step itself. Every step performs default error handling if the step fails. Select from the following:</td>
</tr>
</tbody>
</table>
## Task Builder Toolbar Options

The Debug toolbar (illustrated below) is a small floating window that appears when a task is run from the Task Builder. It contains buttons to stop, pause, re-run and step through an active task to further aid you during the development stage. The Debug toolbar automatically closes upon completion of a running task or you can close it at any time by clicking “x”.

![Debugging Toolbar](image)

The Debug toolbar supports hot-keys (also known as shortcut keys) that can be used as an alternate control mechanism. You can change the default toolbar shortcut keys to any desired key combinations not already in use, and you can change the position of the Debug toolbar on the screen. In addition, you can choose to show/hide the Debug toolbar or have it always appear in the foreground or background. Toolbar options is a subset of Task Builder Options.
## Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| Show toolbar at   | Lower right | If enabled (enabled by default), specifies the location of the debug toolbar (shown below). If disabled, the toolbar will be hidden from view. The available options are:  
  - **Upper left**: Upper left corner of the screen.  
  - **Upper right**: Upper right corner of the screen.  
  - **Lower left**: Lower left corner of screen.  
  - **Lower right (default)**: Lower right corner of screen. |
| Always on Top     | Enabled     | If enabled, the toolbar will always be on top (in the foreground) in front of other windows. If disabled, the toolbar will appear in the background. This parameter is enabled by default. |
| Toolbar Hot-keys | See Description | Allows selection of hot-keys to run, stop, pause or step through a task. The available options (along with their default hot-key parameters) are:  
  - **Run**: CTRL + ALT + F5  
  - **Stop**: CTRL + ALT + F2  
  - **Pause**: CTRL + ALT + F9  
  - **Step**: CTRL + ALT + F6 |
About Creating Tasks

Tasks (or workflows) are the primary objects in AWE. A task is a series of steps designed to achieve some result. A task can be built to almost any business process. Tasks can perform file transfers, batch data processing, application testing, e-mail transmissions and much more. A task can be composed of a single step or hundreds of steps. Tasks can be started by a user or set to run automatically when triggered by a particular event.

A task is created in the EFT Enterprise administration interface by clicking New on the Advanced Workflows node. Priorities, security, error handling, and other properties for the new task can then be set in the Task Builder.

Basic steps of creating a task are:
- Giving the task a name.
- Selecting actions and assigning properties to make up the steps of the task.

Related Topics
- Creating Workflow Tasks for Use in Event Rules
- Using Workflows in Event Rules

Creating Tasks (Workflows) for Use in Event Rules

In EFT, once an event is triggered, assuming all conditions are met, EFT can launch one or user-definable Actions. One of those Actions is "Execute Advanced Workflow."

When you create the Workflow, the Advanced Workflow Engine creates a file with an extension of .aml and saves it in EFT’s AWE folder (by default, C:\ProgramData\Globalscape\EFT Server Enterprise\AWE). The filename is the name of the task, prepended with an underscore and the name of the Site. For example, if you create a task called FTP on a Site called Boston, the task’s filename is Boston_FTP.aml. You can then reference that Workflow in the Event Rule Builder in EFT so that, for example, when a certain file is uploaded, the Workflow is executed on that file.

When a new task is created, a message appears (prior to the Create a workflow dialog box) informing you that the Advanced Workflow module is an optional module and that the 30-day trial begins when the first task is created.

To create a task

1. In the EFT Enterprise administration interface, connect to EFT and click the Server tab.
2. Do one of the following:
   - On the main menu, click Configuration > New Workflow.
   - In the left pane, click the Advanced Workflow node.

The Advanced Workflow pane appears.
3. In the right pane, click **New**. The **Create a Workflow** dialog box appears.

4. In the **What do you want to call this workflow** box, specify a name for the workflow. When you add the workflow to Event Rules, the name you specify here appears in the Rule.

5. (Optional) Provide a description of the workflow, then click **OK**. The AWE **Task Builder** appears.

   The tree in the left pane lists the steps that you can add to the task. The right pane displays the steps in the task.

6. Double-click or drag items from the **Available Actions** list to the **Steps** pane to create your task.
   (Refer to these Examples.)
7. Use the Run icon on the Debug toolbar to test the steps. You can run it all at once, run only a selected step, or a portion of the task starting with a step other than the first step.

The Output tab displays the result of each step. For example:

```
Executing line 5
Starting Input Box with message "What is your name?"
Creating message box "What is your name?" >
Populating variable "theUserName"
Finished Input Box "What is your name?".
The step was okay.
```

8. After you have created your task, click Update and Close. The workflow appears in the Advanced Workflows node of the Site tree and is ready to be used in Event Rules.

9. (Optional) In the Advanced Options area, specify the number of seconds after which to terminate the workflow if it fails to execute. The Terminate the process check box is selected by default. If you clear the check box, the process will run indefinitely, which is probably not the desired behavior.

10. (Optional) Specify the level of debug logging in the Debug log level box, None, Minimal, Normal, or Verbose (None is the default). Click View log folder to view the CSV logs created by this workflow, saved in `<installation_folder>\AWE\Temp`. If you enable logging, you should manually delete the files after you’re done with them or create a Scheduled event in EFT to delete them automatically.

11. Click Apply to save changes on EFT.

Your workflow is now ready to insert into an Event Rule. Refer to Using Advanced Workflows in Event Rules for more information.

### Using Advanced Workflows in Event Rules

After you create Advanced Workflows, you can add them to Event Rules. AWE workflow names can appear in reports and logs.

> If a workflow that is used in an Event Rule is removed from the Advanced Workflows node, any Event Rule that includes that workflow will continue to execute, but will fail when trying to execute the deleted workflow (similar to Commands).

#### To add a workflow to an Event Rule

1. Create an Event Rule.

2. In the Actions list, click Execute Advanced Workflow. The Action is added to the Rule.

   ![Rule Builder](image)

   1. In the Rule pane, click the Advanced Workflow link. The Advanced Workflow dialog box appears.

   ```
   ```
4. The defined workflows appear in alphabetical order in the **Choose a workflow to execute** box. Click the down arrow and click a workflow.

5. (Optional) When you create a new workflow, you can also create new variables. Alternatively, you can pass one or more custom parameters/variables to the Workflow “on the fly” in the **Optional customer parameters** area of the **Advanced Workflow** dialog box. When you click **Add**, the top row in the **Name** column becomes editable. Provide the name of the variable, then press TAB or click in the **Value** column and provide its value. The **Value** column can be a defined EFT variable, but the **Name** column cannot contain a defined EFT variable. For example, the **Name** field can be “any text” and the **Value** field can be “FS.PATH%”. (The GUI translates the periods to underscores in variables that you type here.) AWE variable names are limited to 128 characters.

- If you add a parameter and later decide you do not need it, click the row in the table, then click **Remove**.
- To edit a custom parameter, just click it to make it editable and then type the changes.
The **Advanced Workflow** link in the Rule pane updates with the name of the workflow.

6. Add other Actions as needed, then click **Apply** to save the changes on the Server.

**Adding Task Steps**

Once in the Task Builder, task construction can start by dragging the desired action from the **Available Actions** pane onto the **Steps panel** (as shown below). Doing so will open a properties dialog box for that action where parameters and specifications can be defined. Each type of action has its own set of properties. Specific actions may contain multiple activities to select from. Choosing the desired activity will populate the dialog with the proper properties and settings of that activity.

**To add steps**

1. In the **Actions panel** of the Task Builder, find the action/activity you want to add. You can find an action by clicking the plus signs (+) on any of the categorized folders, or by entering the name of the action or activity in the **Search for action** box at the foot of the Actions panel. When you use the **Search for action** box, the actions list is filtered as you type to include only those actions/activities that contain a match for the entered text (See Finding Actions for more details).

2. Drag the action/activity into the Steps panel (as shown below) or simply double-click the action/activity. The properties dialog for that action opens.

3. Enter properties and click **OK**. The action/activity then becomes a step in your task. Repeat this process to add additional steps in sequence.

4. If you need to change the sequence of the steps, select the step you want to move and use the arrow buttons on the toolbar to move it up or down (for further information, see **Editing Task Steps**).

5. If you need to delete a step, right-click it, and select **Delete**.

6. When you are finished adding steps, click **Run** to test the task. For information on debugging tasks, see **Debugging Tools & Techniques**.

7. When you are finished, click **Save** and then click the close box to close the task.
NOTE: Unlicensed actions and activities will appear opaque in the Actions panel. Double-clicking, dragging or trying to add them to the Steps panel in any other way will generate an error.

Using Add/Insert option

The Ribbon’s Actions command group contains commands that can be used to add or insert a step.

To add an action as the last step

1. From the Actions panel, select the action/activity to add.
2. Click Add Step from the ribbon.

To insert an action in between steps

1. From the Actions panel, select the action/activity to insert.
2. On the Steps panel, select the step that you want the new action/activity to be inserted.
3. Click Insert Step from the ribbon. The new action will be inserted directly below the step you selected.

NOTE: To add steps from the Recent, Frequent, Favorites or Snippets folder, see My Actions Panel.
Creating Variables

Variables can be used to represent data in a task that might be different from computer to computer. Variables can be modified during a task’s execution and can be used to represent and store data that might be different each time the task is run. Once you have created a variable, it becomes available for use in other steps of the task. It also appears in drop-down lists in places where a variable can be entered, and it can be used in expressions by placing the variable name between percent signs (%).

To create a variable

1. In the EFT Enterprise administration interface, connect to EFT and click the Server tab.
2. In the left pane, expand the Site node for the Site that you want to configure, then click the Advanced Workflows node.
3. Edit a task or create a new task to open the AWE interface. The Task Builder appears.
4. On the toolbar click % More > Add Variable. The Variable dialog box appears.
• In the **Variable Name** box, type the name of the variable or click the percent sign (%) to define an expression. Names must begin with a letter, can contain only alphanumeric characters, and cannot contain spaces. Place a percentage sign (%) at the beginning and end of the variable name. Be careful that variable names used do not conflict with a BASIC keyword, function, or instruction. A way to avoid this is to insert unique text to the beginning of all variables names. For example, type `My` in front of all variables so that `%DATE%` (which is a BASIC keyword) would become `%MyDATE%` thus avoiding any name conflict.

• In the **Initial Value** box, type the initial value of the variable or leave it blank for null. The value can be literal or an expression, if surrounded by percent % signs. For example, `%Date%`.

• In the **Description** box, type a description of the variable to remind yourself and others what the variable is used for.

5. Click the **Advanced** arrow to expand the dialog box.

6. In the **Variable type** area, specify whether the variable is **Text** or a **Number**, or leave the default of **Auto**.

7. Select or clear the check boxes as needed:
   
   • **Variable is read only** (Optional): Specifies that the variable is not able to be set to a value other than that specified in the Initial Value parameter. Selecting this option is equivalent to creating a task specific constant.

   • **Variable is private** (Optional): Specifies that the variable is only available to the current task and not to sub tasks started with the Start task action. If this value is left unselected (default) the specified variable will be available to sub tasks.

   • **Treat as parameter** (Required): Specifies that the variable will only be created if it does not already exist. This is particularly useful when a task may have parameters passed to it at runtime (that is, variables of the same name will be created automatically) but a default value should be assumed when debugging and parameters are not present.

   • **Disguise initial value** (Optional): Specifies the value that the variable should be set to initially. As with all parameters, this value may be literal or an expression (if surrounded by percent % signs).

8. Click **Error Causes**.
• Specify what to do in case of certain errors. In the following problems should cause this step to error box, click the down arrow and click All problems, Selected problems, or All except selected.

• If you chose Selected problems, or All except selected, in the Problem Text area, select the check box(es) for the situation(s) that could cause an error, such as Could not create variable or Missing variable name.

• Select the Time out and fail after check box if you want to trigger an error when the step takes longer to process than the number of milliseconds that you specify. 300 milliseconds is the default.

9. Click On Error.
10. If you want to try the step again on errors, select the **Retry the step** check box, then specify the number of times to retry the step, and the error actions:

   a. Beneath the **Execute the following error actions** box, click **Add**. The **Error Action** dialog box appears.
b. In the Error Action box, click an action to perform. The Task to start box changes depending on your selection in the Error Action box. For example, if you specify Write to Windows Event Log, the Task to start box changes to Text, and you can specify the text that is to appear in the Windows Event Log when this task errors.

10. In the After error event box, specify the action to carry out if the step errors: Stop task, continue to next step, Break loop, goto label, or goto step.

11. Click OK to create the variable.

Percent Signs in AWE

A percent sign is used as a special character to indicate the beginning and end of an expression. This allows variables, constants and other expressions to be entered in any parameter of a task’s properties that accepts expressions. For example: %1+1% inside a task will resolve to 2 at runtime. Below are some common scenarios which involve the use percent signs.

SendingLiteral Percent Sign

To send a literal percent sign, escape the percent character. Doubling the percent sign is known as “escaping” the percent sign. For example, to send the literal text 5%, specify 5%%. At runtime the double percentage signs will be recognized and converted into one.

All percent signs are recognized and evaluated by AWE that are contained in any parameter that accepts expressions. For example, if an ‘Open Webpage’ action contained the URL:

http://www.networkautomation.com/urc%admin%/kb_add.cfm

During runtime, AWE will detect the percent signs contained in the URL (%admin%) and automatically classify admin as an expression, which eventually fails the task with the error "A variable in the expression does not exist or is misspelled". In such cases, doubling the percent signs will resolve the issue as shown below:
Concatenating Two Values in an Expression

In some cases, it is necessary to concatenate two variables, functions, or some combination thereof. In this case it is critical to remember that two percent signs always mean a literal percent. So the following will **NOT** work:

```plaintext
%variable1%%variable2%
```

(incorrect)

At runtime, AWE would actually see the expression as `variable1%variable2` which would result in a Syntax Error. The proper way to combine two values in an expression is to use the ampersand (`&`) character as follows:

```plaintext
%variable1 & variable2%
```

(Correct)

Common Misconception

A common misconception is that percentage signs are required anytime a variable is used. This is not always the case because percentages surround expressions which may or may not contain variables. For example, in situations where AWE requires a variable name (e.g., in the 'Set Variable' action), only the variable name is required. It is permissible, however, to use an expression to specify the variable name so the resolved variable is used.

For example, assume we created a named `Var1` with an initial value of `Scott`. If we wanted to change the value of this variable from `Scott` to `Muscle`, we would use a Set Variable action with the properties shown below.

If we add another variable named `Var2` and set its initial value to `Var1`, we could set the `Var1` variable to `Muscle` as we did above by using the expression shown below.

This is because at runtime the `%Var2%` expression will be evaluated to `Var1` before the Set Variable action is executed. The code that executes this example is below.

```plaintext
<AMVARIABLE NAME="Var1">Scott</AMVARIABLE>
<AMVARIABLE NAME="Var2">Var1</AMVARIABLE>
<AMSET VARIABLENAME="%Var2%">Muscle</AMSET>
```

**NOTE:** The code can be copied and pasted directly into the Steps pane of the Task Builder.

See Also

- About Variables
- About Arrays
- About Expressions
- Using Expression Builder
Task Variables

Task variables provide a means of sharing common data within a single function or between more than one function. The parameters of a task variable are similar to that of a task function which comprise a Name, Type and Accessibility descriptor. In addition, a task variable also includes a Description parameter as a way to distinguish each variable. Unlike a task function, however, a task variable cannot be marked as Optional. Some advantages of task variables are as follows:

- **Preservation of task-centric values** - Data that is common to an entire task or used by one or more functions are candidates for elevation to a task variable. An example could be a variable to indicate whether the task is in “normal” or “error recovery” mode.

- **Can be used as a “named constant”** - A constant is defined at the system level. However, there are situations where a value is meant to be preserved throughout the lifetime of a task, regardless of function or external task, but not dictated by the system configuration. A task variable can accomplish this, optionally with the read-only option set.

- **Allows information-protection and controls access by external tasks** - This can be accomplished by using the Accessibility preference of a task variable. They control which task variables are merged into a parent task or accessible through a task object.

Task variables, functions and events are created and managed through a drop-down UI display that is accessible from the Steps panel of Task Builder (as shown below).

NOTE: Task Variables are different from Local Variables which are created using the Create variable activity. In addition, Task Variables are additional features aimed for advanced users. You can build tasks without the use of Task Variables and may choose to use them at your own pace.

Scope and Accessibility provide two different capabilities and serve two different but complimentary roles necessary to realize the full benefits of task variables. They are primary building blocks that can be used to build a solid foundation for a rich object-oriented approach to tasks.

NOTE: Task variables are scoped outside of task functions, and all task variables are accessible to all task functions in that task.
**Scope** limits a task variable’s visibility within a task (e.g., using a variable from “the inside”). Without proper scoping, it would not be possible to provide some of the best advantages to function-oriented design, including information-protection, modularity, maintainability and recursion. Scoping helps take large, unruly tasks and enforce logical restrictions to their structure to provide greater readability and maintainability. They can help optimize and enhance task execution speed by reducing the amount of variables that are present in the system at a given time. Scoping also helps avoid unintentional and confusing data changes and variables accesses, leading to easier debugging, and cleaner and more reliable tasks.

**Accessibility** provides the fundamentals of information-protection, encapsulation and interfacing, all of which are essential for an object-oriented approach to a language. The ability to set task variables and functions as public or private gives a task developer greater control over how a task is used by another task. It makes tasks more portable, manageable and documentable by providing outside access only to those parts of the task that are meant to be used, while providing the user full flexibility of functions. Two levels of accessibility are supported:

- **Public** - The variable or function is visible and accessible to external tasks.
- **Private** - The variable or function is not visible or accessible to external tasks.

The accessibility of a task variable can be adjusted to suit the information-protection needs of the task as it relates to an external or sub-task. By default, a task variable is public, meaning it is accessible to an external task or external task function. This is accomplished in different ways, depending on how the external task is referenced, but the behavior of the task variable is identical regardless.

**Sub-Tasks**

Sub-tasks are task files that are executed within another task by using the Start sub-task activity. In this situation, the parent task’s (e.g., the task executing the Start sub-task step) basic functions, extended functions, public task functions, public task variables, and any local variables (unmarked as private), created up to point where the Start sub-task step is encountered, are accessible to the sub-task. All other variables and functions are not accessible to the sub-task. Conversely, because the Start Task activity is a synchronous operation, the sub-task’s public functions and public task variables are not accessible from the parent task.

**Using Regular Expressions**

Several AWE Actions, notably the Find Text and Replace Text actions, allow the use of "regular expressions." Regular expressions are powerful notations that allow a wide range of text searching using formulas specific to string manipulation. This article describes the regular expression syntax used in AWE, whether you are searching for text or replacing text.

