NiceLabel Automation Unleashed

Aleš Primožič

2013

NiceLabel Automation Unleashed

Second Edition (20131108)

Aleš Primožič

Short Contents

Detailed Contents

Introduction

Who this book is for

This book is intended for different audiences:

- **Decision makers** that decide whether NiceLabel Automation is worthy investing money in and if it supports features their organization needs.
- **Engineers** that implement solutions that are based on NiceLabel Automation to understand how the product works.
- **People that have used NiceWatch** in the past and would like to know what the difference between the two products is and what they need to be prepared for when they decide to use NiceLabel Automation.
- **Administrators** that need to install and maintain application in different environments, including high availability and performance ones.
- **Support personnel** who try to troubleshoot issues their customers might have .
- **Everybody** interested in knowing more about NiceLabel Automation.

How this book is structured

This book can be separated into 6 different parts:

- **The Basics** describe basic terms you need to know when working with NiceLabel Automation or if you want to understand how the product works. The Basics, NiceLabel Automation Workflow, How NiceLabel Automation Service Works, Installation, Automation Builder and Introducing Automation Manager are the chapters in this part.
- Chapters about variables, triggers, filters and actions represent the part about **Core elements of NiceLabel Automation.**
- **Enterprise features** are described in the following chapters: Integration with NiceLabel Enterprise Print Manager, Licensing, Failover cluster, Load-Balanced Cluster.
- **Comparison with NiceWatch**
- If you are looking for **performance and troubleshooting tips and tricks**, see chapters Things to be aware of, Performance tips and tricks, When things go wrong.
- In the **Appendices** you can read in detail which keyboard shortcuts can be used in Automation Builder and Automation Manager, and what the syntax for NiceCommands, Job command files, CSV command files and XML command files looks like.

THE BASICS

The Basics

Variable

Variable is a facility for storing data. The current value of the variable is the data actually stored in the variable at that specific point in time. In NiceLabel Automation and in NiceLabel variables can be used in many places: as a source of value for action properties, as data source for objects that are used as part of label design, in database filters, in functions and scripts. Variables allow you to design flexible and reusable automation solutions.

Depending on source of variable value there are several different types of variables in NiceLabel and NiceLabel Automation:

- Prompt variable
- Database variable
- System clock (date or time) variable
- Printer clock (date or time) variable
- Global variable
- Pick List
- Internal variable

Prompt variable

Prompt variable is a variable that requires the user to enter or confirm its value before printing. It can also get its value from the filter. In NiceLabel this is usually the most used type of variable. NiceLabel Automation only knows this type of variable.

Database variable

Database variable is a variable that gets its value from a database field.

System clock (date or time) variable

"System clock" variable is a variable that gets its value from the computer system clock. Data format of such variable can only be date or time.

Printer clock (date or time) variable

"Printer clock" variable is similar to "System clock" variable but it gets its value from the thermal printer. If printer, for which label is designed for, does not support such functionality, you will not be able to use it; it is not even available in the list of possible sources for variable.

Global variable

Global variable is a variable that can be used on many different labels. Once it is defined, it is stored outside of the current label so it is available to other labels as well. Its last value is stored even after closing the label file and exiting the application. This comes handy for example when continuing serial numbers from previous printing is required. Global variable is not limited for use on only one label. You can use it on as many labels as you like. Please note, that only one label, using the same global variable, can be printed at a time. When using global variables, they are locked for one label and this prevents more labels to use the same global variable at the same time.

Pick list

When pick list is used as a source for variable you can define a list of values that can be used (no duplicates are allowed). No other values can be used or typed in before printing. This type is useful if you have a small, limited set of possible values that variable can use.

Internal variable

Internal variable gets its value automatically and you cannot modify its properties or values. You can only enable or disable them and connect them with different objects on the label or use them in NiceLabel Automation configuration (e.g. in Python scripts).

Action

Actions let you automate tasks and build workflows that accomplish operations quickly and efficiently. NiceLabel Automation comes with dozens of actions you can use.

Trigger

A trigger is a central entity of NiceLabel Automation and is automatically executed in response to certain events. NiceLabel Automation can respond to six different events:

- A change was made to a file or a file was added to a folder
- A change was made to a database
- Data was received on a serial port
- Data was received over a TCP/IP port
- Data was received over a port using HTTP protocol
- A call was made to a web service

Filter

A filter is a module in NiceLabel Automation that processes data that comes into the application via triggers and only extracts data that is useful for the user. NiceLabel Automation can process structured text like CSV files, unstructured data, and XML formatted data.

Configuration

Configuration is a file where triggers, data filter definitions, variables and actions are stored. You can identify it in Windows Explorer by the .misx extension and its icon.

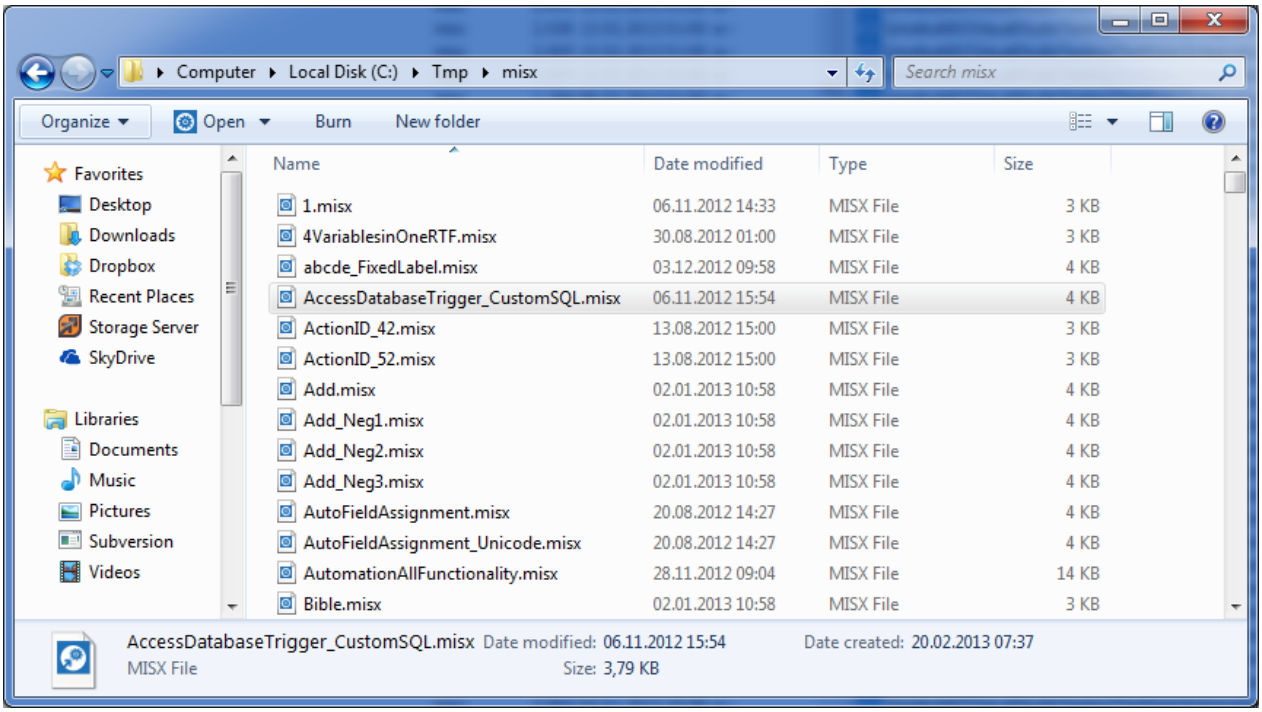


Figure 1 - Configuration Files in Windows Explorer

Print Job

Print job is a file or a set of files that has been submitted to be printed. Job can have options associated with them such as number of copies and priority.

Special Character

A special character or control character or non-printing character is a code point (a number) in a character set, that does not in itself represent a written symbol. Printing control characters were first used to control the physical mechanism of printers, the earliest output device. Later, control characters were integrated into the stream of data to be printed.