**Searching for Text**

Text can be found in a string using a regular expression by specifying a “match expression”. A Match expression operates on a single line of text at one time. No match can span multiple lines of text. Match regular expressions are composed of the following:
• **Period (\'.\')** Matches any single character except newline. A newline (internally) is really two characters in a specific order - <carriage return> followed by <linefeed>. To match a newline, you must always explicitly specify a newline.

• **Caret (^)** Matches at the beginning of a line only. A ^ occurring ANYWHERE in the match expression (except within a character class) is interpreted in this manner. This allows meaningful use of ^ in combination with grouping or alternation (see below).

• **Dollar sign ($)** Matches at the end of a line only. As with ^ the $ character retains its special meaning anywhere within the expression (except in a character class).

• **Backslash (\)** Followed by a single character matches that character. For example, \* matches an asterisk, \n matches a backslash, \$ matches a dollar sign, etc.

**The following sequences have special meaning:**

- \s space (ASCII #32)
- \t tab (ASCII #9)
- \b backspace (ASCII #8)
- \r return (ASCII #13)
- \l linefeed (ASCII #10)
- \n newline (#13 followed by #10)
- \p pipe character |
- \w word delimiter. Matches any of \t\s!"&()*+,-./:;<=>?@\[\]^`{|}~
- \h hex character. Matches any of 0123456789ABCDEF

The special characters above should be used to produce instances of blanks and tabs. Case is ALWAYS significant when using the special characters. Thus \s matches a space while \S matches a capital letter S. A single character not otherwise endowed with special meaning matches that character. Thus z matches a single instance of the letter z.

A string enclosed in brackets [] specifies a character class. Any single character in the string is matched. For example, [abc] matches an a, b, or c. Ranges of ASCII letters and numbers can be abbreviated as, for example, [a-z0-9]. If the first symbol following the [ is a caret (^) then a negative character class is specified. In this case, the string matches all characters EXCEPT those enclosed in the brackets. For example, [^a-z] matches everything except lower case characters (and newlines)

The special characters defined above may be used inside of character classes with the exception of \n, \w and \h, which are shorthand for their own character classes. If the characters - or ] are to be used literally inside of a character class, they should be preceded by the escape character \. Note that *+O\!^\$\& are not special characters when found inside a character class.
Using Closures

A regular expression followed by * matches zero or more matches of the regular expression. This is referred to as a closure. Thus ba*b matches the string bb (no instances of a), bab (one instance), or baaaaab (several instances).

A regular expression followed by a + matches one or more matches of the regular expression. This is another type of closure. In this case ba+b will not match bb, but it will match bab, or baaaaaab.

A regular expression followed by a ? matches zero or one matches of the regular expression. This is another closure. Here, ba?b will match bb or bab, but not baaaaaab.

Concatenated Expressions

Two regular expressions concatenated match a match of the first followed by a match of the second. Thus (abc)(def) matches the string abcdef.

Alternation

Two regular expressions separated by | match either a match of the first or a match of the second. This is referred to as alternation. Any number of regular expressions can be strung together in this way. Alternation matches are tested in order from left to right, and the first match obtained is used. Then the remaining alternate expressions are skipped over.

Grouping Expressions

A regular expression enclosed in parentheses () matches a match of the regular expression. Parentheses are used to provide grouping, and may be nested to arbitrary depth. Open and close parentheses must be balanced. For example, the following two expressions are not equivalent, and the second probably expresses what was intended:

- PROCEDURE|FUNCTION
- (PROCEDURE)|(FUNCTION)

The first expression is equivalent to

PROCEDURE(E|F)UNCTION

The second expression matches either of the two words in their entirety.

Tagged Matches

A regular expression enclosed in curly braces {} forms a tagged match word. Whatever was matched within the braces may be referred to by a Replace expression in a manner to be described. Tagged match words may not be nested. Open and close braces must be balanced. A maximum of nine tagged match words can be referenced by the Replace expression. Note that the use of curly braces in expressions is meaningless. However, these expressions share an expression interpreter with the Match expressions, so no exception is raised. For example, consider the expression

- b(a*)b.
If the string being tested is 'bab', then the tagged match word contains a single 'a'. If the string being tested is 'baaaaaab', then the tagged match word contains 'aaaaaa'. If the string tested is 'bb', then the tagged match word is empty.

**Order of Precedence**

Regular expressions are interpreted from left to right. The order of precedence of operators at the same parenthesis level is $\{\}$, then *+, then |, and then concatenation.

Tag braces are interpreted strictly from left to right and do not control precedence in any way. The first tagged match word found is given a tag of 1, the second a tag of 2, and so on up to a maximum tag of 9. The tag number that each word receives is based on when it is encountered in the line. If tags are skipped over as a result of alternation, then any remaining tags in a line receive shifted tag numbers. For example, consider the expression:

- $(\text{FUNCTION})|((\text{PROCEDURE}))\$s+\[^s\]+$

If a line contains the word PROCEDURE then the word following PROCEDURE has a tag number of 2. If a line contains the word FUNCTION, then the word following FUNCTION has a tag number of 1. It is up to the user to take advantage of this behavior. Generally, it is good practice to surround an entire set of alternates with tag markers:

- $\{(\text{FUNCTION})|((\text{PROCEDURE}))\}s+\[^s\]+$

**Replacing Text**

Replace regular expressions are constructed the same way as Match regular expressions, but the number of operators is reduced. The replacement process occurs in the following manner:

The Match expression finds a string of text that starts at the left-most position in the input line that matches, and continues to the right-most position that matches. The string of matched text is operated upon by the Replace expression. The Match expression is then tried again on the input, starting at the first position beyond the previous match string. This recurs until the end of line is found.

**Replace expressions are composed of the following:**

- **No spaces**: The regular expression may NOT contain any blank space. The special characters below should be used to produce instances of blanks, tabs and the null expression.

- **Null replace**: If a null Replace expression is desired, the special symbol $\z$ is used to indicate a null expression. Null Replace expressions are used to delete text strings.

- **Single character**: A single character not otherwise endowed with special meaning.

- **Backslash($\backslash$)**: Followed by a single character matches that character. For example, $\backslash*$ matches an asterisk, $\backslash\backslash$ matches a backslash, $\backslash\$ matches a dollar sign, etc. A $\backslash$ followed by a single character sends that character to the output. In this way a $\backslash\&$ writes an ampersand and $\backslash\backslash$ writes a backslash.
The following sequences have special meaning:

- `\s` space (ASCII #32)
- `\t` tab (ASCII #9)
- `\b` backspace (ASCII #8)
- `\r` return (ASCII #13)
- `\l` linefeed (ASCII #10)
- `\n` newline (#13 followed by #10)
- `\z` null expression

Unless a newline combination is explicitly matched in the Match expression, it is not necessary to explicitly specify newlines in the Replace expression. Each newline of the input text line is written out in the unmatched category of output using the specified line terminator.

Another special case occurs when `\` is followed by a single digit in the range of 1 through 9. In this case the tagged match word found by the Match expression is sent to the output. If a tagged match word for that tag number was not defined, or if the tagged match word doesn’t match anything, then nothing is output. The tagged match words can be output in any order and can be repeated any number of times.

An ampersand (`&`) appearing in the Replace expression causes all text matched by the match expression to be sent to the output. The ampersand can appear in the Replace expression as many times as desired.

Examples:

Return the first letter of the input line:

- `^[A-B]`

Return everything in the line except for the first word

- `\s.*`

Return the number 123 from the text: This is the number 123

- `[0-9]`

See Also

- Introduction to AML
- Using Expression Builder
- Percent Signs in AWE
Using the Expression Builder

The Expression Builder is a valuable tool used to assist in the creation and examination of expressions. In programming, an expression is a combination of variables, constants, functions, and operators that represents a particular value. Many procedures or actions that one might want to perform with expressions are already contained in AWE’s list of available actions. However, expressions can be used to further expand the capabilities of an action or add more intelligence to a task due to its ability to resolve dynamic data during task execution. The Expression Builder essentially incorporates all of the elements needed in order to create a properly formatted expression for use in any step of an AWE task.

The Expression Builder provides a hierarchical view of all supported functions, constants, datasets, variables, and arrays. It also provides a list of system objects, such as AMError, which can be used to determine a variety of key elements that occur during runtime. If an error occurs, an AMError object can be used to determine detailed information about the error that occurred, such as corresponding error code. Additionally, if a task generates an error, the Error object can be used to acquire a description of the error that occurred, which step it occurred on and which task generated the error.

The Expression Builder also allows for the creation of arrays and variables directly from its UI (User Interface). Double-clicking the Add Variable or Add Array option located in the bottom part of the Expression Builder’s left window pane will open the Create Variable or Create Array properties dialog. These are identical dialog boxes that normally appear when selecting the Create Variable or Create Array action in the Available Actions pane.

Accessing Expression Builder

The Expression Builder interface can be accessed in the Task Builder by clicking the percent sign (%) which is available while editing the steps of a task. In the Properties dialog box of any action, clicking inside any text box that accepts an expression will allow the percent sign to appear to the right of the box. Click the icon (or the F2 key) while in an action’s properties to open the Expression Builder interface. An expression may then be visually constructed and inserted into the step parameter.

The Expression Builder can also be accessed from the Task Builder ribbon by pressing the Application Button and navigating to Tools > Expression Builder.
Sections of the Expression Builder

The Expression Builder is arranged in four primary sections as illustrated above. Each section plays an important role in the creation of an expression. Each section is described in further detail below:

1. **Folders Pane**: The lower left window pane includes folders containing all available functions/extended function, variables, datasets, constants, etc. This section also includes the ‘Create Variable’ and ‘Create Array’ options, which allows for the creation of variables and arrays directly from the Expression Builder interface.

2. **Contents Pane**: The lower right window pane displays the contents of a folder selected from the lower left Folders pane. Help for each function can be found by selecting the desired function from this window pane and pressing the F1 key, or by right-clicking and selecting Help from the popup menu that appears.

3. **Expression Pane**: When double-clicking an item from the Contents pane, the expression is displayed in the upper Expression pane. Help can also be found for a function from this pane by placing the insertion cursor inside a function name and pressing F1.

4. **List of Operators**: Operators which can be added to the expression are shown just below the Expression pane.

Help for each function can be found by selecting the desired function from the right-hand side of the window and pressing the F1 key, or by right-clicking and selecting Help. Help can also be found for a function by placing the insertion cursor inside a function name in the upper expression pane and pressing F1.
Viewing Expressions

The Expression Builder can be used to view specific objects that are created as a result of specific events. The Variables folder located in the Folders pane of the Expression Builder becomes populated if a Create Variable action is used to create the specified variable(s) in an earlier step. Additionally, the datasets folder becomes populated when the specified datasets are initially created by running the step that creates and populates a dataset names and its contents, however, the dataset name itself is available in the Expression Builder if a step that creates the dataset appears in the task.

For example, imagine creating a task that includes a 'POP3 Email' action as the first step to retrieve email messages from a POP3 mail box. The General tab of this action contains a parameter labeled Create and populate dataset with which specifies the name of the dataset that should be created and populated with the mail message data upon execution as shown in Figure 1-2.

During task construction, one may want to view the variables or datasets created by the Get Email action. In order to display the created datasets from the Expression Builder interface in a subsequent step, the Get Email action must initially be executed in order properly create those datasets. This can be accomplished by highlighting the Get Email step (or any step which creates and populates a dataset) In the Task Builder Steps pane and clicking Run or hitting CTRL+F5. This will run only the highlighted step.

Building Expressions

To build an expression via the Expression Builder:

1. Open Expression Builder from an action dialog box by clicking the percent icon %.
2. Add elements to your expression as follows:
   - Click a folder in the left pane to reveal its contents in the right pane.
   - Double-click elements in the right pane to add them to your expression.
   - Select and replace characters in the expression as needed.
   - Click operators below the expression pane to add them to the expression.
3. When satisfied with the expression, click Insert to insert it into the specified action’s properties.
   When the expression is added to the action parameter, the percentage signs (%) will automatically be added to the beginning and end of the expression.

File Masks & Wildcards

File masks and wildcard characters are supported in any action or trigger parameters that indicate folder or file names, commonly, the File system, Compression, Cryptography and FTP actions. A file mask is essentially a pattern of fixed and wildcard characters used to match folder and file names. They provide a flexible means for identification of a specific file or group of files based on their name and extension. A wildcard character can be used to substitute for any other character or characters in a string. By using masks and wildcards, you can easily specify which files and/or folders should be processed and which should be ignored.
Parameters

File masks can consist of any combination of the following:

- **Fixed characters** - Letters, numbers and other characters allowed in file names.
- **Question mark (?)** - Represents any single character.
- **Asterisk (*)** - Represents any sequence of characters (including no characters at all).

While it is well-defined that a file mask containing only fixed characters uniquely identifies a file (e.g., the file mask `myFile.txt` refers to, and only to, a text file named `myFile.txt`), a file mask containing wildcard characters ? and/or * can refer to single or multiple files.

Multiple File masks

In most contexts, you can specify multiple files or combine several wildcard masks by separating each with a pipe (|) character. For example, the following mask includes all JPG and GIF images contained in `c:\temp\`:

```
c:\temp\*.jpg|*.gif
```

Examples

Wildcards allow a file mask to match multiple folder or file names. The table below provides a variety of file mask examples.

<table>
<thead>
<tr>
<th>File Mask</th>
<th>Description &amp; Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Matches all files containing any amount of characters, with or without extensions (e.g., entering A* would match any file or folder starting with the letter A followed by any amount of characters).</td>
</tr>
<tr>
<td><em>.</em></td>
<td>Matches all files containing any amount of characters and with any extension. Even matches files that don't have an extension (e.g., entering A*.* would match any file starting with the letter A followed by any amount of characters along with any extension).</td>
</tr>
<tr>
<td>?</td>
<td>Matches any single character (e.g., entering A? would match any file starting with the letter A followed by any single character).</td>
</tr>
<tr>
<td>*.png</td>
<td>Matches all files with names containing any amount of characters with a .png extension (e.g., image_name.png, cool_pic.png, 1.png).</td>
</tr>
<tr>
<td><em>.p</em></td>
<td>Matches all files with names containing any amount of characters with an extension starting with the letter p (e.g., document.pdf, image_name.png, business_project, 1.ppt).</td>
</tr>
</tbody>
</table>
### File Mask

<table>
<thead>
<tr>
<th>Description &amp; Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pic*.*</code></td>
</tr>
<tr>
<td><code>*mat?.html</code></td>
</tr>
<tr>
<td><code>??????</code></td>
</tr>
<tr>
<td><code>doc?????.pdf</code></td>
</tr>
<tr>
<td>`file1.txt</td>
</tr>
<tr>
<td>`c:\temp1*.txt</td>
</tr>
</tbody>
</table>

### Setting Custom Step Description

The Description tab located in the properties of any action/activity enables users to modify or fully customize the default description of any step when it appears in the Steps panel. This feature can be used to enter important information or display special notes or instructions with regards to the step. The Description tab is a useful way to convey specific information about a step to others.

**NOTE:** Custom step description is only supported if the Steps panel view is set to Visual mode, which displays the steps in plain English, and not AML mode, which displays the steps in AML (AutoMate Markup Language) format.

### Practical Usage

Typically, when a user finishes entering the properties of an activity, a general description of that activity and its properties are displayed in Task Builder’s Steps panel, assuming the View property is currently set to Visual mode (circled below) and not AML mode. There may be a need to include additional information or modify the default description altogether. The Description tab can be used to enter user-defined information such as instructions, reminders or alerts.
To enter a custom description for a step:

1. Open the properties of the step and select the Description tab.
2. Enable the Use custom step description parameter.
3. Enter the desired description (as illustrated below) then click **OK** to save.

The Steps panel displays the custom description you entered in place of the original description (highlighted below).
NOTE:

- Another way to document instructions, notes or other important information directly onto the Steps panel is by adding Comments.
- If you import a task that includes an unlicensed action/activity, whether or not it contains a custom description, the Steps panel will always display the unlicensed step (in Visual view) as currently unlicensed.
- Running a task with an unlicensed step will always fail at that step. You can bypass the step by disabling it or removing it altogether.

On Error Tab

On every Action, the **On Error** tab specifies what AWE should do if this step encounters an error as defined in the **Error Causes** properties. The default behavior is that task execution stops in the immediate step that encounters the error; the On Error properties allow you to designate how AWE should respond upon the occurrence of an error. For example, a step can send an email, execute another task, or log specific text to the Windows Event Log upon error. In addition, a step can be set to continue to the next step in spite of an error.

**NOTE:** All **On Error** properties are optional entries.

ℹ️ To send an email on error, the default mail server must be defined in the registry. Refer to Sending E-Mails with Workflows for details.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Markup</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retry the step x times</td>
<td>Number</td>
<td>(Disabled)</td>
<td>AM_RETRY=&quot;6&quot;</td>
<td>Indicates the amount of times execution of this step should be retried if it encounters an error as defined in the <strong>Error Causes</strong> properties. This property is disabled by default.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Default</td>
<td>Markup</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------</td>
<td>---------------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Pausing x milliseconds</td>
<td>Number</td>
<td>(Disabled)</td>
<td>AM_RETRYDELAY=&quot;3000&quot;</td>
<td>Specifies the amount of milliseconds this step should wait between retries as defined in the Retry the step x times parameter. This property is disabled by default.</td>
</tr>
</tbody>
</table>
| Execute the following error actions | Text    | (Empty)       | a) AM_RUNTASK="Task1" (Start Task)  
b) AM_SETVAR="Var1;0" (Set Variable)  
c) AM_EMAIL="mm@netauto.com;Task2" (Send Email)  
d) AM_WRITETOEVENTLOG="0" (Write to Event Log)  
e) AM_WRITETOAMLOG="Error Task 2" (Write to Log)  
f) AM_PLAYSOUND="c:\temp\error.wav" (Play Sound) | Allows a variety of actions to be executed upon the occurrence of an error. Click Add to add a new error action. This property is disabled by default. |
| After error event                 | Text    | (Empty)       | a) AM_ONERROR="BREAKLOOP"  
b) AM_ONERROR="CONTINUE" |                                                             |
<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Markup</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include Subfolders</td>
<td>Yes/No</td>
<td>(Empty)</td>
<td>SUBFOLDERS=&quot;YES&quot;</td>
<td>When set to YES, specifies that, if present, subfolders should be searched for files matching the mask specified in the Remote Files parameter. The default value is NO.</td>
</tr>
<tr>
<td>Match Case</td>
<td>Yes/No</td>
<td>No</td>
<td>MATCHCASE=&quot;YES&quot;</td>
<td>Indicates whether the properties set within this activity should be case sensitive in relation to the FTP server. The default value is NO.</td>
</tr>
<tr>
<td>Exclude Mask</td>
<td>Text</td>
<td>(Empty)</td>
<td>EXCLUDE=&quot;*.txt&quot;</td>
<td>Causes this activity to omit any files matching the mask(s) specified. Filenames or wildcard masks (e.g., * or ?) may be used.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Default</td>
<td>Markup</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>---------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Regular Expression</td>
<td>Yes/No</td>
<td>No</td>
<td>RE=&quot;YES&quot;</td>
<td>Specifies whether the value in the Exclude Mask parameter is a regular expression. Regular Expression, also referred to as regex or regexp, provide a concise and flexible means for matching strings of text.</td>
</tr>
</tbody>
</table>

If the step errors, carry out the following actions:

If enabled, this allows the task developer to specify that the step should be retried a chosen number of times along with a chosen amount of time to wait between retries, before actually considering it a true step failure, in which case, the remainder of the error actions will be carried out (option disabled by default).

Execute the following error actions

Allows a variety of actions to be executed upon the occurrence of an error. Click Add to open the Error Action dialog box and specify one of the following actions to take when an error occurs:

- **Start Task** — Start another task upon error. Select the task from the drop-down menu. This action can be used to start a task to rectify the error. In the Task to start field, select the drop-down list of currently available set of tasks.

- **Set Variable** — Sets the value of an existing variable. Can be used to set a variable with the return code. Select the variable in the Variable name box, and enter the value in the Variable value field.

- **Send E-mail** — Sends an e-mail upon error. Useful way to alert the proper recipient(s) of the problem. The message will be populated with information about the specific error that occurred as well as other important task information. In the To and From boxes, enter the recipient e-mail address and an e-mail address to be used in the From header. (Many SMTP mail servers require a valid From address.)

- **Play sound** — plays a specified sound file. In the Sound File box, click Open Folder and navigate to the sound file to be played.
• **Write to Windows Event Log** — Writes the specified message to the Windows Event Log upon error. In the **Text** box, enter the text or specify the variable to be written to the log.

• **Write to Log** — Writes the specified message to the AWE Event Log upon error. In the **Text** box, enter the text or specify the variable to be written to the log.

**After Error Event**

Specifies what the step should do regarding execution directly after the error event occurs.

• **Stop task** — Stops the task and initiates task level error handling. This is the default setting.

• **Continue to next step** — Continues the task in spite of the step error. Use this option if the error is non-critical or if additional error handling steps are specified in a subsequent step.

• **Break Loop** — Specifies that if the step is inside a loop, it should stop looping and go to the step (if any) following the End Loop.

• **Goto label** — Goes to the specified label. The label must be created in and earlier step using the ‘Label’ action. In the **Label** box, select the label. If the task has been saved, existing labels appear on the drop-down list. Labels can be viewed under the Labels tab of the Debug pane.

• **Goto step** — Specifies that AWE should skip to another step in the task. In the **Step** box, specify the step number to go to.

**NOTE:** Too many Goto instructions can produce extremely unmanageable tasks. It is usually better to create a separate task and use **Start Task** instead.

**Error Causes Tab**

The **Error Causes** properties allows you to modify how this step should behave upon the occurrence of an error. The default behavior is to monitor for the occurrence of all errors, however, these properties can be set in order for this step to ignore certain errors or only react to specific errors. It also allows you to set the length of time this step should be permitted to execute before causing a time out error. If any **Error Causes** conditions are met during task execution, the procedures set under the **On Error** properties are carried out.

**NOTE:** All **Error Causes** properties are optional entries.

**Practical Usage**

Particularly used as a way to ignore non-critical errors or overlook anticipated issues in order for task execution to proceed without interruption. Can also be used to set a time out value for time-sensitive steps.
Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Markup</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The following problems should cause this step to error</td>
<td>Text</td>
<td>All Problems</td>
<td><code>AM_ERRORS=&quot;18999,18998,20008&quot;</code></td>
<td>Allows you to select / de-select specific errors that will cause this step</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>to fail.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Default</td>
<td>Markup</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------</td>
<td>---------</td>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Custom problem codes</td>
<td>Text</td>
<td>(Empty)</td>
<td>AM_ERRORS=&quot;18999,18998,20008&quot;</td>
<td>Allows a custom error code to be entered in case it does not appear in the list of errors. This parameter will adhere to the specifications set under the parameter labeled <strong>The following problems should cause this step to error.</strong> Variables can be used in this parameter. <strong>NOTE</strong>: Use AMError to determine specific information about an error.</td>
</tr>
<tr>
<td>Timeout and error after</td>
<td>Number</td>
<td>(Empty)</td>
<td>AM_TIMEOUT=&quot;20&quot;</td>
<td>If enabled, allows you to set the total amount of time this step should be allowed to execute before generating a timeout error. For example, a SQL Query step is set to time out after 30 seconds. If the query took longer than 30 seconds to complete, the step would generate a time out error and the instructions set in the <strong>OnError</strong> properties would be carried out. This parameter is disabled by default.</td>
</tr>
<tr>
<td>Timeout scale</td>
<td>Text</td>
<td>milliseconds</td>
<td>AM_TIMEOUTSCALE=&quot;seconds&quot;</td>
<td>The scale in which the timeout value should be set to.</td>
</tr>
</tbody>
</table>
Attachments

Attachments allow files of any type to be embedded within a task so that third-party elements can be incorporated for greater portability. At runtime, attachments are automatically "unwrapped" to a local directory where they can be accessed by the task.

Using Attachments

Once an attachment is added to a task, it can be accessed in any task step by using \%\texttt{AMAttachment.Name}\% where \texttt{[Name]} is the name of the attachment specified in the \texttt{Name} parameter of the Attachment dialog. \texttt{AMAttachment.Name} is essentially used as a placeholder for the attachment file during runtime.

NOTE: Multiple files of any type and size can be attached to a task, however, be aware that the size of the task file expands with the size and number of attachments, which may affect performance. Multiple or large sized attachments used in a single task may slow down task execution.

Adding Attachments

Attachments are added and managed via the Task Builder’s Attachments Debug Panel. This section displays any files that are currently attached to the task along with their original location.

To attach a file to a task:

1. Right-click anywhere inside the Attachments debug panel and select Add from the context menu (displayed below).
2. Enter the following information:
o Name - The name of the attachment (this is what you will use as a placeholder for the file).

o Attachment file - The full path and filename of the attachment file. Use the folder icon to browse to the file you wish to attach.

o Description - An optional description of the attachment.

3. Upon completion, click OK. The new attachment and its properties are displayed in the Attachments Debug panel.

**Auditing Workflows**

As with other Event Rule Actions, AWE Workflows are audited to the Auditing and Reporting Module (ARM) database and can appear in logs and reports--regardless of the type of database you use*

*For EFT to connect to any database, the proper drivers need to be installed on the EFT computer. If the correct client-side software (drivers) is installed on the EFT computer, AWE can make the database connection string to get to that database.

For all Event Actions, the following items are audited:

- Time stamp
- Site Name
- Event Name
- Action Types such as move, copy, OpenPGP, and send e-mail.
- Action Parameters - These are runtime values passed to the action, not the replacement variables. For AWE workflows, this is the path to the temporary file associated with the workflow executed. This file contains more detailed debug logging if enabled for that particular workflow.
- Failed Action Flag - This is captured if this action is the result of a FAILURE sequence on a prior action.
- Action Result Code
- Result

**Backing Up Tasks**

If you create numerous, complex tasks, it is a good idea to back them up to another location, in case they are accidentally deleted in EFT. You can also create backups to share workflows between Servers and Sites. You can create an Event Rule to back the files up periodically by creating a task using the Copy File Action. Or, you can create a .zip file of all of the .aml files.

**To backup workflows**

1. Define a Timer Rule. Specify the frequency depending on how often you create new tasks.
2. Add the Copy/Move (push) file to host Action to the Rule.
   - For the **Source** path, specify the location of the task (.aml) files. (Do **NOT** select the **Delete source file** check box!) The default location is a hidden folder named %systemroot%\ProgramData.
   - For the **Destination** path, specify a location on a remote drive (in case the local drive fails).

3. Click **Apply**.

**Bookmarks**

Bookmarks are a means of marking steps in a task so you can find them easily during construction or testing. You can mark steps in your task with bookmarks and then navigate directly to the steps marked simply by clicking the **Next Bookmark** or **Previous Bookmark** ribbon controls (as shown below).

To set a bookmark:

1. In the Task Builder’s **Steps panel**, select the step(s) for which to place a bookmark. To select more than one step, hold down CTRL during selection.

2. On the ribbon interface, select the **Home** tab and click **Bookmark** from the **Step** command group. A bookmark is indicated by a blue, circled icon located to the left of the step number (as shown below).

To navigate to a bookmark:

- Select the **Home** tab and click **Next Bookmark** or **Previous Bookmark** from the **Step** command group.
  
  OR

- Press **F3** to jump to the next bookmark or **Shift+F3** to jump to the previous bookmark.
Copying and Reusing Task Steps

In computer programming, the reuse of code is a common technique that attempts to save time, reduce redundancy and eliminate errors by using the same segment of code in multiple applications. The Task Builder plays a similar role by allowing task steps to be copied and re-used in another section of the same task or inserted onto the steps of a new or existing task. Reusing of steps can be accomplished in several ways depending on whether you want to reuse all steps of an existing task or just certain steps.

In addition, the Import Steps option can import all of the steps of an existing task onto another.

Saving a Copy of a Task

Steps of an existing task can be re-used in a newly created task by selecting the Save As Copy command. This is an ideal method for copying all of the steps of an existing task onto a new task.

To save a copy of a task

1. Open the task you wish to copy steps from.
2. From Task Builder’s Ribbon interface, select File -> Save copy as...
3. Navigate to the desired location, enter a name for the new task, and click Save.

Copying/Pasting Task Steps

The Copy and Paste commands allow you to reuse steps within a single task by copying a group of steps and pasting them onto another section of the task. Because more than one instance of the Task Builder can be opened simultaneously, code reuse can be performed across multiple tasks as well. The Copy command, in effect, copies the .AML code associated to each task step. This code can be pasted onto a specific step on the same task or another task. It can also be pasted onto a document such as Notepad. Alternatively, a Copy Description command is available, allowing the text description of each step, as opposed to the .AML code, to be copied.

To copy steps from one task onto existing steps of another task:

1. Open the task you wish to copy from.
2. Select the step(s) from the task you wish to copy from. To select a group of steps, hold down CTRL during selection. Use CTRL+A to select all.
3. On the Ribbon, select the **Home** tab and click the **Copy** button from the **Clipboard** command group.

   OR

   Right-click the highlighted step(s) and select **Copy** from the context menu.

4. In the Steps panel of the task you wish to copy steps onto, select the step you want to place the previously copied step(s) and do one of the following:
   - On the Ribbon, select the **Home** tab and click the **Paste** button from the **Clipboard** command group.
   - Right-click and select **Paste** from the context menu.
   - On your keyboard, press CTRL + V.

The copied steps are pasted directly above the selected step of the second task.

**To copy steps from one task onto a newly created task:**

1. First, **create a new task**.
2. Open the task you wish to copy steps from.
3. Select the step(s) from the task you wish to copy from. To select a group of steps, hold down CTRL during selection. Use CTRL+A to select all.
4. On the Ribbon, select the **Home** tab and click the **Copy** button from the **Clipboard** command group.

   OR

   Right-click the highlighted step(s) and select **Copy** from the context menu.

5. Go to the new task, right-click anywhere inside the empty Steps panel and select **Paste** from the context menu.

The copied steps are pasted directly onto the Steps panel of the new task.

**To copy and paste step descriptions:**

1. From the Steps panel, select the step(s) in which to copy the description from. To select more than one step, hold down CTRL during selection or use CTRL+A to select all steps.
2. From the Ribbon interface, select **Copy -> Description** from the **Clipboard** command group or right-click and select **Copy -> Description** from the context menu.
3. Go to the location in the Steps panel where you want to paste the copied descriptions and do one of the following:
   - On the Ribbon, select the **Home** tab and click the **Paste** button from the **Clipboard** command group.
- Right-click and select **Paste** from the context menu.
- On your keyboard, press **CTRL + V**.

**NOTE:** You can also paste copied steps or step descriptions in other documents such as e-mail messages or text files.

### Using Snippets

Another way of using a group of steps within a specific task is to save the steps as Snippets. Snippets provide an easy way to implement frequently used code into a task. They are particularly useful to those who use specific activities that contain commonly used properties or settings. Instead of re-entering the properties for a specific activity or series of activities every time they’re added as task steps, users can save existing steps as a snippet and simply drag and drop the snippet wherever it is needed in any task.

### Deleting Workflows in EFT

You can delete the workflows that you no longer need and are not being used in Event Rules.

**WARNING:** If the workflow is used in an Event Rule and the workflow is deleted from the **Advanced Workflow** node, the Event Rule will fail when it attempts to execute the workflow.

**To delete workflows**

1. In the left pane, expand the **Advanced Workflows** node.
2. Right-click the workflow you want to delete, then click **Delete**. The workflow is removed from the tree.

### Editing Task Steps

Along with the ability to edit the **Properties** of individual task steps, many adjustments can be performed on any step once they are added to the Task Builder’s **Steps panel**. You can move, indent, disable or delete one or more existing steps. You can perform cut/copy/paste commands as well. Commands are conveniently located on the **Ribbon** or via a context menu that appears when right-clicking a specific element. Certain commands become active on the Ribbon only for the items that they apply to, making it easier for you to discover/detect relevant commands. The Quick Access Toolbar contains **Undo** and **Redo** commands that apply to the various edit operations allowing you to undo an operation in case you make a mistake or redo an operation in case you change your mind.

The properties of any task step can be easily viewed or modified once it is added to the Steps panel of the Task Builder.

**To view or modify the properties of a step:**

1. In the Steps panel of the Task Builder, select the step that contains the properties you wish to view or modify and click the **Edit** command on the Ribbon (shown below).
1. Double-click the step
2. Right-click the step and select **Edit Step** from the context menu that appears.

The properties dialog of that step opens. Make appropriate changes to any parameters.

3. Click **OK** to save changes and close the properties dialog box. If you decide not to save changes, simply click **Cancel**.

A developer may want to indent or move task steps up or down, or to some other position. Task Builder allows you to increase/decrease indentation of one or more steps or you can use an auto-format command to format the indentation of a block of steps automatically. In addition, Task Builder supports various methods of moving steps up/down a specific number of lines.

**To increase or decrease indentation:**

1. In the Steps panel of the Task Builder, select the step(s) you want to indent. To select a block of steps, hold down CTRL during selection.
2. Do one of the following:
   - From the Ribbon’s **Home** tab, click the **Indent** button to increase indentation or the **Decrease Indent** button to decrease indentation (as shown above).
   - Right-click the highlighted step(s) and select **Indent/Decrease Indent** from the context menu.
   - On your keyboard, press CTRL + RIGHT to increase indentation or CTRL + LEFT to decrease indentation.

**NOTE:** The indentation value can be modified via **Task Builder Options - Formatting**.

**To move steps up/down using Move Command:**

1. In the Steps panel of the Task Builder, select the step(s) you want to move up/down the step sequence. To select a block of steps, hold down CTRL during selection.
2. Do one of the following:
   - To move up one step, click the **Move Up** command from the Ribbon, enter CTRL + UP or right-click the step(s) and select **Move Up** from the context menu.
To move down one step, click the **Move Down** command from the Ribbon, enter CTRL + DOWN or right-click the step(s) and select **Move Down** from the context menu.

**To move steps up/down using Cut & Paste:**

1. In the Steps panel of the Task Builder, select the step(s) you want to move up/down the step sequence. To select a block of steps, hold down CTRL during selection.
2. Click the **Cut** command from the Ribbon (shown below), enter CTRL + X or right-click the step(s) and select **Cut** from the context menu.
3. Select the step you wish to paste the previously cut step(s) and click the **Paste** command from the Ribbon (shown below), enter CTRL + V or right-click the step(s) and select **Paste** from the context menu. The step(s) are pasted directly above the selected step.

**NOTE:** Indentation or movement of multiple sequenced steps (e.g., steps 3, 4 and 5) or non-sequenced steps (e.g., steps 3, 6 and 8) are supported.

**To enable/disable auto-formatting of steps:**

- From the Ribbon’s **Step** command group, click the **Format Steps** command (shown below). This enables/disables auto-formatting of steps. When enabled, blocks of steps contained within a conditional (If) statement or Loop process are formatted (indented) to improve distinction (this command is enabled by default).

Steps of a task can be disabled or re-enabled so that you can use or test different versions of the same task. The **Disable** button on the ribbon or context menu acts as a toggle to disable an enabled step or re-enable a disabled step. When a step is disabled, it is greyed out by default in the Steps panel and it is treated as a comment and ignored when the task runs.

**To disable or enable a task step:**

1. In the Steps panel of the Task Builder, select the step(s) you want to disable. To select more than one step, hold down CTRL during selection.
2. Do one of the following:
   - Click the **Disable** button on the ribbon under the Home tab.
   - Right-click and select **Disable** from the context menu.

**NOTE:** To re-enable, follow the same instructions.

**To delete a step:**

1. In the Steps panel, select the step or steps you wish to delete. To select more than one step, hold down CTRL during selection.
2. Do one of the following:
• Click the **Delete** button located on the Ribbon under the **Home** tab.

• Right-click and select **Delete** from the context menu.

By default, the Steps panel displays numbers to represent each step in the task are displayed for each step in the Task Builder’s Steps panel along with an icon designating the type of action used. Also the text (or AML code) that appears in this panel is word wrapped by default allowing all text to be visible in the window regardless of its size. Nonetheless, these options can be customized according to each user’s preference. For instance, one may d

**To turn Word Wrap off/on:**

1. From the Ribbon, select the **Home** tab.

2. From the Layout command group, click the **Word Wrap** button to toggle Word Wrap on/off (shown below).

**To disable/enable step numbers :**

1. From the Ribbon, select the **Home** tab.

2. From the Layout command group, click the **Step Numbers On/Off** button to toggle step numbers on/off (also shown below).

**Switching from Visual to AML View**

The Task Builder offers two ways in which to view the steps of a task. The default appearance displays a plain-English, easy to comprehend text description of each step. They can also be displayed in **AML (Markup Language)** format. When set to this view, those who understand AML can edit their code directly from the Task Builder’s **Steps panel** or copy the code onto an external text editor such as Notepad.

**To switch from plain text to AML or AML to plain text:**

1. From the Ribbon, select the **Home** tab.

2. In the View command group, select **AML** to switch to AML code view or **Visual** to switch to plain text view (shown below).

![Task Builder ribbon with AML and Visual options highlighted](image)

**To edit code in AML view:**

1. From the Steps panel, select the step to edit.

2. Click the step again to make the code editable.

3. Make the appropriate changes.
Undoing / Redoing Changes

To allow easy creation and editing of tasks, an Undo and Redo option is included on the Quick Access Toolbar (shown below).

To undo Changes:

- Click the **Undo** button on the Quick Access Toolbar or press CTRL + Z. You can perform as many Undos as you like.

To redo a change after an undo:

- Click the **Redo** button on the Quick Access Toolbar or press CTRL + Y. The **Redo** button is normally grayed out and becomes active only if an **Undo** is performed.