Because special characters do not have written symbols, NiceLabel Automation uses special notation for such characters, for example <FF> for Form Feed, or <CR> for Carriage Return. You can enter them manually, or insert them from a drop down menu, that is available on the right side of every configuration property in Automation Builder that supports such characters.

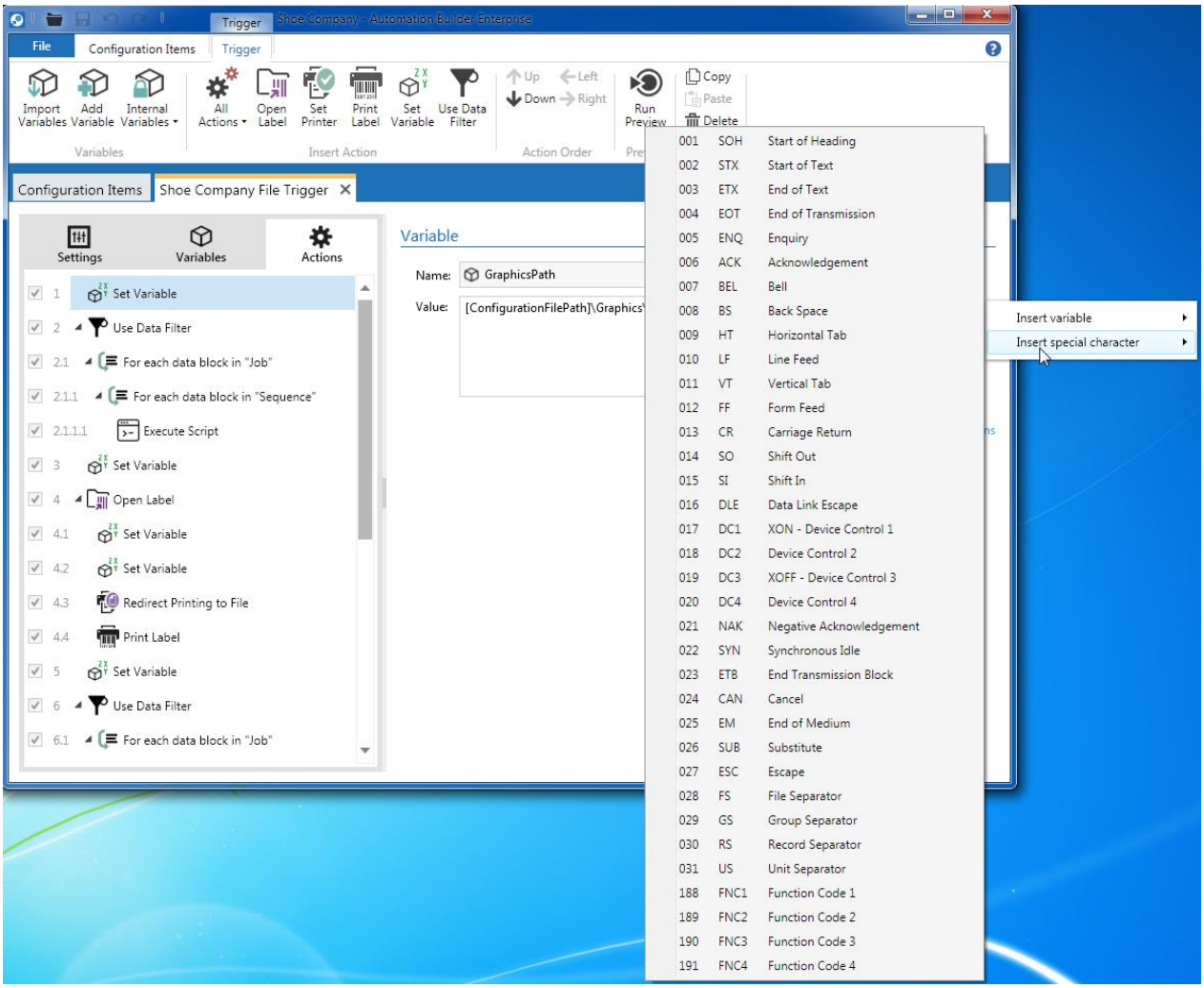
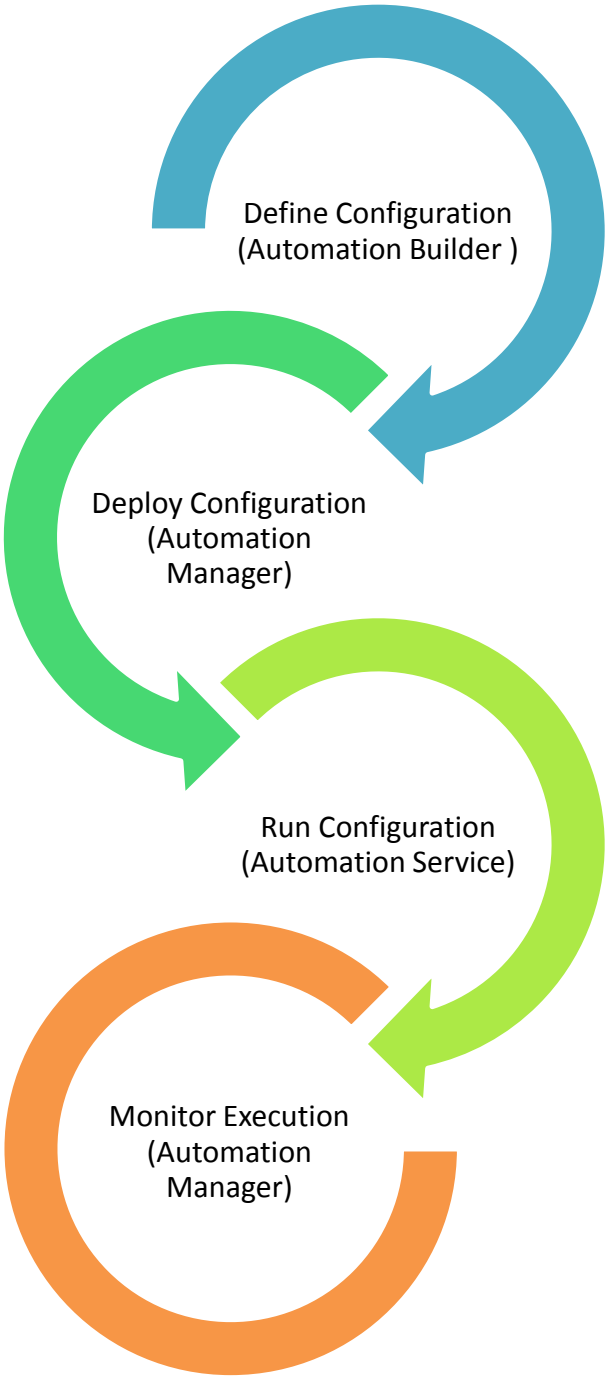