Finding & Replacing Text

Task Builder encompasses a comprehensive **Find** and **Replace** feature found in the ribbon’s **Home** tab that enables you to search for every occurrence of a particular word or phrase in one or more open tasks and optionally, replace them with the specified text. You can scope searches to the current task, current function, selected steps or all open tasks. Options and rules that you set can restrict the results to those that meet the criteria that you specify. For example, you can specify what direction you want to search or require that only text that matches the case of the text you typed should be found.

The **Find** and **Replace** controls are located in the Task Builder’s ribbon under the **Home** tab (displayed below).

Importing Workflows (Tasks)

Although creating an Advanced Workflow is quick and easy, you might want to share workflows between Sites, import them from other users, or import them from Globalscape Customer Support. When you create a workflow, the Advanced Workflow Engine creates a file with an extension of **.aml** and saves it in EFT’s **AWE** folder. The filename is the name of the workflow, prepended with an underscore and the name of the Site.

*For Sites in an active-active (HA) configuration, AML files are saved in the shared configuration location.*
Importing workflows into EFT

1. In the EFT administration interface, connect to EFT and click the Server tab.
2. In the left pane, click the Advanced Workflows node. The Advanced Workflow manager appears in the right pane.
3. In the right pane, click Import. The Open dialog box appears.
4. In the Open dialog box, navigate to the workflow that you want to import (an .aml file), then click Open. The imported workflow appears in the list of workflows.

Print Task Steps

You can print the steps of a task as a means to save a hard-copy version. Printed data can be in AML mode or Visual (plain text) mode.

To print a task

1. Click File -> Print (as shown below).
2. Enter the print specifications in the Print dialog box, and click OK.

AMError

When a task generates an error, it automatically creates a dataset called AMError. This dataset can be used within a task to determine specific characteristics about the error that occurred, including unique error number, error description, step number that generated the error and more. If a subtask fails, AMError values can be passed to the parent task. If the task that failed is part of a workflow, AMError information can be passed to any task or an Evaluation object succeeding the task that failed.

AMError is a standard dataset and can be used much like the datasets created by the SQL Query and E-Mail activities. The fields of AMError can be used to examine properties of the specific error in order to determine task failure.
For example, using a Message Dialog activity with the Message to display parameter set to:

%AmError.ErrorDesc%

would display a message box during task execution returning a textual description of the error.

Parameters

Every instance of AMError is populated with the following fields:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMError.ErrorDesc</td>
<td>String</td>
<td>Returns a textual description of the error that occurred.</td>
</tr>
<tr>
<td>AMError.ErrorNumber</td>
<td>String</td>
<td>Returns the error number associated with the error text.</td>
</tr>
<tr>
<td>AMError.FunctionName</td>
<td>String</td>
<td>Returns the name of the function that generated the error (if applicable).</td>
</tr>
<tr>
<td>AMError.IPAddress</td>
<td>String</td>
<td>Returns the IP address of the computer that generated an error.</td>
</tr>
<tr>
<td>AMError.ParentTaskName</td>
<td>String</td>
<td>Returns the name of the parent task (if the task that failed was a child task or sub-task).</td>
</tr>
<tr>
<td>AMError.StepNumber</td>
<td>String</td>
<td>Returns the step number that generated an error.</td>
</tr>
<tr>
<td>AMError.StepText</td>
<td>String</td>
<td>Returns a textual description of the step that generated an error in AML format (similar to how it would appear in the Task Builder’s Steps panel if the View parameter is set to AML).</td>
</tr>
<tr>
<td>AMError.TagLine</td>
<td>String</td>
<td>Returns the AML tag line of the step that failed with an error.</td>
</tr>
<tr>
<td>AMError.TaskName</td>
<td>String</td>
<td>Returns the name of the task that generated an error.</td>
</tr>
</tbody>
</table>

**AMError.ErrorDesc**

**Syntax:**

```
%AMError.ErrorDesc%
```

**Type:** String

**Description:**

Returns a textual description of the error that occurred.
### AMError.ErrorNumber

**Syntax:**

```
%AMError.ErrorNumber%
```

**Type:**

Number

**Description:**

Returns the unique error number associated to the error that occurred.

**Example:**

```
<AMRUN AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMSHOWDIALOG WINDOWTITLE="Error Number">Error Number - %AMError.ErrorNumber%</AMSHOWDIALOG>
```

### AMError.FunctionName

**Syntax:**

```
%AMError.FunctionName%
```

**Type:**

String

**Description:**

Returns the name of the function that caused the error (if applicable).

**Example:**

```
<AMRUN AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMSHOWDIALOG WINDOWTITLE="Error Function">The function that caused the error was %AMError.FunctionName%.</AMSHOWDIALOG>
```

### AMError.IPAddress

**Syntax:**

```
%AMError.IPAddress%
```
Type:
Number

Description:
Returns the IP address of the computer where the error occurred.

Example:

```xml
<AMRUN AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMSHOWDIALOG WINDOWTITLE="Error IP Address">The IP address the error occurred in was %AMError.IPAddress%.</AMSHOWDIALOG>
```

**AMError.MachineName**

Syntax:

```
%AMError.MachineName%
```

Type:
String

Description:
Returns the name of the computer where the error occurred.

Example:

```xml
<AMVARIABLE NAME="theError" />
<AMRUN AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMSHOWDIALOG WINDOWTITLE="Error Description">Error Description - %AMError.ErrorDesc%</AMSHOWDIALOG>
```

**AMError.ParentTaskName**

Syntax:

```
%AMError.ParentTaskName%
```

Type:
String

Description:
If a sub-task caused an error, returns the name of the parent task.
Example:

```xml
<AMVariable NAME="theError"/>
<AMRun AM_SETVAR="theError;%AMerror.ErrorDesc%" AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMShowDialog WINDOWTITLE="AMError return value">AMError.ErrorDesc return value: %AMerror.ErrorDesc%
</AMShowDialog>
```

**AMError.StepNumber**

**Syntax:**

```
%AMerror.StepNumber%
```

**Type:** Number

**Description:** Returns the step number that caused the error.

**Example:**

```xml
<AMRun AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMShowDialog WINDOWTITLE="ERROR ">The step number that generated the error: %AMerror.StepNumber%. 
</AMShowDialog>
```

**AMError.StepText**

**Syntax:**

```
%AMerror.StepText%
```

**Type:** String

**Description:** Returns a textual description of the error that occurred.

**Example:**

```xml
<AMRun AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMShowDialog WINDOWTITLE="AMError">The AML description of the step that generated the error: %AMerror.StepText%. 
</AMShowDialog>
```
**AMError.TagLine**

**Syntax:**

```
%AMError.TagLine%
```

**Type:**

String

**Description:**

Returns the AML tag line that generated the error.

**Example:**

```xml
<AMRUN AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMSHOWDIALOG WINDOWTITLE="Error IP Address">The AML tagline that generated the error is %AMError.Tagline%.
</AMSHOWDIALOG>
```

**AMError.TaskName**

**Syntax:**

```
%AMError.TaskName%
```

**Type:**

String

**Description:**

Returns the name of the task that generated the error.

**Example:**

```xml
<AMVARIABLE NAME="theError"/></AMVARIABLE>
<AMRUN AM_SETVAR="theError;%AMError.ErrorDesc%" AM_ONERROR="CONTINUE" FILE="BogusFile.exe" />
<AMSHOWDIALOG WINDOWTITLE="AMError return value">AMError.ErrorDesc return value: %AMError.ErrorDesc%
</AMSHOWDIALOG>
```
## Actions

AWE provides an incredibly easy-to-use, intuitive interface for developing automation applications. The basic building blocks are called actions, which are plain-English, drag-and-drop, fill-in-the-blank dialog boxes that you can use to build AWE tasks. Actions enable the development of sophisticated automation processes, and eliminate the need for code. You could think of them as the steps that do the actual work of a task. You can drag and drop actions in the Task Builder to build a series of AWE steps that collectively make up the task. You can view all available actions in the Task Builder’s Available Actions pane, categorized into folders.

You build a task by dragging actions from the Available Actions pane into the Steps pane of the Task Builder. AWE’s pre-built actions include starting applications, sending keystrokes, clicking controls, uploading files, and much more. These actions can be further expanded by the use of variables, constants, and expressions.

AWE Actions are organized in groups (or categories) to facilitate the development process. For details of each of the AWE Actions, refer to [https://help.globalscape.com/help/awe10/Actions/Available_Actions.htm](https://help.globalscape.com/help/awe10/Actions/Available_Actions.htm).

<table>
<thead>
<tr>
<th>Action Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Directory</td>
<td>Allows automation of common Active Directory operations.</td>
</tr>
<tr>
<td>Array</td>
<td>Use Array actions to create, resize, or set the values of one or more elements in an array.</td>
</tr>
<tr>
<td>Amazon DynamoDB</td>
<td>Use Amazon DynamoDB actions to manage your DynamoDB.</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>Use Amazon EC2 (Elastic Compute Cloud) actions to manage your EC2 session and Elastic Block Store (EBS).</td>
</tr>
<tr>
<td>Amazon RDS</td>
<td>Use Amazon Relational Database Service (RDS) actions to create and manage RDS instances.</td>
</tr>
<tr>
<td>Amazon S3</td>
<td>Use Amazon S3 actions to create and manage S3 buckets.</td>
</tr>
<tr>
<td>Amazon SES</td>
<td>Use the Amazon Simple Email Service (SES) actions to send and manage SES emails.</td>
</tr>
<tr>
<td>Amazon SimpleDB</td>
<td>Use the Amazon SimpleDB actions to run queries on structured data in real time.</td>
</tr>
<tr>
<td>Amazon SQS</td>
<td>Use Amazon SQS (Simple Queue Service) actions to move data between distributed components of your applications that perform different tasks without losing messages or requiring each component to be always available.</td>
</tr>
<tr>
<td>Azure Storage</td>
<td>Azure actions allow you to interact with your Azure storage.</td>
</tr>
<tr>
<td>BASIC Script</td>
<td>Use the BASIC Script action to run BASIC scripts.</td>
</tr>
<tr>
<td>Compression</td>
<td>Allow for automatic compression/decompression of files using commonly-used formats.</td>
</tr>
<tr>
<td>Computer</td>
<td>Use the Computer actions to control the computer’s Start menu Power button features, such as log off.</td>
</tr>
<tr>
<td>Cryptography</td>
<td>Provides encryption automation to secure files before they are transported.</td>
</tr>
<tr>
<td>Database</td>
<td>Enable the execution of SQL queries and stored procedures on any ODBC database. Microsoft SQL Server, Oracle, mySQL, and more are supported.</td>
</tr>
<tr>
<td>Action Group</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Dialog</strong></td>
<td>These actions allow for user interaction with and input to tasks.</td>
</tr>
<tr>
<td>DLL</td>
<td>Use the DLL actions to execute a specified method in a managed or unmanaged dynamic-link library (DLL) and store the return value of the method in an variable.</td>
</tr>
<tr>
<td>Dynamics CRM</td>
<td>Use the Dynamics CRM action to automate a variety of common CRM operations.</td>
</tr>
<tr>
<td>Email</td>
<td>Use the Email action to automate common to complex email operations and simultaneously manage several e-mail accounts from as many mail servers as required.</td>
</tr>
<tr>
<td>Environment Variables</td>
<td>Use this action to perform the creation, manipulation, and management of environment variables.</td>
</tr>
<tr>
<td>Event Log</td>
<td>Use this action to log specified events to the Windows Event Log.</td>
</tr>
<tr>
<td>Excel</td>
<td>Allow for automation of data input and retrieval from Microsoft Excel spreadsheets and templates.</td>
</tr>
<tr>
<td>Exchange</td>
<td>Use this action to create and manage Exchange objects and sessions.</td>
</tr>
<tr>
<td>File System</td>
<td>File actions enable automation of file and folder commands such as copy, move, rename, delete, etc. These actions also allow data to be read from and written to files.</td>
</tr>
<tr>
<td>FTP</td>
<td>Use this action to send specific commands to an FTP server (upload, download, create folder, remove folder, and so on).</td>
</tr>
<tr>
<td>HTTP</td>
<td>Use this action to automate common HTTP operations, such as GET, PUT, POST, DELETE, and more.</td>
</tr>
<tr>
<td>If</td>
<td>Use this action to perform if/then operations in AWE workflows. (See also Select.)</td>
</tr>
<tr>
<td>Image</td>
<td>Use this action to perform common tasks on images, such as resize, rotate, crop, flip, etc.</td>
</tr>
<tr>
<td>Label</td>
<td>Use this task to label and go to a specific point in a workflow.</td>
</tr>
<tr>
<td>Loop</td>
<td>Provide the ability to loop through data sets, files, processes, lists, and more for dynamic automation.</td>
</tr>
<tr>
<td>MSMQ</td>
<td>Use the MSMQ action for MSMQ message delivery and retrieval. It contains individual activities that lets you to create, send, retrieve, clear, delete or wait for MSMQ messages on a local or remote machine.</td>
</tr>
<tr>
<td>Network</td>
<td>Network actions enable drive mapping and the sending of network messages.</td>
</tr>
<tr>
<td>OCR</td>
<td>Use this action to convert typewritten, handwritten, or printed text as well as text contained in images to an variable or dataset, making it possible to search for a word or phrase, store text more compactly, and apply techniques such as text mining or text to speech.</td>
</tr>
<tr>
<td>OpenDocument Spreadsheet</td>
<td>Use this action to manipulate spreadsheet contents, format cells, and protect files.</td>
</tr>
<tr>
<td>PDF</td>
<td>Use the PDF action to streamline document creation and sharing, and automate an assortment of common PDF operations.</td>
</tr>
<tr>
<td>Action Group</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PowerShell</td>
<td>Use the PowerShell action to embed existing PowerShell scripts to a task or point to an external .PS1 file to allow for seamless integration with other actions.</td>
</tr>
<tr>
<td>Printer</td>
<td>Use the Printer action to get, remove, and set a default printer, and print documents.</td>
</tr>
<tr>
<td>Processes</td>
<td>Use this action to start, stop, control, and query a process.</td>
</tr>
<tr>
<td>Registry</td>
<td>Use this action to create, manipulate, rename, and delete registry keys, sub-keys or value data.</td>
</tr>
<tr>
<td>Run</td>
<td>Use this action to run a specified program, application, or document.</td>
</tr>
<tr>
<td>Security</td>
<td>Security actions allow for user authentication in the context of a task.</td>
</tr>
<tr>
<td>Select</td>
<td>Use the Select action instead of If for more complex decisions where there are several possible answers (e.g., favorite color=red, blue, green or yellow)</td>
</tr>
<tr>
<td>Services</td>
<td>The Services action provides various ways to the management of local or remote services, including operations such as starting, stopping, installing, uninstalling, or querying for a list of services installed on the system along with their current state.</td>
</tr>
<tr>
<td>SharePoint</td>
<td>Use this action to automate all SharePoint business and IT processes.</td>
</tr>
<tr>
<td>SNMP</td>
<td>Allow for automatic processing of SNMP (simple network messaging protocol) events that occur on the network.</td>
</tr>
<tr>
<td>Task</td>
<td>The Task action allows task-level operations to be performed during runtime, including starting a secondary task (also known as a child task or subtask) from a parent task or start another task as a separate thread, stop execution of the current task, or call a function defined in the task.</td>
</tr>
<tr>
<td>Terminal</td>
<td>Enable automation of all terminal emulation and Telnet processes.</td>
</tr>
<tr>
<td>Text</td>
<td>Text actions enable the manipulation and parsing of textual data.</td>
</tr>
<tr>
<td>Timer</td>
<td>Used to gauge the execution time between selected steps.</td>
</tr>
<tr>
<td>Type</td>
<td>The Type action is used to create an object from the defined custom type or web service, and define a custom object type or object types defined in a Web Service or import Web Service or .Net assembly types.</td>
</tr>
<tr>
<td>Variables</td>
<td>Provide the ability to create and perform complex programming elements such as creating and running variables and datasets.</td>
</tr>
<tr>
<td>VMWare Guest</td>
<td>Provides VMWare guest OS operations.</td>
</tr>
<tr>
<td>VMWare Host</td>
<td>Provides VM host OS operations.</td>
</tr>
<tr>
<td>Wait</td>
<td>Wait actions provide the ability to pause task execution until a specified system event (e.g., the appearance of a file) transpires.</td>
</tr>
<tr>
<td>Web Service</td>
<td>This action allows you to automate the ability to execute Web Service calls via WSDL (Web Service Definition Language) URI (Uniform Resource Identifier) path.</td>
</tr>
<tr>
<td>WMI</td>
<td>The WMI - Query action can monitor and control managed resources on a local or remote computer, thus improving manageability of computers in a networked environment.</td>
</tr>
</tbody>
</table>
### My Actions

Having quick access to common items is essential for a productive work environment. Task Builder supports this capability with the **My Actions** pane. Similar to Jump Lists in Windows 7 or newer operating systems that enable quick access to frequently and recently used items, **My Actions** can take you right to the actions and activities you turn to each day. The **My Actions** pane is designed to make it easier for you to access common actions or recently used activities quicker by conveniently placing them in common folders directly above the Actions view (circled below).

You can use **My Actions** to open actions/activities, and you can also "pin" favorites, so you can quickly get to the actions that you use every day. You can drag an action from one of the folders in **My Actions** directly to the Steps panel. In addition, you can save actions that contain common properties as "snippets." Snippets store the current properties set for the action so you can reuse it in different tasks. The **My Actions** view houses four types of items:

**Favorites**

Favorites can be used as a placeholder for a collection of preferred actions and activities. When an activity is set as a favorite, a copy of that activity is added to the **Favorites** folder located in the Actions panel (highlighted in red below). That way, users spend less time searching through the long list of available actions and activities. They can simply select the activity from their **Favorites** list. Individual folders can be created to consolidate common activities into specific categories, forming a more organized development environment.
NOTE: You can add actions and individual activities to your favorites list. By default, the list is set in chronological order, starting with the earliest addition and following the order in which they occurred. However, you can resort the list in alphabetical order by right-clicking Favorites and selecting Reorder by title.

To set an Action or activity as a Favorite:

1. Do one of the following:
   - In the Actions panel, navigate to the action or activity you wish to add to your favorites list (only one action/activity/folder can be selected at a time).
   - OR
   - In the Steps panel, select the step(s) that perform the activity you wish to add to your favorites. To select more than one step, hold down CTRL during selection.

2. Do one of the following:
   - Drag the action/activity onto the Favorites folder or any existing sub-folder.
   - On the ribbon, select the Actions tab and click the Add to Favorites option.
   - Right-click the action/activity and select Add to Favorites from the context menu that appears.

To add a Folder/Sub-folder to Favorites:

1. In the Actions panel, right-click Favorites and then click Create Folder. A new folder is added to your favorites list.
2. Rename the newly created folder to the desired name.

3. Add sub-folders under the newly created folder by right-clicking that folder and selecting **Create Folder**.

4. Add actions/activities to a folder under **Favorites** by simply dragging them onto the folder of your choice.

**To remove actions, activities, or folders from favorites:**

1. In the **Actions** panel, under **Favorites**, click the action, activity or folder you want to remove (only one action/activity/folder can be selected at a time).

2. Do one of the following:
   - On the ribbon, navigate to the **Actions** tab and click **Remove**.
   - Right-click the action/activity and click **Remove**.

3. To clear Favorites of all items (including actions, activities and folders), do one of the following:
   - On the ribbon, navigate to the Actions tab and select Clear.
   - Right-click Favorites and select Clear from the context menu that appears.

**Snippets**

For those who use common activities that perform identical operations in many tasks, such activities can be saved as snippets. Snippets provide an easy way to implement frequently used steps into a task. Instead of re-entering the properties for a specific activity or series of activities every time they’re added as task steps, you can save the steps as a snippet and simply drag and drop the snippet wherever it is needed in any task. The main idea is to make the process of reusing as easy as possible to avoid wasting your valuable time to enter the same properties again.

A snippet can consist of one or more task steps that perform common actions. For example, a developer may create a basic Send keystrokes snippet to enter specific keys when a user opens a particular window or clicks a specific button. Other snippets might be used to perform Open file and Save file operations.

Once a snippet is added to a task, its contents act as normal steps (e.g., they can be reorganized or deleted or their properties can be modified as needed). Task steps can be easily saved as a snippet via drag-and-drop. Sub-folders can be added to the Snippets folder so developers can easily organize common or related snippets into categories, creating a cleaner, more manageable development environment.
NOTE: Users will be able to save an unlicensed activity as a snippet and drag and drop that snippet into the Steps panel. However, during Runtime, the unlicensed step will fail, generating an “unlicensed” error.

To add one or more steps to Snippets:

1. Highlight the desired step(s) from the Steps panel. To select more than one step, hold down CTRL during selection. Note that selected steps don’t need to be sequential.

2. Drag the step(s) onto Snippets or any existing folder/sub-folder under Snippets or right-click the step(s) and click Add to Snippets.

To add a Folder/Sub-folder to Snippets:

1. Right-click Snippets and click Create Folder. A new folder is added to your favorites list.

2. Rename the newly created folder to the desired name.

3. Add sub-folders under the newly created folder by right-clicking that folder and click Create Folder.

4. Rename the newly created sub-folder to the desired name.

Recent / Frequent

Task Builder automatically saves a history of recently and frequently used actions and activities, and displays them in Jump Lists under the Recent and Frequent folders (shown below) so you can easily “jump” to these items with a click of a button. The lists are automatically populated based on how frequently and how recently items have been used. For example, if the last activity you opened or used in a task was a VMWare Guest - Delete file activity, the next time you open Task Builder, this activity will be contained in your Recent folder, and possibly, your Frequent folder as well, depending on how often you used the activity in the past. By default, the Recent and Frequent folders are populated with up to 10 entries.
Task Builder makes it easy to clear your recent and/or popular list of actions/activities or clear all Jump List items altogether.

**NOTE:** Clearing Jump List items removes items from your Recent and Frequent lists. It doesn’t delete the items from your computer permanently.

**Finding & Organizing Actions**

There are over 400 available activities that you can use to create the steps of your task. This may seem overwhelming to some individuals, especially new or novice users. Therefore, the Task Builder provides a variety of ways to assist in easily finding actions or activities you are searching for. In addition, it keeps track of a user’s favorite, most often and most recently used actions/activities and places them in the My Actions panel for quick and easy access.

**Finding Actions**

The bottom portion of the Actions panel contains an intuitive search dialog that enables you to define search criteria and provide more efficient and effective searches. It supports dynamic search filtering, allowing you to enter all or part of an action’s name and view only actions that contain matching text. The list is updated as you type each letter.

**To search for an action or activity**

1. Click inside the search dialog box or press CTRL+ SHIFT + F to automatically place the cursor inside the dialog box.
2. Type all or part of the action name you want to search for. The list of relevant actions narrow as you
type each letter.

3. Use the UP or DOWN arrows to navigate to the desired action then press ENTER or click and drag
the action to the Steps pane.

4. To clear the search, click "X" located on the right side of the search dialog or press ESC.

**Modifying Actions List**

By default, actions are categorized in ascending alphabetical order and activities (or sub-actions) are
categorized in a logical sequence. Alternatively, actions can be viewed in descending alphabetical order or they
can be expanded to reveal all activities.

**To change the sort order of the available actions folders**

- Right-click anywhere in the side panel and select **Sort A to Z**.

**To expand all actions**

- Right-click anywhere in the side panel and select **Expand All**.

**To Collapse all actions**

- Right-click anywhere in the side panel and select **Collapse All**.

**My Actions View**

The array of structured actions can automate virtually any type of business process one can think of.
Nonetheless, most developers find themselves using just a handful of activities specific to the needs of their
business or department. The My Actions view allows easy access to such activities. That way, you don’t have to
search through numerous actions or type a keyword to find an activity. Simply select the activity from the
appropriate list in the My Actions panel.

**Favorites**

Favorites can be used as a placeholder for a collection of preferred activities. Favorites can be created and
managed by the user themselves. When an activity is defined as a favorite, a copy of that activity is added to
the Favorites list. That way, users don’t need to spend time searching through the long list of available
activities. They can simply select the activity from their Favorites list. Users can also create folders to
consolidate common activities into specific categories, forming a more organized development environment.

**To add Actions or activities to Favorites**

1. In the Task Builder’s **Available Actions** panel, navigate to the action or activity you wish to add to
your favorites (only one action/activity can be selected at a time).

2. Drag the action/activity onto **Favorites** or any existing folder/sub-folder under **Favorites** or right-
click the action/activity and select **Add to Favorites**.

   OR
1. In the Task Builder’s **Steps** panel, highlight the step(s) that contains the activity you wish to add to your favorites. To select more than one step, hold down CTRL during selection.

2. Drag the step(s) onto **Favorites** or any existing folder/sub-folder under **Favorites** or right-click the step(s) and select **Add to Favorites**.

**To add A Folder/Sub-folder to Favorites**

1. Right-click **Favorites** and select **Create Folder**. A new folder is added to your favorites list.

2. Rename the newly created folder to the desired name.

3. Add sub-folders under the newly created folder by right-clicking that folder and selecting **Create Folder**.

4. Rename the newly created sub-folder to the desired name.

**Snippets**

For those who not only use specific activities but ones that contain commonly used properties or settings, such activities can be saved as snippets. Snippets provide an easy way to implement frequently used code into a task. Instead of re-entering the properties for a specific activity or series of activities every time they’re added as task steps, users can save the steps as a snippet and simply drag and drop the snippet wherever it is needed in any task. A snippet can consist of one or more steps. Once a snippet is added to a task, its contents act as normal steps (e.g., they can be reorganized or deleted or their properties can be modified as needed). By using folders, users can easily organize common or related snippets into categories, creating a cleaner development environment.

**To add steps to Snippets**

1. Highlight the desired step(s) from the **Steps** panel. To select more than one step, hold down CTRL during selection. Note that selected steps don’t need to be sequential.

2. Drag the step(s) onto **Snippets** or any existing folder/sub-folder under **Snippets** or right-click the step(s) and select **Add to Snippets**.

**To add A Folder/Sub-folder to Snippets**

1. Right-click **Snippets** and select **Create Folder**. A new folder is added to your favorites list.

2. Rename the newly created folder to the desired name.

3. Add sub-folders under the newly created folder by right-clicking that folder and selecting **Create Folder**.

4. Rename the newly created sub-folder to the desired name.
**Action Properties**

A task is one of the primary elements in structure and design. A task is essentially a series of steps that are assembled sequentially to perform an operation or complete a certain goal. A major point of task construction is selecting the appropriate actions/activities to include as task steps. This can be accomplished within the Task Builder in a number of ways:

- Dragging the action/activity from the *Actions panel* and dropping it into the desired line in the *Steps panel*.
- Double-clicking the desired action/activity.
- Right-clicking the action/activity and from the context menu, select the following:
  - Add step - Automatically places the action at the end of the task.
  - Insert step - Inserts the action directly above the currently highlighted step.

During this process, a dialog window appears titled Action Properties (also known as the Action Editor) displaying the selected activity’s available properties which are categorized into different tabs. The goal for action editor interface design is to make it easier for developers to access the range of properties and parameters supported for each activity as well as reduce redundancy. As a result, the new action editor is redesigned to provide easier and quicker ways to edit a task by focusing on the properties that are of most importance. The left portion of the editor now contains the full list of consolidated activities common to a given action. This enables jumping from one activity to another without the need to re-enter required parameters or re-open the action editor. Below illustrates the main sections and elements of a typical action editor interface.
Depending on the action being chosen, certain parameters must be entered with valid data in order to properly add the action to the Steps panel. After the properties are set and the proper parameters assigned, clicking the OK button saves the settings and closes the properties dialog. The action then becomes a step and its settings are then displayed in the Steps panel.

NOTE: More information regarding the properties and parameters of an action/activity can be found by clicking the Help button located on the bottom left corner of the editor.

Examples

The section below provides examples of how to define workflows in the AWE interface.

- Example Workflows
- Example: Downloading Files from a Remote Server
- Example: Archiving Files
- Excel Actions Example
Sample Workflows

The Advanced Workflow Engine comes with several sample workflows to demonstrate how to create a workflow. The comments in the workflow provide instructions. The sample files are stored in C:\ProgramData\Globalscape\EFT Server Enterprise\AWE. When you create a Site, a copy of the sample file is saved prepended with the Site’s name. For example, on Site “MySite,” Sample - _README FIRST_.aml is saved as MySite_Sample - _README FIRST_.aml.

To view the sample workflows

1. In the EFT administration interface, connect to EFT and click the Server tab.
2. In the left pane, expand the Site node for the Site that you want to configure, then click the Advanced Workflows node. The node expands to show the Sample Workflows.
3. In the left pane, click a sample workflow. The right pane displays the properties of the selected workflow.
4. Do one of the following to open the workflow in the Task Builder:
   - In the right pane, click Edit.
   - In the left pane, double-click the workflow.
5. View the comments in the Steps pane for instructions on how to configure the workflow. Use this guidance to create similar workflows.
6. If you want to save the sample workflow with your changes, click Save and Close. The .aml is saved and prepended with the Site name. (e.g., MySite_Sample - Ping)

If you have accidentally overwritten a sample workflow and you want to revert to the Site’s version of the sample file, you can copy the original in the default location and save it with the Site’s name, as described above. You can also create an Event Rule to periodically back up the folder that contains your custom workflow.
Example: Downloading Files from a Remote Server

The following workflow will download files from a folder on a remote FTP server and send an e-mail that they were downloaded.

To move a file

1. In the EFT administration interface, connect to EFT and click the Server tab.
2. In the left pane, expand the Site node for the Site that you want to configure, then click the Advanced Workflows node.
3. In the left pane, click the Advanced Workflows node. The right pane displays details of defined workflows.
4. In the right pane, click New. The Create a Workflow dialog box appears.

5. In the What do you want to call this workflow box, specify a name for the workflow. When you add the workflow to Event Rules, the name you specify here appears in the Rule.
6. (Optional) Provide a description of the workflow, then click OK. The Task Builder interface appears.
7. The tree in the left pane lists the steps that you can add to the workflow. The right pane displays the steps in the workflow. In the Available Actions tree, expand the Network and FTP nodes. The list of FTP Actions appears.
8. Drag the **FTP Log on** Action to the steps pane. The **FTP Log On** dialog box appears.

   Provide the connection information, then click **OK**. (If you need a certificate and/or connection through a proxy, specify that information also.)

9. Click **FTP Advanced**. The **FTP Advanced** options appear. Specify any required FTP server commands and parameters, then click **OK**.

10. Click **Download file(s)**. The **FTP Download** options appear.

11. In the **Remote File** box, specify the file to download. For example, to download all files, type `*`.

12. Click the folder icon to open the **FTP Browser** dialog box. In the **FTP Browser** dialog box, you can connect to the FTP Server and view the file list, which will allow you to specify a particular directory, as well as verify that your connection information is correct.

13. In the **Local Files** box, specify the destination at which to save the files. (EFT must have permission to write to the folder.) If you type a folder that does not exist, EFT will create it.

14. Click **OK** to close the **FTP Download** dialog box.

15. Drag **Email > Send Message** to the steps pane. The **Send E-Mail** dialog box appears.

16. Define the **To**, **From**, **Subject**, and **Body** of the message, then click **OK**.

17. Click **Save and Close** to save the workflow. The workflow is now ready to be used in Event Rules. For example, you could create a Timer Rule and add this workflow as an Action to download all of the files from a specific folder on an FTP Server every day at 8 a.m., and send an e-mail notification that the files were downloaded.

**Example: Archiving Files**

This example describes how to create an Advanced Workflow that zips a group of documents together into one file and places it onto a shared drive. The Advanced Workflow can then be inserted into an Event Rule to execute the workflow on a regular schedule.

**To define the workflow and insert it into an Event Rule**

1. Open the AWE interface.

2. Drag the **Compress Files** action to the right pane. The **Compress Files** dialog box appears.

   First, select the files you want to zip.

   - To zip just one file, click the open folder button to the right of the **Files to compress** box. Browse to your My Documents folder, select a document, then click **OK**. The full path to that document appears in the **Files to compress** box.
• To zip multiple files you need to use a wildcard. A wildcard character causes the Server to use any file that matches a mask, replacing an asterisk with any characters required to make the file match. In this example, we want to match any file that ends with .doc or a .txt in our My Documents folder. Start by highlighting all the text that appears after the path to My Documents. In the Files to compress box, type * .doc, and then * .txt. (The asterisk means to match any filename, in this case ending in .doc or .txt.)

3. Click the open folder icon to the right of the Destination file box, and browse to My Documents, name the file (e.g., type archive.zip,) then click Save.

4. Select the text in the Files to compress box, right-click the selection, then click Copy. Copy places the full path to the zip file on the clipboard so that you can use it in the next step.

5. Click OK to save the Compress Files step.

6. When the workflow runs, the Server zips the specified documents into a zip file named archives.zip. Now we need to move it from our My Documents folder and onto the shared drive.

7. In the Available Actions tree, expand the File node, then drag the Move File action to the right pane. The Move File dialog box appears.

   In the Move Files dialog box, you need to specify which file should be moved, and where to move it. Right-click in the Source box, then click Paste. The path to archive.zip appears.

8. In the Destination box, browse to or type the shared drive location (be sure that the Server has write permission to the location) and type what you want to name the file that you are moving, in this case, archive.zip. (You can have it renamed it in the Destination, if you want.)

   Those two simple steps will compress all of the files in your My Documents folder into a .zip file, and then copy the zip file to the shared drive. The workflow is complete, but now you have to insert the workflow into an Event Rule.

9. Click Update And Close to save your changes and close the interface. The workflow appears in the Advanced Workflows node and in the right pane.

10. (Optional) In the Advanced Options area, select the Terminate the process check box and specify the number of seconds after which to terminate the workflow if it fails to execute.

11. (Optional) Specify the level of debug logging in the Debug log level box, None, Minimal, Normal, or Verbose (None is the default). When you are logged in to the EFT computer, you can click View log folder to view the logs created by this workflow.
Finally, create the Scheduler (Timer) Rule in the EFT administration interface to execute the workflow on a schedule. The completed Event Rule should look similar to the illustration below.

**Example: Using Excel Actions**

AWE’s Excel actions use Microsoft’s Excel automation engine to perform their work. Therefore, Microsoft Excel must be licensed and installed on the EFT computer for these actions to work. Excel objects run in the context of the user executing them. If EFT is running as a Local System, then the Excel object is executed using the Local System profile. Before using an Excel action in Event Rules, refer to [Globalscape Knowledgebase article #11089](#) for important information.

This example demonstrates the use of various Excel actions:

```xml
<AMEXCELCREATEWORKBOOK WORKBOOK="c:samploes.xls" OVERWRITE="YES" />
<AMEXCELADDWORKSHEET WORKSHEETNAME="My New Sheet" LASTWORKSHEET="YES" />
<AMEXCELAUTOMATICWORKSHEET WORKSHEET="My New Sheet" />
<AMEXCELSETCELL NEWVALUE="Set this data here" CELROW="1" CELCOLUMN="1" />
<AMPAUSE ACTION="waitfor" SCALAR="3" />
<AMEXCELSETCELL NEWVALUE="and this here" CELROW="2" CELCOLUMN="1" />
<AMPAUSE ACTION="waitfor" SCALAR="3" />
<AMEXCELCLOSEWORKBOOK SAVE="NO" />
```

Below is a description of each step of the task, including the action used and details about that action:

1. **Create Workbook Action** - Create new Excel workbook “C:\Temp\sampledoc.xls.” Overwrite file if it already exists.
2. **Add Worksheet Action** - Add new Excel worksheet “My New Sheet” to workbook open in session “ExcelSess1.” New worksheet will be the last one of the workbook.
3. **Activate Worksheet Action** - Activate Excel worksheet "My New Sheet" in workbook open in session "ExcelSess1."

4. **Set Cell Action** - Write text "Set this data here" in Excel cell located at row:1 and column:1.

5. **Pause Task Action** - Pause task for 3 seconds.

6. **Set Cell Action** - Write text "and this here" in Excel cell located at row:2 and column:1.

7. **Pause Task Action** - Pause task for 3 seconds.


**AML (Markup Language)**

This section is a reference for the Automate Markup Language.

- Introduction to AML
- Arrays
- Constants
- Datasets
- Expressions
- Functions/Extended Functions
- Variables
- Shared Arrays
- Shared Variables

AML (Automation Markup Language)

AML (Automation Markup Language) is the primary internal language. It is a markup language similar to HTML or XML and defines a set of rules for encoding the structure, layout and contents of tasks. AML is powerful and scalable, yet its format is designed to emphasize simplicity and usability, allowing data to be easily interpreted and shared amongst experienced programmers and novices alike. AML is comprised of a variety of available actions/activities and hundreds of functions and instructions with built-in support for variables, constants and expressions.

Do not let the fact that there is an underlying "language" scare you away. The Task Builder component provides an easy to use visual interface layer over the top of AML. It is not necessary to learn the actual internal syntax of any of these commands; this information is documented for advanced users that wish to familiarize themselves with AML. .
Actions (also known as steps or tags), are objects that encapsulate common things that a user may need to in Windows. These include running a program, sending keystrokes to an application, clicking a button, downloading a file, running a database query and much more!

BASIC Scripting encompasses all available functions which may be used either as expressions inside any task step by surrounding the function in % signs, or inside a BASIC script by using the BASIC Script action. **It is not necessary to use BASIC scripting to use AML.** BASIC scripting is only available to extend the built-in capabilities of AML.