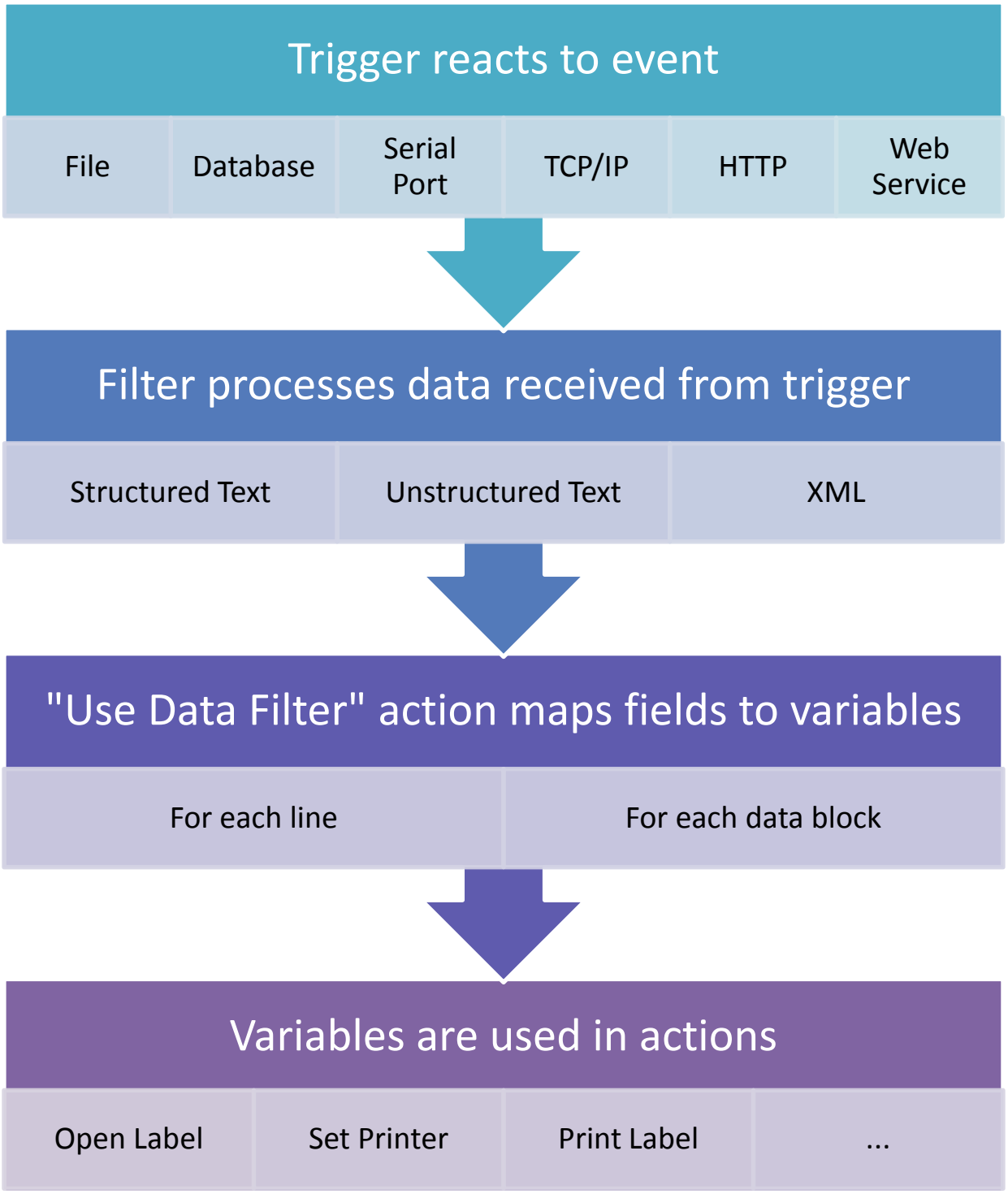
Figure 2 - Automation Builder - Special Characters

NiceLabel Automation Workflow

The whole workflow in NiceLabel Automation is quite simple. First you run Automation Builder to define the configuration by setting triggers, configuring filters and adding actions. Once the configuration is finished, deploy it to Automation Manager, and consequently Automation Service runs it. When the configuration is running, you can monitor trigger execution using Automation Manager.



How NiceLabel Automation Service Works



Once configuration is deployed, Automation Service takes over its execution. Triggers are entry points to the configuration. Their job is to react to events. NiceLabel Automation can react to several different events that are triggered by software (data coming from legacy applications, ERP systems like SAP or Oracle, from databases) or hardware (scales, barcode readers, etc.).

Depending on the trigger type, trigger can read data from a file or a database, from a serial port, it can receive data using TCP/IP and HTTP protocol, or from Web Service. Trigger then stores the data in a temporary data file (path to the file is stored in **DataFileName** variable) and passes it to a filter.

Filter processes the data and transforms it in a shape that can be used by **Use Data Filter** action.

Use Data Filter action maps filter fields to variables.

Variables can then be used in actions, like **Open Label**, **Set Printer**, **Print Label**, **Execute Script**, etc. (there are dozens of actions available). Selected actions are executed for each line or data block in the data file.

Installation

Installation

The whole installation process for NiceLabel Automation is quite straightforward and it can be done in a minute or so. There are only six screens that you have to click through:

- Language Selection
- Welcome Screen
- License Agreement
- Additional Tasks
- Service Configuration
- Done

Language Selection

The first screen you will see when you start the installation procedure is the language selection dialog. Setup process detects the language settings on the computer and uses that information to display the window in the appropriate language. You can see from the screenshot that at the time of the installation my operating system had locales set to German, so the window was displayed in German language.

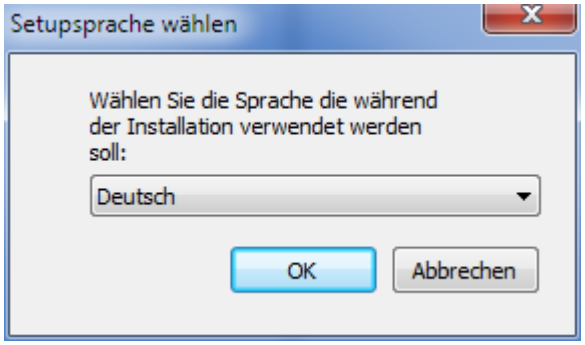


Figure 3 - Installation - Language Selection

NiceLabel Automation is available in 14 languages. When you select the language, the installation will then continue in that language. Also NiceLabel Automation will start in the selected language when you open it for the first time.

Welcome Page

The installation starts with the Welcome Page. All you have to do here is to click the **Next** button (or press the ENTER key).

License Agreement

Today almost every piece of software comes with the end-use license agreement. NiceLabel Automation is no exception, so if you want to use it, you have to accept the terms of the license agreement.

Additional Tasks

By default, NiceLabel Automation is installed in "c:\Program Files (x86)\EuroPlus\NiceLabel Automation" folder. If you want to change the installation path, you can do it by clicking the **Browse…** button or entering the new path in the **Product install path** field.

Additionally you can also choose to have icons created on the desktop or in the quick launch area (this is not available in Windows Vista and newer versions of Microsoft Windows).

Service Configuration

NiceLabel Automation setup installs two Windows services that need access to label files and printer drivers. In order for NiceLabel Automation to successfully access files and printers, you have to tell the installation which Windows user has such privileges. If you want to use files that are stored on a network share, or print to a network printer, make sure you use a user that can access those resources. This means you should avoid using “Local System” account, because it usually cannot access network resources.

If you log in into a domain make sure you include the domain name in the user name (e.g. DOMAIN\USER).

Note

Be careful when installing services under “Local System” account. Make sure you have access to the same resources (like default printer, or network resources) as the user under which you are testing NiceLabel Automation (running preview for example in Automation Builder). If you do not have the same resources available, the two (NiceLabel Automation service and Automation Builder) might behave differently.

Done

When all the files have been copied and services configured, the final screen is shown. Now all you have to do is click the **Finish** button.