The code below displays a simple task as it would appear in AML format. It is comprised of 3 steps:

A comment step which can be used to describe what the task will do.

Creates a variable with the value “Hello World!”

Displays the variable’s value in a message dialog.

```AML
<!-- This is an optional comment. It can be used to describe this task. -->
<AMVARIABLE NAME="Name_of_Variable" VALUE="Hello World!" DESCRIPTION="Description of Variable" />
<AMSHOWDIALOG MESSAGE="Hello World!" WINDOWTITLE="Message Box Title" POSITION="lower_left" ICON="information" />
```

**Arrays**

Like variables, arrays are used to represent data in a task that may be different each time a task runs. But unlike standard variables, arrays can contain multiple rows and optionally multiple columns. You may want to think of an array as nothing more than a list, such as a shopping list, to-do list, birthday list, etc. All items in an array, like the items in a list, have a position somewhere between first and last. Items are numbered from lowest to highest in arrays, therefore, they are accessed by number. For example, to retrieve the element in Row 2, Column 10, the following expression would be entered:

```
%arrayName(2,10)%
```

**One & Two Dimensional Arrays**

An array can be used to store a list of data read from a text file or other system containing customer data. For instance, if there are 10 customers, the array would need to have 10 rows. If the data consisted of first name, last name, and company name it would need 3 columns and would need to be a two dimensional array. An array can also be a simple list of text or numeric values (e.g., multiple rows of data, only one column), this is called a one dimensional array.

An array with only one dimension is linear. In other words, it contains a list of data that can be referenced by a number. For example if my one dimensional array named myArray had 3 values, then the syntax would be:

```
myArray(1) = value1
myArray(2) = value2
myArray(3) = value3
```
In a two dimensional array, you have both rows and columns, such as a spreadsheet. You would then need to reference a cell by row and column, like the expression below which references row 1, column 5.

 [%Myarray(1,5)%]

Creating & Setting Arrays

The **Array** action contains individual activities that allow you to create or modify an array. To create an array, use the Create array activity. In the properties of this activity, you give your array a name, choose whether you want your array to be a one, two, or three dimensional array and optionally, set its values. After creating an array, you can set its values in any task step using the **Set array** activity or you can re-size an array using the **Resize array** activity.

Examples

**NOTE**: The sample AML code below can be copied and pasted directly into the Steps panel of the Task Builder.

Sample 1

This is a simple task that associates each array with a color.

```aml
<AMARRAY NAME="myarray" TYPE="TEXT" ROWS="3" />
<AMSET VARIABLENAME="myarray(1)">Red</AMSET>
<AMSET VARIABLENAME="myarray(2)">Blue</AMSET>
<AMSET VARIABLENAME="myarray(3)">Green</AMSET>
<AMSHOWDIALOG>%myarray(1)%
%myarray(2)%
%myarray(3)%</AMSHOWDIALOG>
```

Sample 2

This is a more complex task that uses functions and performs loops (requires a folder called C:\test\ with a few files in it).

```aml
<AMVARIABLE NAME="thefilename"></AMVARIABLE>
<AMVARIABLE NAME="counter"></AMVARIABLE>
<AMARRAY NAME="myarray" TYPE="TEXT" ROWS="%FileCount('c:\test\')%" />
<AMLOOP TYPE="FOLDER" FOLDER="c:\test\"
RESULTVARIABLE="thefilename">
    <AMINCREMENTVARIABLE RESULTVARIABLE="counter" />
    <AMSET VARIABLENAME="myarray(%counter%)">%thefilename%</AMSET>
</AMLOOP>
<AMLOOP TOTALLOOPS="%UBound(myarray, 1)%"
RESULTVARIABLE="counter">
    <AMSHOWDIALOG>%myarray(counter)%</AMSHOWDIALOG>
</AMLOOP>
```
**Shared Arrays**

Like variables, arrays are used to represent data in a task that may be different each time a task runs. But unlike standard variables, arrays can contain multiple rows and optionally, multiple columns. In BPA Server, arrays have the ability to become "shared", which means they are accessible by any objects contained in a given workflow. Unlike normal arrays that are created and shared within a single task, shared arrays are created and declared at the workflow level. This informs the workflow that its contents can be read by other objects or modified by other tasks that reside in the workflow.

Once an array is created at the workflow level, it is declared a shared array, making it accessible by any object contained in that workflow as long as its name is enclosed with percentage signs (e.g., %arrayName%). Unlike a normal array, a shared array can be accessed within any task step without the need to first create it. In addition, a shared array can be passed from the parent workflow to a sub-workflow and vice versa.

**Creating Shared Arrays**

In order for an array to be declared as a shared array, it must be added to the *Shared Arrays* tab located in the Output/Shared Objects panel of the Workflow Designer.

**To create a shared array:**

In the Workflow Designer, select the *Arrays* tab.

Right click anywhere inside the *Arrays* panel to open the context menu.

Select *Add Array*, as shown below.

A dialog titled *Array* appears as shown below.

Enter the required information then click *OK* to save and close.
Parameters

The **Array** dialog described above is used to enter required and optional parameters. The available parameters are described below.

**General**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the array that should be shared throughout the workflow.</td>
</tr>
<tr>
<td>Initial Value</td>
<td>An optional value that the array should be set to initially.</td>
</tr>
<tr>
<td>Description</td>
<td>An optional text description that describes the purpose of the shared array.</td>
</tr>
</tbody>
</table>

**Managing Shared Arrays**

**To delete a shared array:**

On the **Arrays** tab, right-click the array to delete. To select more than one array, hold down CTRL during selection.

From the context menu that appears, select **Delete Array** (to delete a single array) or **Delete Selected Arrays** (to delete all selected arrays).

A prompt appears confirming deletion. Click **Yes** to complete the operation.
NOTE: Make sure that any reference to deleted arrays are removed from sub-workflows or tasks, otherwise an error will occur during execution.

To edit a shared array:

On the **Arrays** tab, right-click the array to edit.

Select **Edit Array** from the context menu that appears.

The **Workflow Array** dialog appears.

Make the appropriate modifications and click **OK** to save changes.

To rename a shared Array:

On the **Arrays** tab, right-click the array to rename.

Select **Rename Array** from the context menu that appears.

Enter a new name for the array.

NOTE: Make sure that associated workflows or tasks point to the new array name, otherwise an error will occur during execution.

### Constants

Like **variables**, constants can be used to represent data in a task. However, unlike variables, constants have a fixed value which cannot be modified during a task’s execution. Constants can represent information that is often referenced within a task or data that may be common to many tasks. This makes information portable and simple to update. The elements of a constant consists of a name and value. A constant’s defined value is global, thus, are available to all tasks on a particular system. When you use a constant in a task step, substitutes its assigned value during execution. When you modify a constant’s value, every occurrence of that value is updated across the system. The Constants properties page (available from **Options -> System Settings -> Constants**) lists all the user-defined constants in the system.

Constants are useful for assigning data that is used often and may change over time, such as user names, file locations, or e-mail addresses. A constant’s name/value is pre-defined for a particular system. For example, you could create a constant named **AdminEmail** and assign the network administrator’s email address as its value. This constant can be referenced in multiple tasks where you wanted to send an e-mail notification to the network administrator. If you wanted to deploy the task to a remote client (or Runtime) you would simply create a constant of the same name at the other location and assign it the e-mail address of the other network administrator. If changes are maid to the e-mail address, you would simply change the value of the constant to match the new e-mail address.
Creating & Defining Constants

Constants are local to each computer and are stored in the local machine settings of the registry in the 8 key. Constants are created and defined via **System -> Options -> Constants** tab of the Task Administrator.

**To create a new constant:**

From the **System** menu in the Task Administrator window, select **Options**.

Click the **Constants** tab.

Click the **Add** button. A dialog titled **Constants** appears.

In the **Constants** dialog box, enter a name and value for the constant. A constant name must begin with a letter, and contain only letters and numbers. You can also enter a comment for reference (the comment is optional and does not affect how the constant behaves).

Click **OK**.

**To remove an existing constant:**

Select the constant from the User defined constants list.

Click **Remove**.

Click **OK**.

**NOTE:** Before removing a constant, be sure you have removed any references to it in your tasks.

**To modify an existing constant:**

Select the constant from the User defined constants list.

Click **Modify**.

In the **Constants** dialog box, modify the name, value and comment relating to the constant.

To disable a constant, clear the **Enable** check box.

Click **OK**.

For complete instructions on how to create, define and modify constants.

Using Constants

Similar to variables, constants may be used in any step parameter by specifying the name of the constant surrounded by percentage (%) signs. Using a constant instead of specifying a value multiple times in a task can simplify code maintenance. Additionally, constants can act to make tasks more portable, as they can easily pass on information to the task at runtime about the current environment.

Tasks may be designed on one machine and run on others - either by exporting and importing or by deploying a task via remote administration. Because of this, sometimes it is necessary to specify a path to a file or any other piece of information that may be different from machine to machine. It is impractical to design a separate task for each machine. For instances such as this, allows the use of constants.
Datasets

What are Datasets?

In the simplest terms, a dataset is any named group of records. Like variables, datasets are used to represent data in a task that may be different each time a task runs. But unlike standard variables, datasets can contain multiple rows and columns. This is useful when retrieving a collection of data, such as a database or spreadsheet, or retrieving information that describes one or more objects or items. Datasets can hold information such as medical or financial records. Datasets are also used to store information returned by applications or the operating system itself. For example, AWE returns information about task errors (e.g., error name, number, description) via an AMError dataset. Datasets can be cataloged, which permits them to be referred to by name without specifying where they are stored.

The data in a dataset is laid out like a database table, which has a unique name and consists of columns and rows. The columns consist of pre-defined units of data, such as one item in a database or personnel data about one member of a customer list. The rows contain the actual data for the columns. An example of a simple dataset containing customer data is illustrated below. The name of the dataset is Customers. The first row (in bold) contains the unique names of the dataset columns, which in this case, describes the data type. All other rows include the actual data as described by each column.

Table: Customers

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Brown</td>
<td><a href="mailto:John.Brown@mydomain.com">John.Brown@mydomain.com</a></td>
<td>626</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>222-2222</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>323</td>
</tr>
<tr>
<td>Steven</td>
<td>Goldfish</td>
<td><a href="mailto:goldfish@fishhere.net">goldfish@fishhere.net</a></td>
<td>455-4545</td>
</tr>
<tr>
<td>Paula</td>
<td>Smith</td>
<td><a href="mailto:ps@mycompany.org">ps@mycompany.org</a></td>
<td>416</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>323-8888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>416</td>
</tr>
<tr>
<td>James</td>
<td>May</td>
<td><a href="mailto:jim@supergig.co.uk">jim@supergig.co.uk</a></td>
<td>323-3232</td>
</tr>
</tbody>
</table>
**Using Datasets**

Datasets are accessed in the same way that one would access information in a database, by specifying the column and row where the data resides. Every dataset created and used must have a unique name, much like variables. But because datasets are more like tables (as opposed to variables, which can be thought of more like containers that hold one value), they are referenced differently than other objects. When using datasets, the unique name of the dataset must be referenced followed by the column name enclosed in percentage signs. For instance:

```
%DatasetName.ColumnName%
```

When a dataset is created, the current row is automatically set to 1 (assuming that the dataset has any data, since it is possible for a dataset to have 0 rows, such as when a SQL Query returns no data). A dataset is of minimal use, however, unless one can access the other rows. Typically, this is accomplished by using the **Loop Dataset** action which takes a dataset name as a parameter and automatically increments the current row with each iteration. The loop continues until all the rows have been accessed. In this way, one could make a task that performs operations on each row of the dataset while using the same expression. Using the table above as an example, one can retrieve the email address of each customer using the following expression inside a Loop Dataset action:

```
%Customers.Email%
```

Using a Loop Dataset step is not the only way to access dataset rows. It is possible to directly access a particular row of a dataset by supplying the row number within the expression. For example, if the dataset contains five rows and you need to get the data in row 2, simply enter the row number enclosed in parenthesis directly after the dataset name. For example:

```
%DatasetName(2).ColumnName%
```

Again, using the table illustrated above, if you want to retrieve the phone number specified in row 3 (in this case, Paula Smith’s phone number), the following expression will do the trick:

```
%Customers(3).Phone%
```

**Common Actions that use Datasets**

Several actions and activities create and populate datasets. The table below describes some of those actions.

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Query</td>
<td>Queries a database and populates a dataset with the data retrieved.</td>
</tr>
<tr>
<td>Get Email</td>
<td>Retrieves one or more messages from the specified server and populates a dataset with the results.</td>
</tr>
<tr>
<td>SNMP Get</td>
<td>Populates a dataset with the data sent by the agent.</td>
</tr>
</tbody>
</table>
Pre-Named Datasets

In addition to datasets that are created and populated during execution of an action or activity, a collection of pre-named (or fixed field) datasets are also available which can provide more insight about the behavior of a running task, determine system states or examine other elements, such as triggers used to fire off a task or errors that were generated by a task. The table below describes some of these types of datasets.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMError</td>
<td>Determine specific characteristics about the task error that occurred, including error code or number, error description, action/activity that generated the error, and other values.</td>
</tr>
</tbody>
</table>

Example using SQL Query action

The SQL Query action is an example of an action that creates and populates a dataset. The fields contained within that dataset are determined by the query that was executed. For example if the following query is executed:

```
SELECT firstname, lastname, company from customer where city='Los Angeles';
```

Then the following dataset would be generated:

```
datasetname
|-- --firstname
|-- --lastname
|-- --company
```

A record (row) is created for each record (row) that is retrieved from the server. To access this data use the Loop Dataset action to loop through the records. Inside the loop you can extract the data from the field of your choice (from the current record) by using an embedded expression such as the one that follows:

```
%mydatasetname.firstname%
```
or you could combine two fields together like this:

```
%mydatasetname.firstname + '' + mydatasetname.lastname%
```

Expressions such as these can be used in any parameter in any action. The AML code to display the data in a message box would look like this:

```
<AMMESSAGEBOX MESSAGETEXT="%mydatasetname.firstname%" WINDOWTITLE="The firstname of the current record is"> 
```

At runtime the text %mydatasetname.firstname% is replaced by the contents of the subject of the current record.

The percent signs (%) at the beginning and end of the variable name indicates that the text in-between the percent signs is an expression and should not be taken literally. Instead, it is replaced with the current contents of that column in the current row at runtime.
Common Dataset Fields

Most of the fields (columns) that are returned in a dataset are dictated by the action creating and populating the dataset. For example, when using the SQL Query action, the field names are controlled by the columns returned by the query. In the SNMP Get action, the field names are dependant on the SNMP query data being returned by the agent. Nonetheless, a common set of fields are available which can be used to retrieve information about any created dataset. Most of these fields are global, which means they can be used on any dataset, regardless of which action or activity the dataset originated from. These fields can be accessed in the same manner as other fields, though some of their values are read only. Also, the values of these fields are the same regardless of the row being accessed.

The table below describes the common set of fields (columns) that a dataset creates (assuming the dataset name assigned is theDataset).

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>theDataset.CurrentRow</td>
<td>Number</td>
<td>The current row that will be accessed in the dataset by an expression that does not contain a specific row index.</td>
</tr>
<tr>
<td>theDataset.TotalRows</td>
<td>Number</td>
<td>The total number of rows in the dataset</td>
</tr>
<tr>
<td>theDataset.TotalColumns</td>
<td>Number</td>
<td>The total number of columns (not including the static columns) in the dataset.</td>
</tr>
<tr>
<td>theDataset.ExecutionDate</td>
<td>Date</td>
<td>The date and time the dataset was created and populated</td>
</tr>
<tr>
<td>theDataset.RowsAffected</td>
<td>Number</td>
<td>The number of rows affected by an update.</td>
</tr>
<tr>
<td>theDataset.SQLQuery</td>
<td>Text</td>
<td>The SQL Query that was used to generate this dataset (If a SQL Query was not used, this value is empty).</td>
</tr>
<tr>
<td>theDataset.Datasource</td>
<td>Text</td>
<td>The data source used for the SQL Query, if applicable.</td>
</tr>
<tr>
<td>theDataset.ColumnNames</td>
<td>Text</td>
<td>A comma-delimited list of the column names in the dataset</td>
</tr>
</tbody>
</table>

Expressions

An expression, in programming, is a combination of symbols that represents a value. The expression is interpreted according to the rules of the programming language and a value is returned. The Expression Builder provides a quick and convenient way to create and evaluate BASIC expressions from directly within the parameters of a task step. Expressions work by taking the text found between percentage (%) signs and passing it to the BASIC expression interpreter. During runtime, the BASIC interpreter replaces the original expression (including the percentage signs) and returns the expression’s resulting value instead.
Using Expressions

Many actions that normally require expressions may already be built into activities. However, you can use expressions to further expand an action’s capabilities. For instance, you can build expressions that resolve complex, dynamic data at runtime with the use of variables, functions, extended functions and operators.

Example 1 - Simple mathematical expression

In the following example, a task runs a single step that performs a simple mathematical calculation and displays the results in a message box. To create this task:

- In the Task Builder, add a Message Dialog activity to the Steps panel.

- From the General properties of this activity, enter %7+7% in the Message to display parameter (as shown below) and click OK to save changes.

- From the Task Builder’s ribbon, click the Run button to start the task.
Since the phrase 7+7 is surrounded in percentage signs, knows that it should try to resolve the expression and not display it literally. To resolve the expression, at runtime, the value between the percentage signs is passed to the BASIC expression interpreter, where it is processed and the result is returned. A message dialog should appear with the value of 14 (as shown below).

Example 2 - Expression with variable

Expressions can also contain variables. Taking the previous example a step further, assume we now want the task to give the user the ability to enter a number he/she would like squared.
Use the Create variable activity to create a variable named NUMTOSQUARE with the initial value of 1. This variable will be used during runtime to retain the user’s response to the question.

Use the Input dialog activity as the second step to display the question **What number would you like squared?** and populate the NUMTOSQUARE variable with the answer.

Use a Message Dialog as the last step which will display the result by use of an expression multiplying the variable by itself. `%NUMTOSQUARE * NUMTOSQUARE%`.

The complete task is displayed below in AML format. For convenience, the AML code can simply be copied and pasted directly into the Steps panel of the Task Builder.

```aml
<AMVARIABLE NAME="NUMTOSQUARE">1</AMVARIABLE>
<AMINPUTBOX RESULTVARIABLE="NUMTOSQUARE">Which number would you like squared?</AMINPUTBOX>
<AMMESSAGEBOX>%NUMTOSQUARE * NUMTOSQUARE%</AMMESSAGEBOX>
```

**Notes**

Notice that we first had to create the variable we intend to use. All variables to be passed into a scripting language, either explicitly through the use of a BASIC script step or implicitly through the use an expression, need to be created first. You can accomplish this using the Create Variable action.

Variable names are not case sensitive; the variables are capitalized here for clarity.

Because each expression is passed to the scripting language engine, any command available from a common BASIC script is available to you for use in expressions in your step parameters. Refer to BASIC Scripting for details on these powerful commands.

**Functions & Extended Functions**

In computer programming, a function is predefined code which can generate various kinds of values depending on certain input from the user. Most programming languages such as VBScript come with a pre-written, built-in range of functions that perform various procedures or routines. Functions can be used as an expression in any task step, thus, adding more intelligence to a task and further broadening the functionality. Additionally, contains an extensive collection of additional functions that increase the functionality of the Scripting Engine which are called extended functions.
Function Syntax

Most functions require other parameters in order to properly complete a procedure or routine. The general format of a function is its name followed by any arguments contained in between parenthesis. An argument (sometimes referred to as parameters) is a value sent to the function when it is called upon. By passing an argument to a function, the function is given information to work on. A basic function would look like this:

\[
\text{FunctionName}(\text{arguments})
\]

Certain functions may not need arguments to properly perform its duty. A function without arguments must include an empty set of parentheses (). For example, the `Date()` function returns the current system date and the `Now()` function returns the current date along with the time. These functions require no arguments or parameters in which they need to perform any procedures on, therefore the proper syntax are simply the functions themselves, with nothing entered in between the parenthesis. For instance, if the following syntax was entered:

\[
\text{Date()}
\]

The returned result would be the current date value such as 1/1/2010.

An example of a function which requires arguments is the `Len()` function, which returns the number of characters in a string. The syntax for this function is `Len(“string”)` which requires a string to be entered inside the parenthesis in which it will perform calculations on. For example, if this was entered:

\[
\text{Len(“Hello”)}
\]

The returned value would be 5.

Another example is the extended function, `ExtractFileName()`, which requires a path and filename in between the parenthesis in which it will extract the file name from. So, if the following was entered:

\[
\text{ExtractFileName(“c:\foldername\filename.txt”)}
\]

The returned value would be `filename.txt`.

An example of a function requiring more elaborate arguments is the `Left()` function, which returns a specified number of characters from the left side of a string. The syntax for this function is `Left(“string”, length)` which requires a valid string followed by the amount of characters to return inside the parenthesis. If the following was entered:

\[
\text{Left(“AutoMate”, 4)}
\]

The return value would be `Auto`.

Another example is the `InStr()` function which returns the position of the first occurrence of one string within another. The search begins at the first character of the string. The syntax for this function is `InStr(“string1”, “String2”)`. So, if the following was entered:

\[
\text{InStr(“Hello”, “o”)}
\]

The return value would be 5 signifying that the letter “o” is the fifth character in the word “Hello”.


Using Functions

Built-in VBScript functions along with extended functions can be used to return data inside any action parameter of a task that accepts expressions by surrounding the function with percent signs. This tells that the contents in between the percent signs should not be taken literally, but used as an expression instead. For example, the `Len()` function can be used within Message dialog activity by simply entering the following syntax in the Message to display parameter (highlighted below):

```
%Len("Hello")%
```

As previously mentioned, the `Len()` function returns the number of characters in a string. When the task runs, the `Len()` function will be performed and a message dialog displaying the results will appear, as illustrated below.
Expressions such as variables, constants or other functions can be used as arguments or parameters entered inside the parenthesis of a function. In such cases, the specified expression needs to be entered by itself (omitting any percent signs, quotes or other characters). For example, assume that a variable named \%theFile\% is populated with the string value c:\folderName\fileName.txt. In order to extract only the filename from this string and view the results in a message dialog, the proper syntax to enter in a Message Dialog step would be:

\%ExtractFileName(theFile)\% 

As shown below.
During task execution, a message dialog will display the properly extracted filename as shown below.

Using Expression Builder

The Expression Builder is a valuable tool used to assist in the creation and examination of expressions. When using the Expression Builder, help regarding each function can be accessed by first selecting the Functions folder from the lower left pane, then selecting the desired function from the lower right pane and pressing the F1 key or by right-clicking the function and selecting Help from the pop-up menu that appears.

See Expression Builder for more details.
Variables

A variable (also known as a local variable) is a placeholder for varying or changeable data. Variables play an important role in because they enable developers to write flexible tasks. Rather than entering data directly into a task step, a developer can use variables to represent the data. Then, when the task runs, the variables are replaced with real data. This makes it possible for a task to perform actions on values that may or may not be known until runtime. It also allows a single task the ability to hold different sets of data.

Variables are created and/or set in Task Builder during task creation. They are commonly used when a task involves collecting data from a source and then performing some action on it. You use the variable to contain the collected data, and then set up the actions to be performed on the data by referencing the variable. For example you could create a variable in the beginning of a task that will be populated with the user’s input in a form or message box. In a subsequent step you could perform calculations on the data the user entered by referencing the variable.

Creating Variables

The Create variable activity generates a local variable which can be used to store dynamic values for utilization in any step of the task or any sub-tasks started with the Start Task action. The developer can enter a value to initially populate the variable during creation or the value field can be left blank. Instead, variables can be set with a value using the Set Variable action, which adds or changes the contents of an already existing variable. Certain available actions that support populating variables can also set or modify a variable’s contents, such as the Input Box action. This action displays an input box allowing the user to enter a value which is saved to the variable specified.

Once a variable is created, it becomes available for use in subsequent steps of the task. It will appear on drop-down lists in places where a variable can be entered, and it can be used in expressions by simply placing the name of the variable between percent signs (%). For example, entering %UserInput% tells to populate the expression with the current value of the variable named UserInput.

It is important to note that in order for an variable to be used in a task step, it must initially be created within an earlier step.

Variable Naming Conventions

Variable names must contain only alphanumeric characters, must start with a letter and cannot contain spaces. Variable names are not case-sensitive. When choosing a variable name, it is good practice to select a name that is descriptive of what the variable holds. For example, if a variable holds the size of a shoe, then name it ShoeSize or TheSize. This makes the task more comprehensible. Also, be sure to avoid using BASIC keywords, functions, or instructions. Names such as DATE or TIME would create a conflict. One way to avoid this is to include distinguishing characters in the variable name such as VAR, THE or MY. For example, a date variable could be named MyDate, DateVar or theDate.
Variable names must be unique within a task, but can be repeated from one task to the next. For example, if you create a variable in one task named UserInput, you can create a variable of the same name in another task without any conflict occurring.

**Variable naming restrictions**

There are limitations to the names that can be given to variables. These limitations derive from the fact that when the expressions between % signs are evaluated as a script, even if they are a simple variable name. Therefore all BASIC Script keywords, functions names, and operators are forbidden as variables names (e.g. while, wait, like, instr, date, time, now, daysinyear, etc.).

Also all Windows environment variables such as PATH, DATE, TMP, PROMPT are also reserved and cannot be declared as new variables. They are, however, accessible and readable, whether or not they were declared “as parameters” in an AMVARIABLE statement.

This is quite a lot of common names which are in fact reserved. Nothing warns that they are not allowed, but the error will appear at execution time, with some strange error message that is all but obvious to understand. A wise precaution to avoid any mistake is to initiate all variable names by a specific (group of) character(s), say v_ (underscore) which then allows variables like v_wait, v_do, v_while, v_path without any ambiguity.

**Scoping & Accessibility**

Local variables created within a task function (e.g., by using Create Variable, Create Array, etc.) are scoped to that function. Local variables, therefore, are not visible and thus cannot be used outside the function in which they are created. If a variable is created within a function with the same name as a task variable, the local variable “hides” the task variable and takes precedence.

Scoping a created variable to the function in which it is created has the following advantages:

- **Variables are only created when required** - Because functions are distinct units designed for a single purpose, variables created within them are typically only needed to serve that purpose before returning an end result. Scoping the variables to the function keeps these variables localized to where they are needed. Once their purpose is complete, they are discarded.

- **Keeps temporary variables where they are needed** - Many functions make use of temporary variables, for example when indexing an array, looping, etc. Local variables provide a means to create indexing or temporary variables that are only available within that function without creating a global variable that can be trampled on by other functions. An example of the danger of using a common index variable without local scoping is creating an index variable named i that, in a loop, calls another function which itself uses a variable i as an index. When the second function is finished, the first function will have the wrong value! Local variables avoid this problem.

- **It does not preclude the use of task variables** - A function can still set and retrieve the value of a task variable, or assign a task variable the value of a local variable. This improves readability and keeps the number of variables a user needs to remember or debug to a minimum.
**Makes tasks easier to debug** - The use of local variables and task variables means there are less variables the user needs to keep in their head at any given time. “The less you have to keep in mind, the smaller the chance that you’ll make an error because you forgot one of the many details you needed to remember.” Additionally, it provides a means to separate variables in a debugger or other visual display (see Design Time Implementation).

**Allows recursion** - Recursion is impossible without local variables. Recursion relies on local variables to store the state of the function during each call. If a variable is scoped outside the function, each call of that function will overwrite the data of the previous call, thus rendering the recursion useless.

A local variable’s or local array’s accessibility is defined by the **Variable is private** option on the Create Variable or Create Array step. By default, this option is set to OFF and therefore the variable’s default accessibility is public. This means that an external task can access a public local variable.

---

**Sub-tasks**

Sub-tasks are task files run synchronously at the step level using the Start Task action. Sub-tasks are treated as individual modules that maintain their own scope and accessibility. Sub-tasks can access public variables, functions and extended functions of the parent task, but not vice versa. Functions and variables contained in the sub-task, however, whether private or public, are accessible to functions called within the sub-task while the sub-task is executing, exactly as if the sub-task were running independently. An important point to note regarding this is how events (which are functions that are optionally implemented by a task and called implicitly by the task engine) are scoped; a sub-task’s events are fired during sub-task execution, not the parent’s. This follows the rules of variable and function scoping and accessibility.

---

**Shared Variables**

In BPA Server, variables take on a much larger role with their ability to be “shared” amongst other objects within a given workflow. Similar to a variable used inside a task, a shared variable is used as a storage mechanism to store and share dynamic data during execution. However, unlike a task variable that’s created and accessible only within the task level, a shared variable is created and declared at the workflow level, thus, its value can be recognized by other workflow objects (e.g., result arrow, evaluation object, sub-workflow, etc.) and its contents can be set or modified within any task that exists in that workflow.

Once a variable is created at the workflow level, it is declared a shared variable making it accessible by any object contained in that workflow as long as its name is enclosed with percentage signs (e.g., %variableName%). Unlike a normal variable, a shared variable can be accessed within any task step without the use of a Create variable activity. In addition, a shared variable can be passed from a parent workflow to a child workflow (or sub-workflow) and vice versa.

The ability to define shared variables within a workflow provides an abundance of new functionality. Processes that were once limited to one task running on one computer can now be fully integrated into a multi-machine environment.
Declaring shared variables

In order for a variable to be declared as “shared”, it must be added to the Variables tab located in the Output/Shared Objects panel of the Workflow Designer.

To create a shared variable:

From the Workflow Designer, select the Variables tab.

Right click anywhere inside the panel to display the context menu. Select the Add Variable option (as shown below, circled in red).

A dialog titled Variable appears, as shown below.

Enter the desired settings under the General and Advanced tab and click OK to save changes and close the dialog box.
Variable Parameters

The **Variable** dialog described above is used to enter the name of the shared variable, its initial value (if required) and other parameters that dictate its behavior. The available parameters are described below.

**General**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the variable that should be shared throughout the workflow.</td>
</tr>
<tr>
<td>Initial Value</td>
<td>An optional value that the variable should be set to initially. As with all parameters, this value may be literal or an expression (if surrounded by percent % signs).</td>
</tr>
<tr>
<td>Description</td>
<td>An optional text description that describes the purpose of the shared variable.</td>
</tr>
</tbody>
</table>

**Advanced**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable type</td>
<td>Causes the variable to assume a specific type. The available options are:</td>
</tr>
<tr>
<td></td>
<td><strong>Auto (default)</strong> - The variable will auto-detect whether it is populated with a number or text. The variable will adapt to the proper type when possible depending on the operation being performed.</td>
</tr>
</tbody>
</table>
### Property | Description
--- | ---
Text | The variable will always be treated as text (by default) regardless of its contents. If an operation is attempted that is only valid for numbers, an error will be generated.
Number | The variable will always be treated as a number regardless of its contents. If an operation is attempted that is only valid for numbers, an error will be generated.

### Passing variables to/from sub-workflows

Dictates how the shared variable should react when being passed from the parent workflow to a sub-workflow and vice versa. The available options are:

**Pass the value from the parent workflow to this variable when this workflow starts, overriding this variable’s initial value** - If enabled, the value of the shared variable that resides in the parent workflow is allowed to be passed to any embedded workflow (or sub-workflow). The value passed from the parent workflow will override any initial value that the sub-workflow is set to. If disabled, the value of the shared variable is only available to the parent workflow and will not be passed to any sub-workflow (disabled by default).

**Pass the value from this variable to the parent workflow when this workflow finishes** - If enabled, the value of a shared variable that resides in the sub-workflow is allowed to propagate back to the parent workflow upon completion. If disabled, the shared variable’s value will not be passed to the parent workflow (disabled by default).

**NOTE**: Only shared variables with the same name are passed between parent and sub-workflows and vice versa (when the settings permit).

---

### Managing Shared Variables

The Workflow Designer’s Variables tab is ideal in managing and debugging shared variables. Here, you can add, edit, rename or delete shared variables and examine variable content, particularly its initial and current value, during runtime without inserting additional steps to output the values. Furthermore, you can insert breakpoints at certain points in your workflow to halt execution in order to view the contents of a shared variable to determine if the current value has been properly set or modified by a specific task.

**To delete a shared variable:**
- From the WFD Variables tab, right-click the variable to delete. To select more than one variable, hold down CTRL during selection then right-click the highlighted variables.
- From the context menu that appears, select Delete Variable (if deleting a single variable) or Delete Selected Variables (if deleting multiple variables).
- A prompt appears confirming deletion. Click YES to complete the operation.
NOTE: Make sure that any reference of deleted shared variables are removed from sub-workflows or tasks, otherwise an error will occur during execution.

To edit a shared variable:

From the WFD Variables tab, right-click the variable to edit.

Select Edit Variable from the context menu that appears.

Make the appropriate modifications and click OK to save changes.

To rename a shared variable:

From the WFD Variables tab, right-click the variable to rename.

Select Rename Variable from the context menu that appears.

Enter a new name for the variable.

Parameters

The Variables tab generates information divided into seven columns:

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the shared variable.</td>
</tr>
<tr>
<td>Type</td>
<td>The type of shared variable as set in the variable’s properties (Auto, Text or Number).</td>
</tr>
<tr>
<td>Initial Value</td>
<td>Displays the initial value of the shared variable as set in the variable’s properties. If no value was set, this field is blank.</td>
</tr>
<tr>
<td>Current Value</td>
<td>Displays the current value of the shared variable during workflow execution. The value is updated after each workflow object executes.</td>
</tr>
<tr>
<td>Description</td>
<td>A description of the shared variable as set in the variable’s properties. The description does not affect variable performance at runtime.</td>
</tr>
<tr>
<td>Access</td>
<td>Public - The variable or function is visible and accessible to external tasks. Private - The variable or function is not visible or accessible to external tasks.</td>
</tr>
<tr>
<td>Scope</td>
<td>Local - Specifies that the variable is local to the current context or scope. Usually, this means the procedure or function you are currently executing. Task - Specifies that the variable is a task variable that is considered global to the entire task.</td>
</tr>
</tbody>
</table>

Context Menu

Right-click a shared variable or an empty region inside the Variables tab to view the following menu items.
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Variable</td>
<td>Adds a shared variable to the current workflow.</td>
</tr>
<tr>
<td>Edit Variable</td>
<td>Opens the properties dialog of the shared variable, allowing you to modify its contents.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies the shared variable and its properties.</td>
</tr>
<tr>
<td>Paste</td>
<td>Pastes the shared variable and its properties.</td>
</tr>
<tr>
<td>Rename Variable</td>
<td>Renames a shared variable.</td>
</tr>
<tr>
<td>Delete Variable</td>
<td>Deletes the selected shared variable.</td>
</tr>
<tr>
<td>Delete Selected Variables</td>
<td>Deletes the selected shared variables. To select more than one variable, hold down CTRL during selection.</td>
</tr>
</tbody>
</table>
BASIC Scripting

The BASIC Scripting engine is designed to extend the functionality available in the primary AML language. AML language is easy to use, drag-and-drop, and very robust; however, there may be occasions when it is necessary to use BASIC scripting to access advanced objects or increase execution time. All functions, instructions, and definitions are designed to be Visual BASIC for Applications (VBA™ Microsoft) compatible. The BASIC language is used for expressions inside AML task steps, and for Multi-line BASIC scripts.

BASIC Expressions

BASIC scripting functions can be used to return data inside of an action parameter by surrounding a valid expression with percent signs. This is called an expression.

Example:

```
%left(varname, 2)%
```

Multi-line BASIC Scripts

A BASIC script may be executed by using the BASIC Script action, which allows creation of multi-line BASIC compatible scripts.

When to use BASIC scripts:

- To access the Windows API
- To access COM objects, OLE objects, ActiveX objects
- When fast execution time is a priority
- Advanced File/Read/Write/Seek operations
- Advanced text parsing

For details of the BASIC scripting language, refer to https://help.globalscape.com/help/awe10/Basic_Scripting/Introduction_to_Scripting.htm in the AWE online help.

BASIC Script IDE

The BASIC Script action encompasses a BASIC Script IDE, an interactive design environment used for developing, examining and testing BASIC scripts. To access the BASIC Script IDE, click the Edit Script button located in the properties of the BASIC Script action.

The Basic Script IDE consists of the following interfaces:

- **Main IDE (Interactive Design Environment)** - The main IDE (shown below) is the primary interface where a macro or module can be created, loaded from disk or edited. Other interfaces can be accessed from the main IDE.

- **References Dialog** - The References dialog displays the current references of a macro or module. For more details, see References Dialog.
**User Dialog Editor** - The User Dialog Editor can be used to create custom dialog boxes for use with this activity. For more details, see User Dialog Editor.

**References Dialog**

The References dialog (displayed below) is one of three sub-interfaces contained in the BASIC Script IDE. This dialog shows the current macro/module’s references. References to type libraries may be added (by checking the corresponding check-boxes) or removed (by unchecked the corresponding check-boxes) and the relative priority can be modified. Checked references are available to the current macro/module. Each checked reference is searched in order from top to bottom. A checked reference’s name can be changed using the Name text box. Priority of each reference can be modified by selecting the desired reference then clicking the Up/Down Arrow. To access the References Dialog from the main BASIC Script IDE, select Edit -> References...
References Overview

Adding a reference to a BASIC Script allows the selected COM objects properties and methods to be accessed without an explicit call to CreateObject. To Access available References, from within the BASIC scripting IDE, select Edit | References. The following dialog will be displayed.
The References dialog shows the current BASIC script references. References to type libraries may be added (checked) or removed (unchecked) and the relative priority can be changed. Checked references are available to the current macro/module. Each checked reference is searched in order from top to bottom.