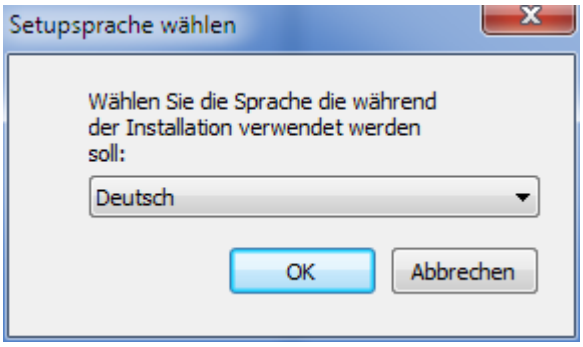


Figure 4 - Installation - Language Selection

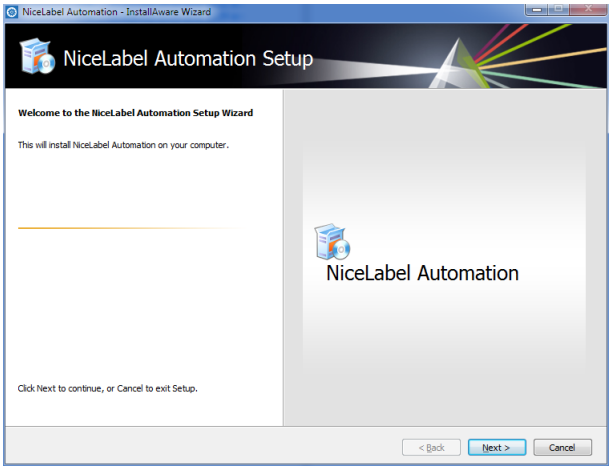


Figure 5 - Installation - Welcome Page

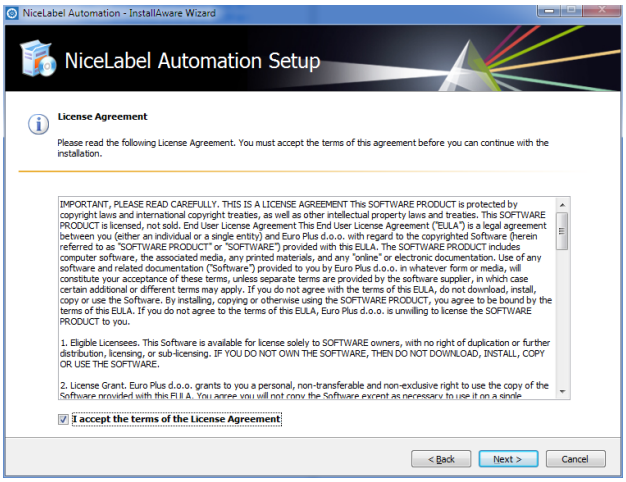


Figure 6 - Installation - License Agreement

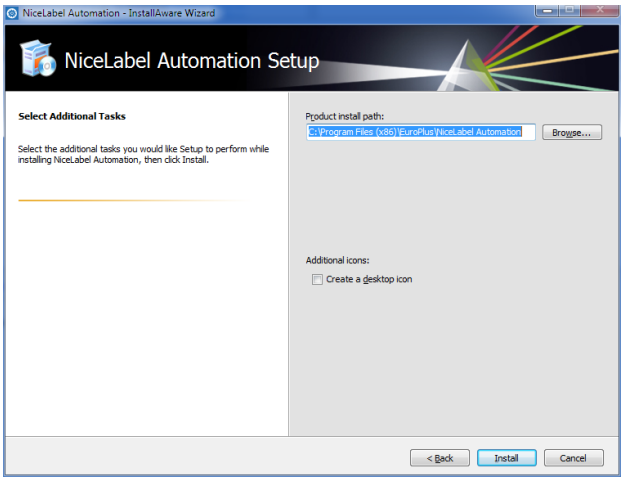


Figure 7 - Installation - Additional Tasks

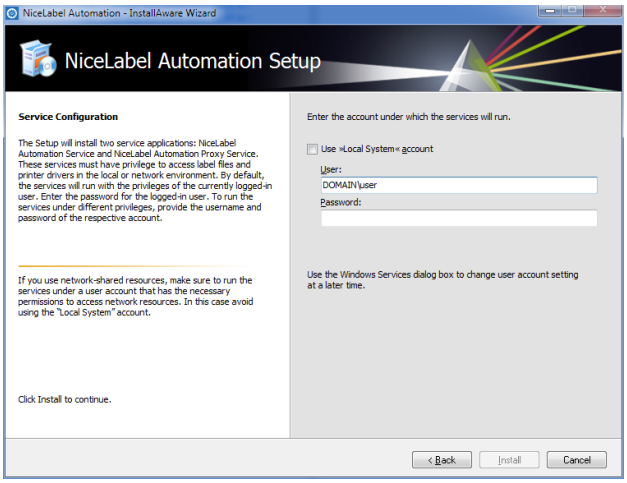


Figure 8 - Installation - Service Configuration

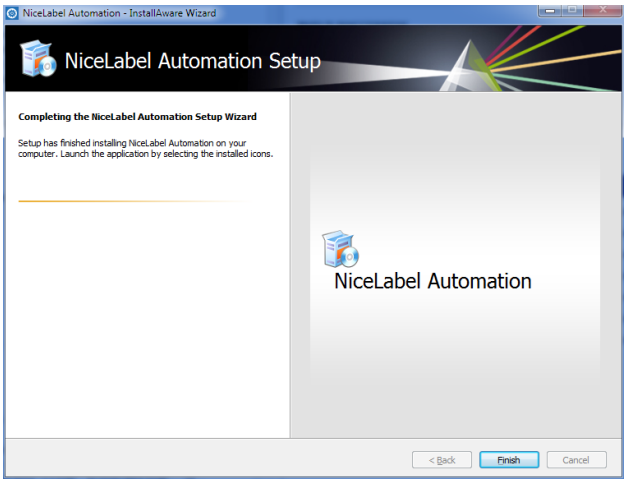


Figure 9 - Installation - Done

What Do You Get?

When installation procedure is completed, you get a new item in the Windows Start Menu with four shortcuts to the installed applications:

- Automation Builder
- Automation Manager
- Automation Configuration
- NicePrintQueue

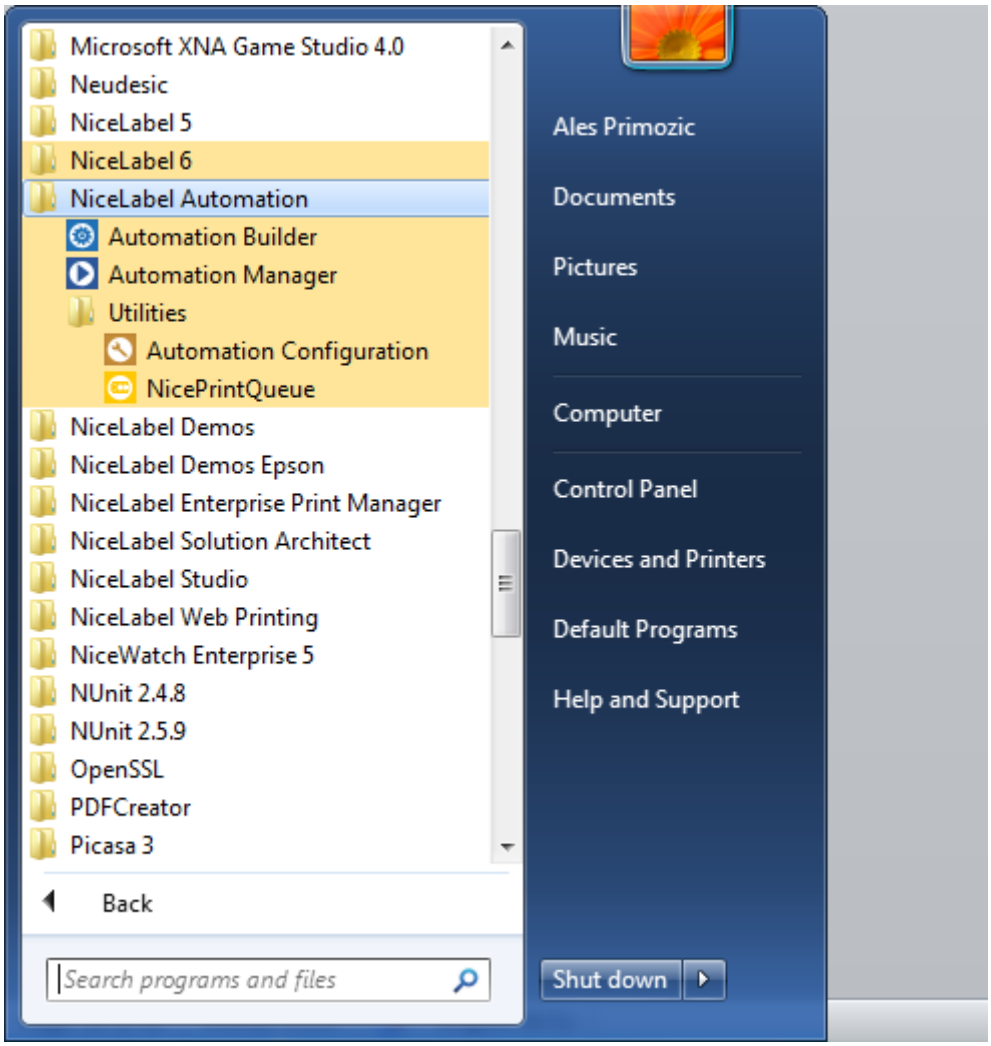


Figure 10 - Windows Start Menu After Installation

Automation Builder

Automation Builder is an application where you build configurations by defining triggers, data filters, variables and actions.