**UserDialog Editor**

The UserDialog Editor is one of three sub-interfaces contained in the BASIC Script IDE. This dialog can be used to create custom dialog boxes for use in the task. A UserDialog is described by a Begin Dialog...End Dialog block. To graphically edit a UserDialog, place the current selection in a UserDialog block and click Edit -> UserDialog or click the Edit UserDialog button located in the main IDE’s toolbar. Doing so will open the UserDialog Editor (shown below). Use the provided toolbar buttons to create the custom dialog.

**NOTE:** Hovering the cursor over a particular button will display a description of that button.

![UserDialog Editor](image)

**BASIC Scripts & Custom Functions**

The library of actions and activities offers a robust set of functionality that can be started automatically and configured visually. However, during certain occasions, it may be necessary to apply BASIC scripting to perform complex operations, such as accessing API calls or COM objects, implementing server-side scripting, and much more. Solution is its full-featured Visual Basic for Applications compliant scripting language combined with a built-in BASIC scripting engine.
The scripting engine integrates seamlessly into the existing framework, appearing as an extra IDE in the macro builder called ‘Run a Script’. In an automated procedure, you may have an unlimited number of ‘script’ steps, making decisions, handling complex File I/O routines, etc. The added flexibility of the BASIC Scripting Language is that an unlimited number of customized oriented commands can be added to the language to make life (and automation) easier for you.

The built-in BASIC scripting engine can further enhance scripts by fully automating them. Scripts may be embedded directly into a task file, or alternatively, may reference an external BAS file. See BASIC Script action for more details. An example of this type of usage is shown below.

**NOTE**: The sample AML code below can be copied and pasted directly into the Steps panel of the Task Builder.

```aml
<AMVARIABLE NAME="NUMTOSQUARE" VALUE="1" />
<AMVARIABLE NAME="theresult" />
<AMSHOWDIALOG ACTIVITY="input" MESSAGE="Please enter a number that you would like squared." RESULTVARIABLE="NUMTOSQUARE" />
<AMSCRIPT>Sub Main
    theresult = NUMTOSQUARE * NUMTOSQUARE
    theresult = "The result is" & Str(theresult)
End Sub</AMSCRIPT>
<AMSHOWDIALOG RESULTVARIABLE="NUMTOSQUARE">%theresult%.</AMSHOWDIALOG>
```

### Custom Functions

BASIC scripting can also be used for creating custom functions. A custom function performs a calculation, and may be used as many times as needed as an expression in future steps. In fact, the above task would be more efficient if created in this manner.

An example of the same task restructured with a custom function is shown below:

```aml
<AMVARIABLE NAME="NUMTOSQUARE" VALUE="1" />
<AMVARIABLE NAME="theresult" />
<AMSHOWDIALOG ACTIVITY="input" MESSAGE="Please enter a number that you would like squared." RESULTVARIABLE="NUMTOSQUARE" WINDOWTITLE="Please enter a number" ICON="information" />
<AMSCRIPT>Function SquareNumber(thenumber)
    SquareNumber = thenumber * thenumber
End Function</AMSCRIPT>
<AMSHOWDIALOG>The result is %SquareNumber(NUMTOSQUARE)%</AMSHOWDIALOG>
<AMSHOWDIALOG>The result of the squared number squared is %SquareNumber(SquareNumber(NUMTOSQUARE))%</AMSHOWDIALOG>
```

As you can see, the basic script was converted into a custom function so that it can be called multiple times in the task and so that different values may be passed to it for processing. The function will of course yield a different result depending on the value passed to it.
Regular Expressions

Several activities, notably Find text and Replace text, allows the use of "regular expressions" (abbreviated regex or regexp). Regular expressions are powerful notations that allow a wide range of text searches to be performed using formulas specific to string manipulation. This article describes the regular expression syntax.

Fundamentals

**Match anywhere** - By default, a regular expression matches a substring anywhere inside the string to be searched. For example, the regular expression abc matches abc123, 123abc, and 123abcxyz. To require the match to occur only at the beginning or end, use an anchor.

**Escaped characters** - Most characters like abc123 can be used literally inside a regular expression. However, the characters `\.*?+[]\{[^\]}` must be preceded by a backslash to be seen as literal. For example, `\.` is a literal period and `\` is a literal backslash. Escaping can be avoided by using `\Q...\E`. For example: `\QLiteral Text\E`.

**Case-sensitive** - By default, regular expressions are case-sensitive. This can be changed via the "i" option. For example, the pattern `\)$abc` searches for "abc" without regard to case. See options for other modifiers.

Common Syntax & symbols

<table>
<thead>
<tr>
<th>Syntax/Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>A dot or period matches <em>any single</em> character (except newline: <code>\r</code> and <code>\n</code>). For example, ab. matches abc and abz and ab. .</td>
</tr>
<tr>
<td>*</td>
<td>An asterisk matches zero or more of the preceding character, class, or sub-pattern. For example, a* matches ab and aaab. It also matches at the very beginning of any string that contains no &quot;a&quot; at all. <strong>Wildcard:</strong> The dot-star pattern .* is one of the most permissive because it matches zero or more occurrences of any character (except newline: <code>\r</code> and <code>\n</code>). For example, abc.*123 matches abcAnything123 as well as abc123.</td>
</tr>
<tr>
<td>?</td>
<td>A question mark matches zero or one of the preceding character, class, or sub-pattern. Think of this as &quot;the preceding item is optional&quot;. For example, color?r matches both color and colour because the &quot;u&quot; is optional.</td>
</tr>
<tr>
<td>+</td>
<td>A plus sign matches one or more of the preceding character, class, or sub-pattern. For example a+ matches ab and aaab. But unlike a* and a?, the pattern a+ does not match at the beginning of strings that lack an &quot;a&quot; character.</td>
</tr>
<tr>
<td>(min,max)</td>
<td>Matches between <em>min</em> and <em>max</em> occurrences of the preceding character, class, or sub-pattern. For example, a{1,2} matches ab but only the first two a’s in aaab. Also, {3} means exactly 3 occurrences, and {3,} means 3 or more occurrences. <strong>NOTE:</strong> The specified numbers must be less than 65536, and the first must be less than or equal to the second.</td>
</tr>
</tbody>
</table>
### Syntax/Symbol | Description
--- | ---
Classes of characters: The square brackets enclose a list or range of characters (or both). For example, `[abc]` means "any single character that is either a, b or c". Using a dash in between creates a range; for example, `[a-z]` means "any single character that is between lowercase a and z (inclusive)". Lists and ranges may be combined; for example `[a-zA-Z0-9_]` means "any single character that is alphanumeric or underscore". A character class may be followed by *, ?, +, or (min,max). For example, `[0-9]`+ matches one or more occurrence of any digit; thus it matches xyz123 but not abcyxz.
The following POSIX named sets are also supported via the form `[:xxx:]`, where `xxx` is one of the following words: alnum, alpha, ascii (0-127), blank (space or tab), cntrl (control character), digit (0-9), xdigit (hex digit), print, graph (print excluding space), punct, lower, upper, space (whitespace), word (same as `\w`). Within a character class, characters do not need to be escaped except when they have special meaning inside a class (e.g., `[^a]`, `[a\-b]`, `[a\]`), and `[\a]`).
[^...] Matches any single character that is not in the class. For example, `[^/]*` matches zero or more occurrences of any character that is not a forward-slash, such as http://. Similarly, `[^0-9xyz]` matches any single character that isn’t a digit and isn’t the letter x, y, or z.
\d Matches any single digit (equivalent to the class `[0-9]`). Conversely, capital `\D` means "any non-digit". This and the other two below can also be used inside a class; for example, `\d.-` means "any single digit, period, or minus sign".
\s Matches any single whitespace character, mainly space, tab, and newline (`r` and `\n`). Conversely, capital `\S` means "any non-whitespace character".
\w Matches any single "word" character, namely alphanumeric or underscore. This is equivalent to `[a-zA-Z0-9_]`. Conversely, capital `\W` means "any non-word character".
Circumflex (^
) and dollar sign ($) are called anchors because they don’t consume any characters; instead, they tie the pattern to the beginning or end of the string being searched.
^ may appear at the beginning of a pattern to require the match to occur at the very beginning of a line. For example, `^abc` matches abc123 but not 123abc.
$ may appear at the end of a pattern to require the match to occur at the very end of a line. For example, `abc$` matches 123abc but not abc123.
The two anchors may be combined. For example, `^abc$` matches only abc (e.g., there must be no other characters before or after it).
If the text being searched contains multiple lines, the anchors can be made to apply to each line rather than the text as a whole by means of the "m" option. For example, `m)^abc$` matches 123 `r`nabc `r`n789. But without the “m” option, it wouldn’t match.
<table>
<thead>
<tr>
<th>Syntax/Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>\b means “word boundary”, which is like an anchor because it doesn’t consume any characters. It requires the current character’s status as a word character (\w) to be the opposite of the previous character’s. It is typically used to avoid accidentally matching a word that appears inside some other word. For example, \bcat\b doesn’t match catfish, but it matches cat regardless of what punctuation and whitespace surrounds it. Capital \B is the opposite: it requires that the current character not be at a word boundary.</td>
</tr>
<tr>
<td></td>
<td>The vertical bar separates two or more alternatives. A match occurs if any of the alternatives is satisfied. For example, gray</td>
</tr>
</tbody>
</table>
| | Items enclosed in parentheses are most commonly used to: 

   - Determine the order of evaluation. For example, 
     (Sun|Mon|Tues|Wednes|Thurs|Fri|Satur)day matches the name of any day. 
   - Apply *, ?, +, or \{min,max\} to a series of characters rather than just one. For example, (abc)+ matches one or more occurrences of the string “abc”; thus it matches abcabc123 but not ab123 or bc123. 
   - Capture a sub-pattern such as the dot-star in abc(.*)xyz. For example, 
     RegExMatch() stores the substring that matches each sub-pattern in its output array. Similarly, RegExReplace() allows the substring that matches each sub-pattern to be reinserted into the result via backreferences like $1. To use the parentheses without the side-effect of capturing a sub-pattern, specify ?: as the first two characters inside the parentheses; for example: (?:.*). 
   - Change options on-the-fly. For example, (?im) turns on the case-insensitive and multiline options for the remainder of the pattern (or sub-pattern if it occurs inside a sub-pattern). Conversely, (?-im) would turn them both off. All options are supported except DPS\r\n\a. |
# Index

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Creating Tasks</td>
<td>75</td>
</tr>
<tr>
<td>Adding a Comment</td>
<td>51</td>
</tr>
<tr>
<td>Adding Task Steps</td>
<td>79</td>
</tr>
<tr>
<td>Advanced Workflow Engine</td>
<td>7</td>
</tr>
<tr>
<td>AMError</td>
<td>117</td>
</tr>
<tr>
<td>AMError.ErrorDesc</td>
<td>117</td>
</tr>
<tr>
<td>AMError.ErrorNumber</td>
<td>118</td>
</tr>
<tr>
<td>AMError.FunctionName</td>
<td>118</td>
</tr>
<tr>
<td>AMError.IPAddress</td>
<td>118</td>
</tr>
<tr>
<td>AMError.MachineName</td>
<td>119</td>
</tr>
<tr>
<td>AMError.ParentTaskName</td>
<td>119</td>
</tr>
<tr>
<td>AMError.StepNumber</td>
<td>120</td>
</tr>
<tr>
<td>AMError.StepText</td>
<td>120</td>
</tr>
<tr>
<td>AMError.Tagline</td>
<td>120</td>
</tr>
<tr>
<td>AML (Automation Markup Language)</td>
<td>140</td>
</tr>
<tr>
<td>Arrays</td>
<td>141</td>
</tr>
<tr>
<td>Attachments</td>
<td>105</td>
</tr>
<tr>
<td>Auditing AWE Workflows</td>
<td>107</td>
</tr>
<tr>
<td>Backing Up Workflows</td>
<td>107</td>
</tr>
<tr>
<td>BASIC Scripts &amp; Custom Functions</td>
<td>170</td>
</tr>
<tr>
<td>Bookmarks</td>
<td>108</td>
</tr>
<tr>
<td>Breakpoints</td>
<td>37, 50</td>
</tr>
<tr>
<td>Comments</td>
<td>51</td>
</tr>
<tr>
<td>Concept</td>
<td>8</td>
</tr>
<tr>
<td>Constants</td>
<td>145</td>
</tr>
<tr>
<td>Copying and Re-using Task Steps</td>
<td>109</td>
</tr>
<tr>
<td>Creating and Managing Task Steps</td>
<td>75</td>
</tr>
<tr>
<td>Creating Variables</td>
<td>81</td>
</tr>
<tr>
<td>Creating Workflows for use in Event Rules</td>
<td>75</td>
</tr>
<tr>
<td>Custom Step Description</td>
<td>97</td>
</tr>
<tr>
<td>Datasets</td>
<td>147</td>
</tr>
<tr>
<td>Debug Panel - Attachments</td>
<td>35</td>
</tr>
<tr>
<td>Debug Panel - Breakpoints</td>
<td>37</td>
</tr>
<tr>
<td>Debug Panel - Labels</td>
<td>39</td>
</tr>
<tr>
<td>Debug Panel - Output</td>
<td>40</td>
</tr>
<tr>
<td>Debug Panel - Regions</td>
<td>42</td>
</tr>
<tr>
<td>Debug Panel - Stack</td>
<td>43</td>
</tr>
<tr>
<td>Debug Panel - Variables</td>
<td>43</td>
</tr>
<tr>
<td>Debug Panel - Watches</td>
<td>45</td>
</tr>
<tr>
<td>Debugging Tools &amp; Techniques</td>
<td>49</td>
</tr>
<tr>
<td>Deleting Workflows</td>
<td>111</td>
</tr>
<tr>
<td>Editing Task Steps</td>
<td>111</td>
</tr>
<tr>
<td>Error Causes</td>
<td>103</td>
</tr>
<tr>
<td>Error Handling Overview</td>
<td>56</td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>Archiving Files</td>
<td>137</td>
</tr>
<tr>
<td>Downloading Files from a Remote Server</td>
<td>136</td>
</tr>
<tr>
<td>Using Excel Actions</td>
<td>139</td>
</tr>
<tr>
<td>Example Workflows</td>
<td>135</td>
</tr>
<tr>
<td>Examples</td>
<td>134</td>
</tr>
<tr>
<td>Expression Builder</td>
<td>93</td>
</tr>
<tr>
<td>Expressions</td>
<td>150</td>
</tr>
<tr>
<td>Features</td>
<td>21</td>
</tr>
<tr>
<td>File Masks &amp; Wildcards</td>
<td>95</td>
</tr>
<tr>
<td>Find &amp; Replace</td>
<td>115</td>
</tr>
<tr>
<td>Finding &amp; Organizing Actions</td>
<td>130</td>
</tr>
<tr>
<td>Finding Actions</td>
<td>130</td>
</tr>
<tr>
<td>Functions/Extended Functions</td>
<td>153</td>
</tr>
<tr>
<td>Importing Workflows</td>
<td>115</td>
</tr>
<tr>
<td>Installing the Advanced Workflow Engine</td>
<td>23</td>
</tr>
<tr>
<td>Markup Language (AML)</td>
<td>140</td>
</tr>
<tr>
<td>My Actions</td>
<td>126</td>
</tr>
<tr>
<td>On Error Tab</td>
<td>99</td>
</tr>
<tr>
<td>Percent Signs in AWE</td>
<td>85</td>
</tr>
<tr>
<td>Printing or Sending Task Steps</td>
<td>116</td>
</tr>
<tr>
<td>References Dialog</td>
<td>168</td>
</tr>
<tr>
<td>References Overview</td>
<td>169</td>
</tr>
<tr>
<td>Regions</td>
<td>52</td>
</tr>
<tr>
<td>Regular Expressions</td>
<td>171</td>
</tr>
<tr>
<td>Restricted VBA Functions</td>
<td>174</td>
</tr>
<tr>
<td>Setting Action Properties</td>
<td>132</td>
</tr>
<tr>
<td>Setting Custom Step Description</td>
<td>97</td>
</tr>
<tr>
<td>Setting Custom Step Description Options</td>
<td>97</td>
</tr>
<tr>
<td>Setting Debug Toolbar Options</td>
<td>60</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Shared Arrays</td>
<td>143</td>
</tr>
<tr>
<td>Shared Variables</td>
<td>160</td>
</tr>
<tr>
<td>Task Builder - Overview</td>
<td>24</td>
</tr>
<tr>
<td>Task Builder (Interface)</td>
<td>24</td>
</tr>
<tr>
<td>Actions Panel</td>
<td>28</td>
</tr>
<tr>
<td>Status Bar</td>
<td>32</td>
</tr>
<tr>
<td>Steps Panel</td>
<td>30</td>
</tr>
<tr>
<td>Task Builder Debug (Tools)</td>
<td>65</td>
</tr>
<tr>
<td>Attachments</td>
<td>105</td>
</tr>
<tr>
<td>Bookmarks</td>
<td>108</td>
</tr>
<tr>
<td>Breakpoints</td>
<td>50</td>
</tr>
<tr>
<td>Regions</td>
<td>52</td>
</tr>
<tr>
<td>Testing Tasks Using Run Options</td>
<td>46</td>
</tr>
<tr>
<td>Watches</td>
<td>54</td>
</tr>
<tr>
<td>Task Builder Options</td>
<td>62</td>
</tr>
<tr>
<td>Debugger</td>
<td>65</td>
</tr>
<tr>
<td>Fonts</td>
<td>67</td>
</tr>
<tr>
<td>Formatting</td>
<td>69</td>
</tr>
<tr>
<td>General</td>
<td>70</td>
</tr>
<tr>
<td>Toolbar</td>
<td>73</td>
</tr>
<tr>
<td>Task Builder Quick Tour</td>
<td>26</td>
</tr>
<tr>
<td>Task Variables</td>
<td>86</td>
</tr>
<tr>
<td>Testing Tasks Using Run Options</td>
<td>46</td>
</tr>
<tr>
<td>The BASIC Language_ All Groups</td>
<td>170</td>
</tr>
<tr>
<td>User Dialog Editor</td>
<td>170</td>
</tr>
<tr>
<td>Using Advanced Workflows in Event Rules</td>
<td>77</td>
</tr>
<tr>
<td>Using Expression Builder</td>
<td>93</td>
</tr>
<tr>
<td>Using Regular Expressions</td>
<td>88</td>
</tr>
<tr>
<td>Variables</td>
<td>158</td>
</tr>
<tr>
<td>Watches</td>
<td>54</td>
</tr>
<tr>
<td>What's New in AWE</td>
<td>9</td>
</tr>
</tbody>
</table>