Automation Manager

Use ***Automation Manager*** to manage configurations and triggers.

NicePrintQueue

NicePrintQueue allows you to manage printers and print jobs on your computer. It does not replace Windows Spooler but provides advanced functionality and overview of printer statuses that are not available in the Windows Spooler. For example, some printers allow bi-directional communication with NiceLabel Drivers, which allows you to review printer status in real time (e.g. "out of labels", "out of ribbon", "print head open").

With NicePrintQueue you can:

- view, group and manage printers installed on your computer,
- pause and resume printers ,
- view print jobs for selected printer ,
- pause, resume and restart selected jobs,
- delete jobs for selected printer,
- change order of selected jobs,
- change priorities of selected jobs,
- show detailed printer status (e.g. open printer internal web page, if supported).

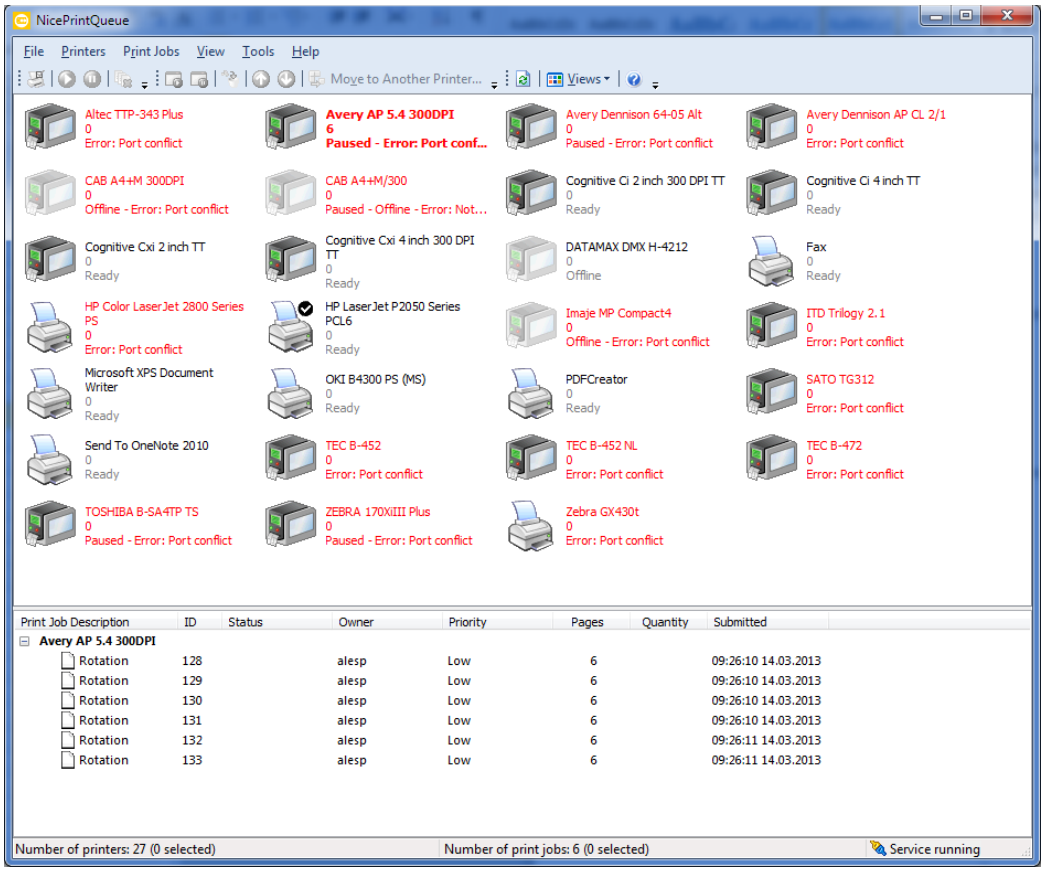


Figure 11 - NicePrintQueue

Automation Configuration

Automation Configuration is a module where you configure NiceLabel Automation. It allows you to

- set default folders,
- select the language NiceLabel Automation applications should run in,
- define passwords that will allow opening protected label designs,
- manage authentication and users,
- configure event and print job monitoring,
- define the NiceLabel Enterprise Print Manager server and settings location,
- set NiceLabel Automation settings,
- enable cluster support.

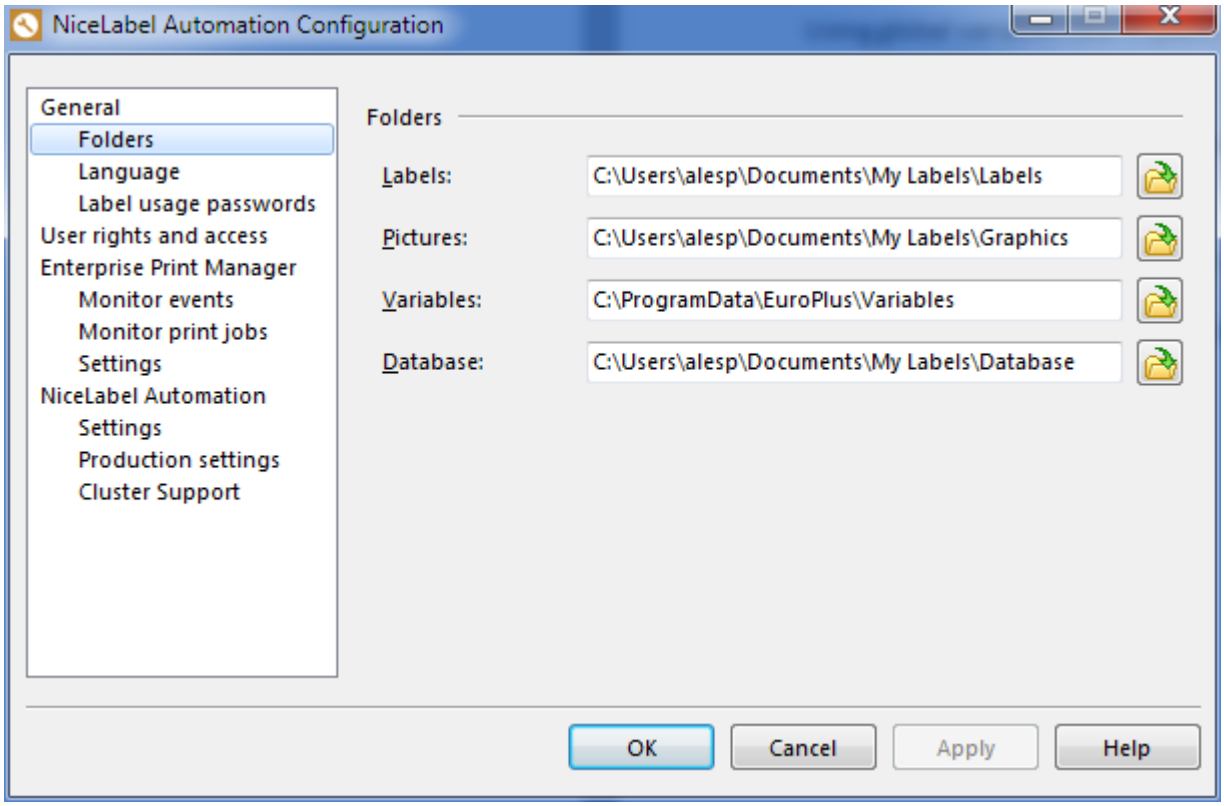


Figure 12 - NiceLabel Automation Configuration

Services

In addition to previously mentioned applications, there are two Windows services installed:

- NiceLabel Automation Proxy Service
- NiceLabel Automation Service

NiceLabel Automation Proxy Service is used primarily for the communication between 32 and 64-bit applications that are part of NiceLabel Automation. One such example of a 64-bit application is NiceLabel Automation Service (on 64-bit operating system only), and Windows Script Host that is used for executing VBScript scripts as 32-bit application.

NiceLabel Automation Service is the work horse of the application as it does most of the work once a configuration is deployed. It runs triggers, processes data using filters and executes actions.

Tip

To start both services from the command line you can use the following commands:

```
net start NiceLabelAutomationProxyService
net start NiceLabelAutomationService
```

If you want to open configuration file when service is started, use

net start `NiceLabelAutomationService FilePath`

For example

net start `NiceLabelAutomationService "e:\Shoe Company\Shoe Company.misx"`

And to stop services use:

```
net stop NiceLabelAutomationProxyService
net stop NiceLabelAutomationService
```

But first make sure that you are running "Command Prompt" with administrative privileges.

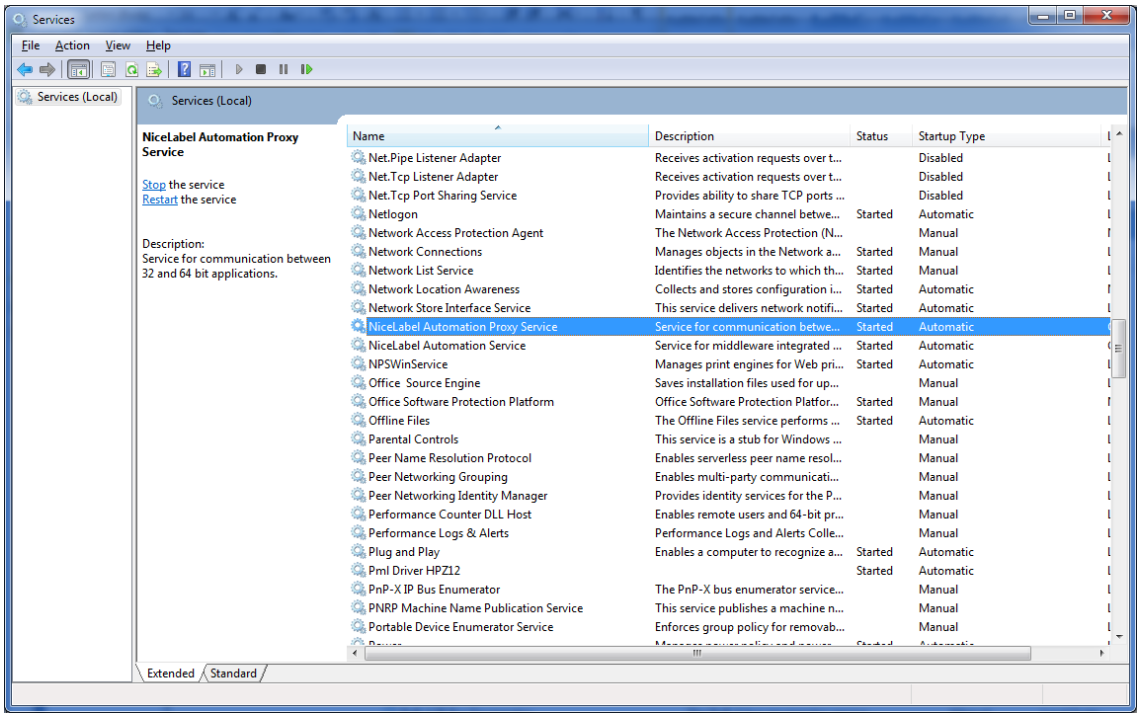


Figure 13 - Windows Services

Running NiceLabel Automation Service as 32-bit Service

When NiceLabel Automation is installed on 64-bit operating system it will automatically run as 64-bit application. Most of the time this is just the way you would want it, because it means there are no problems with splwow64.exe that may occur when printing from 32-bit application on 64-bit printer driver.

Did you know?

Splwow64.exe is printer driver host for 32 bit applications. In other words, splwow64.exe allows 32-bit applications to connect with the 64-bit printer spooler service on 64-bit Windows.

However there are situations where you would want to run the service as 32-bit application. Usually this happens when you want to connect to 32-bit database like Microsoft Access .mdb files.

To run NiceLabel Automation service as 32-bit you need to use Registry Editor to change the name of the executable in the following registry key

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\services\NiceLabelAutomationService
```

to

```
NiceLabelAutomationService.x86.exe
```

where you need to make sure the path to the executable stays the same. Administrative privileges are required to run Registry Editor. After you make the changes, you need to restart the service.

Warning

Using Registry Editor incorrectly can cause serious problems that may require you to reinstall Windows. Use Registry Editor at your own risk and consider creating a backup before making changes.

Silent Install

NiceLabel Automation installation can be started in silent mode. This means that the installation will not display messages or windows during its progress.

There are several command line switches that can be used with the installation program.

- **TARGETDIR** defines the location where NiceLabel Automation will be installed.
- **ICONDESKTOP** sets whether shortcut icon should be created on the desktop.

- Set **ICONQUICKLAUNCH** to TRUE to create shortcut icon in the quick launch area. This option is available only on Windows XP and Windows 2003.
- Set **SERVICEUSER** to define under which Windows user NiceLabel Automation service should run.
- **SERVICEPASSWORD** is needed to correctly use the user you defined with **SERVICEUSER**.

If **SERVICEUSER** and **SERVICEPASSWORD** are not passed to the installation, then NiceLabel Automation service will run under Local System account.

Setting service user and password

```
NiceLabelAutomation.exe /s SERVICEUSER=domain\username SERVICEPASSWORD=password
```

Setting Destination Path and Disabling Desktop Icon

```
NiceLabelAutomation.exe /s TARGETDIR= "D:\Apps\Path to Automation" ICONDESKTOP=FALSE
```

Enabling Quick Launch Shortcut

```
NiceLabelAutomation.exe /s ICONQUICKLAUNCH=TRUE
```

Tip

All command line switches are case sensitive. This means you have to enter them exactly as shown above (in uppercase).

Automation Builder

Automation Builder

Automation Builder is an application that allows you to build configurations by defining triggers, data filters, variables and actions.

When you start NiceLabel Automation Builder for the first time, you are presented with a Welcome screen.

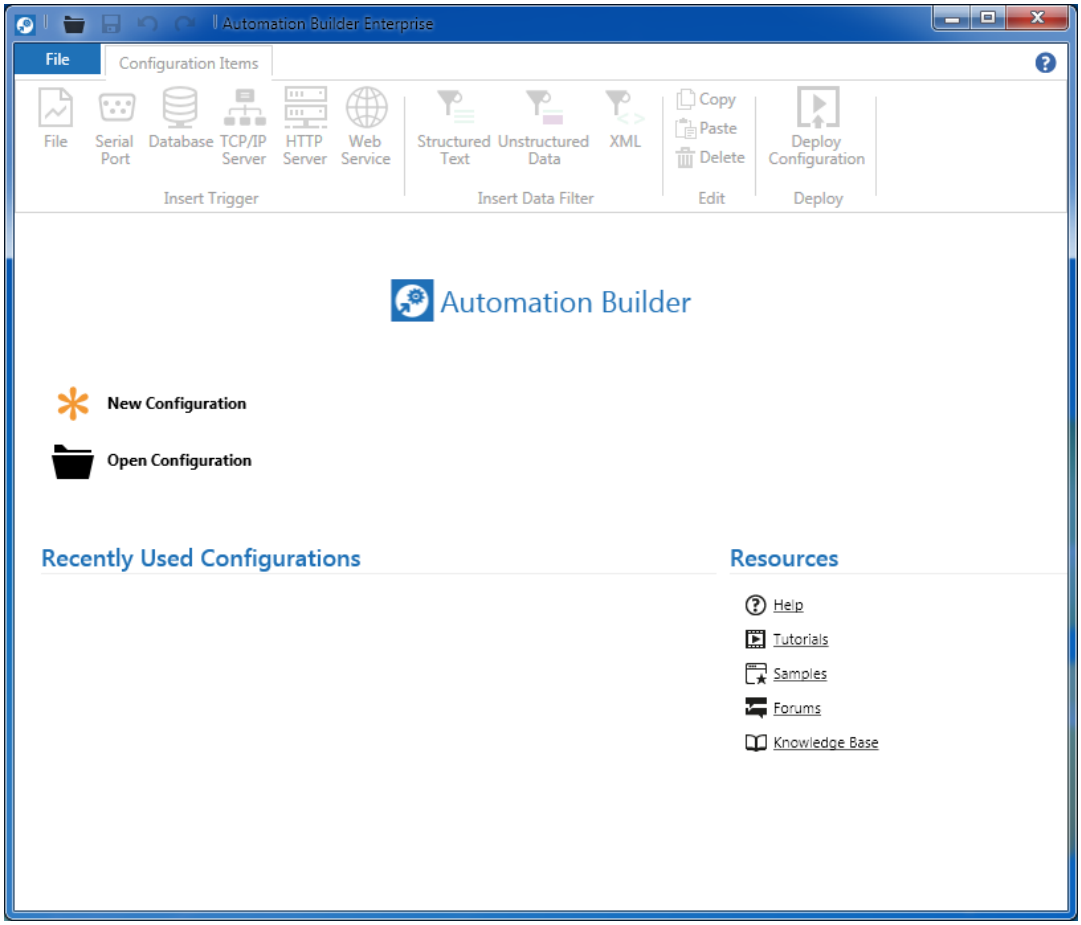


Figure 14 - Automation Builder

The Welcome screen allows you to quickly create new configuration, open an existing one or access previously opened configurations. If you need any assistance with the application you can easily view the help files, check tutorials, post a question on a forum, consult NiceLabel knowledge base, or contact support.

Visual Tour Around Automation Builder

In the screenshot below you can see the main elements of the Automation Builder user interface.

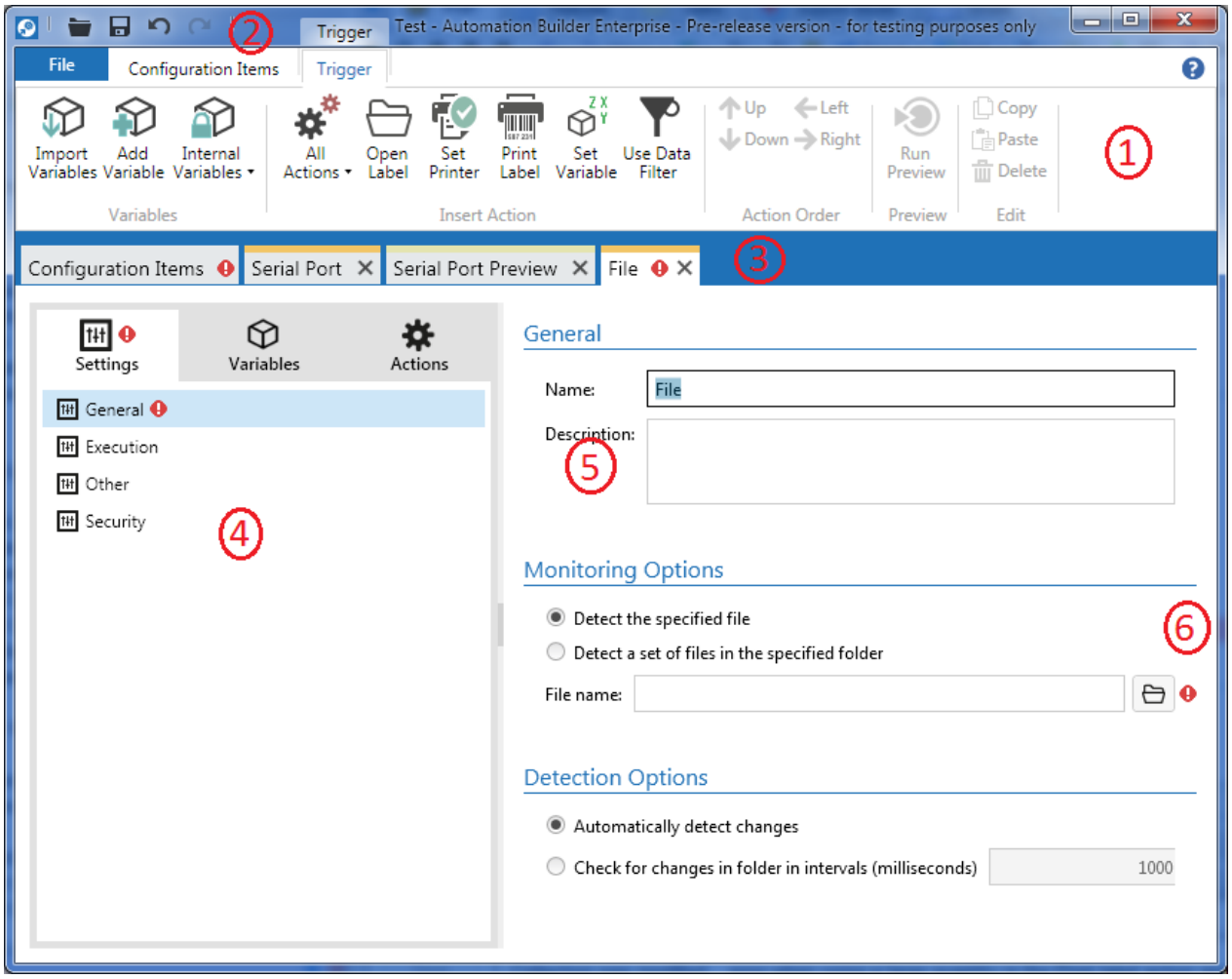


Figure 15 - Automation Builder Visual Tour

1. **Ribbon** with the command buttons for manipulating configuration items like triggers, data filters, actions and variables, and a **File** menu. Available commands in the ribbon depend on the selected configuration item. To access **File** menu click the **File** in the left top corner.
2. **Quick Access Toolbar** is used to quickly access **Open**, **Save**, **Undo**, and **Redo** commands.
3. **Configuration items tab area** where you can switch between different configuration items that are currently open. Every trigger and data filter has its own tab. **Trigger Preview** tab is used to display results of trigger preview execution. Only one **Trigger Preview** tab can be open at the time. Next preview will close the previous one and update the display with the new information. Tabs for different types of configuration items can easily be distinguished by the color of the top edge of the tab.
   Tabs can be closed by clicking the X button on the tab or by pressing CTRL + W. **Configuration Items** tab is a special tab and cannot be closed.
4. Left part of the work area is used for displaying categories of settings, list of variables and actions.
5. **Properties area** is used for properties that need to be defined for triggers, filters, actions and variables.

6. **Error indicator** that indicates that there is an error in the configuration setup. An error in the configuration item definition will be indicated on all levels of the configuration. As you can see on the screenshot above, because of the error in the definition of the **Monitoring Options** property of the **File Trigger**, error indicator is shown next to the field for that property, in the category where the property is displayed, in the **Settings** tab, in the trigger tab and on the **Configuration Items tab**. This way you can easily find the error in the configuration.

Creating New Configuration

You can create new configuration in three ways. You can click the **New Configuration** button in the **Welcome screen**, select the **New Configuration** from the **File** menu, or just press CTRL + N.

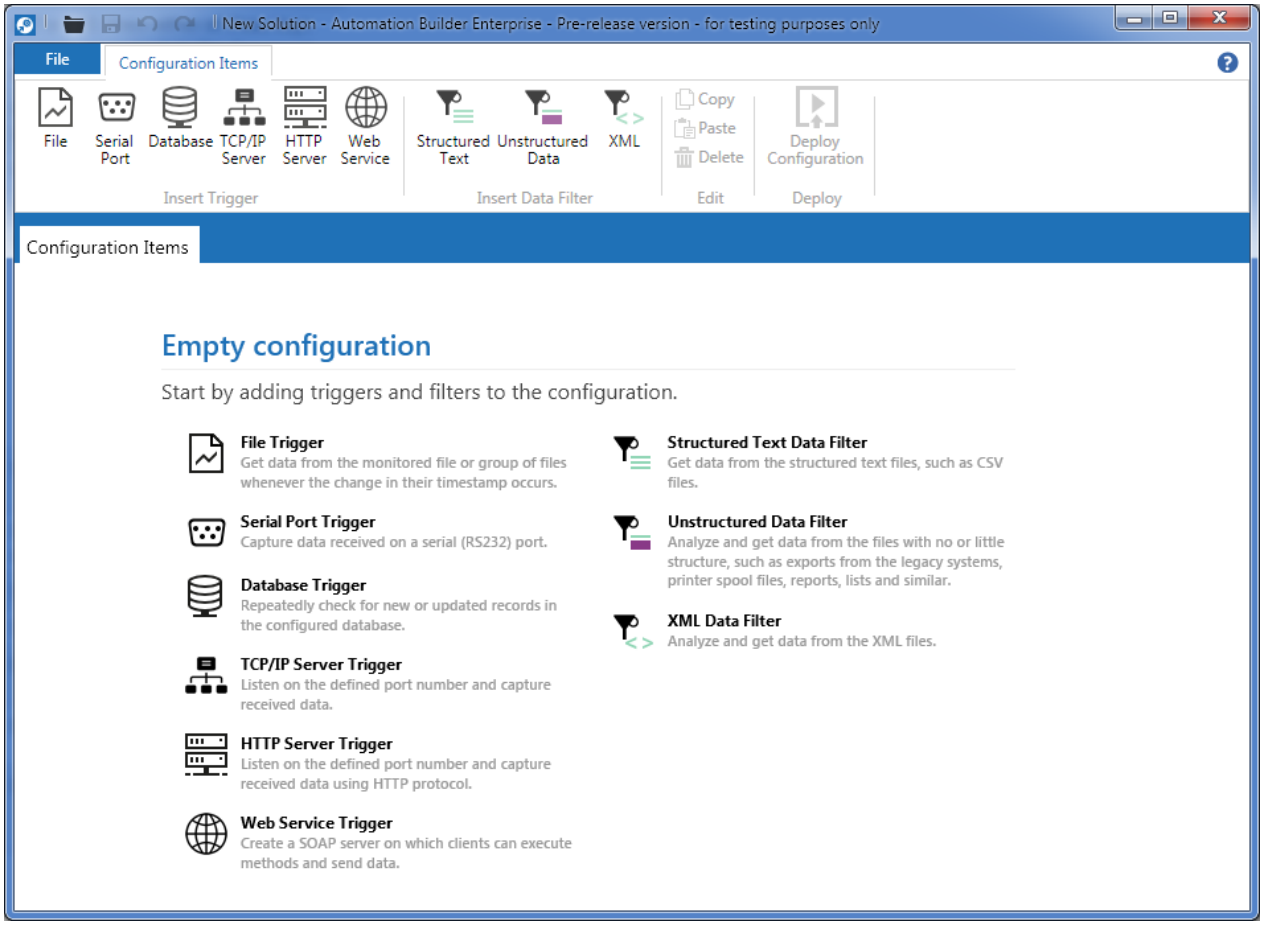


Figure 16 - Automation Builder - New Configuration

You will be presented with a screen that allows you to quickly see which triggers and data filters are available to you, and to start adding them to the configuration. To add new trigger or filter, click the button with the filter name on the screen or select it in the ribbon at the top.

Note

Number of triggers and data filters you see on your screen might be different, because not all triggers and data filters are available in all editions of NiceLabel Automation.

Saving Configuration

When you want to save the configuration, press CTRL + S, or select **Save** option in the **File** menu, or click the **Save** button in the left corner at the top of the screen (**Quick Access Toolbar**).

If your configuration is already saved, and you want to save it under another name, select **Save As** option in the **File** menu.

Opening Configuration

There are several possibilities how to open an existing configuration.

From Welcome Screen

Click the **Open Configuration** button in the **Welcome screen**, or select one of previously opened configurations.

From the File menu

Click **Open Configuration** from the **File** menu or select the configuration from a list of **Recently Used Configurations** that is also available in the **File** menu.

From Quick Access Toolbar

Open Configuration button is also available in the **Quick Access Toolbar** at the top of the window.

From Windows Explorer

If you are in a Windows Explorer and want to quickly open the configuration file, you can either double click the file (this will open new instance of Automation Builder), or just drag and drop the file into Automation Builder that is already running.

From Automation Manager

Automation Manager also allows you to edit configuration if you need to change it. To do this click the drop-down arrow next to the configuration name and select **Edit Configuration** from the popup menu.

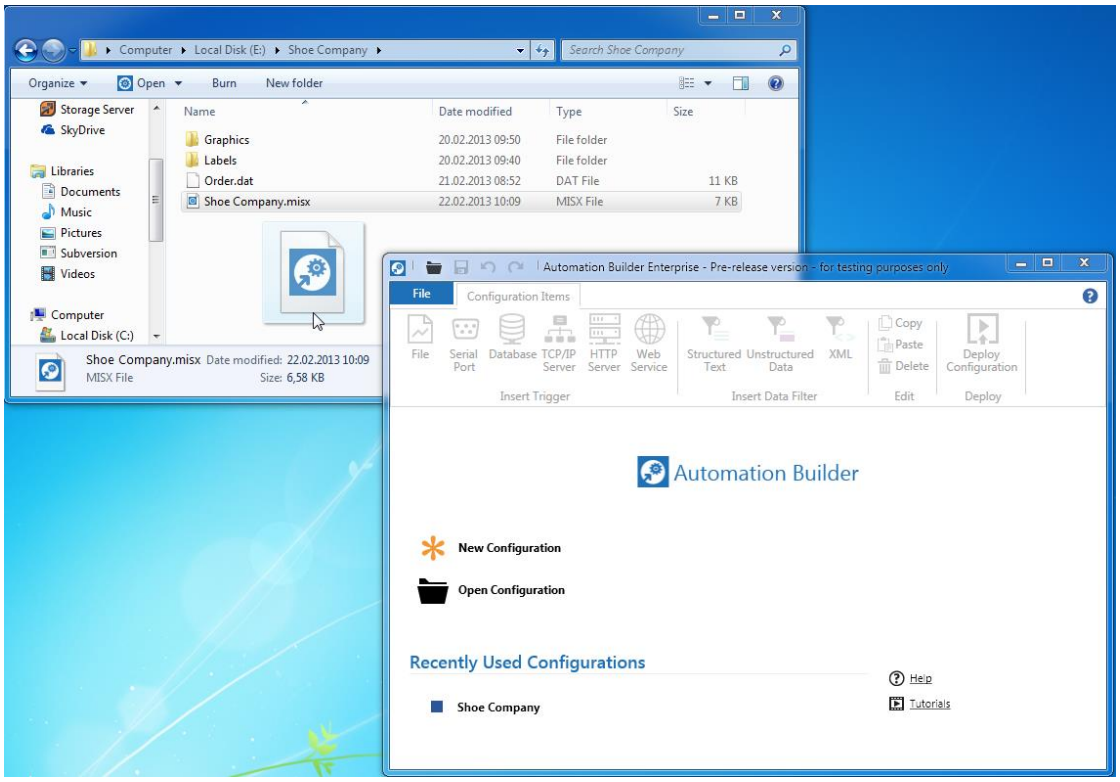


Figure 17 - Drag and Drop Configuration File

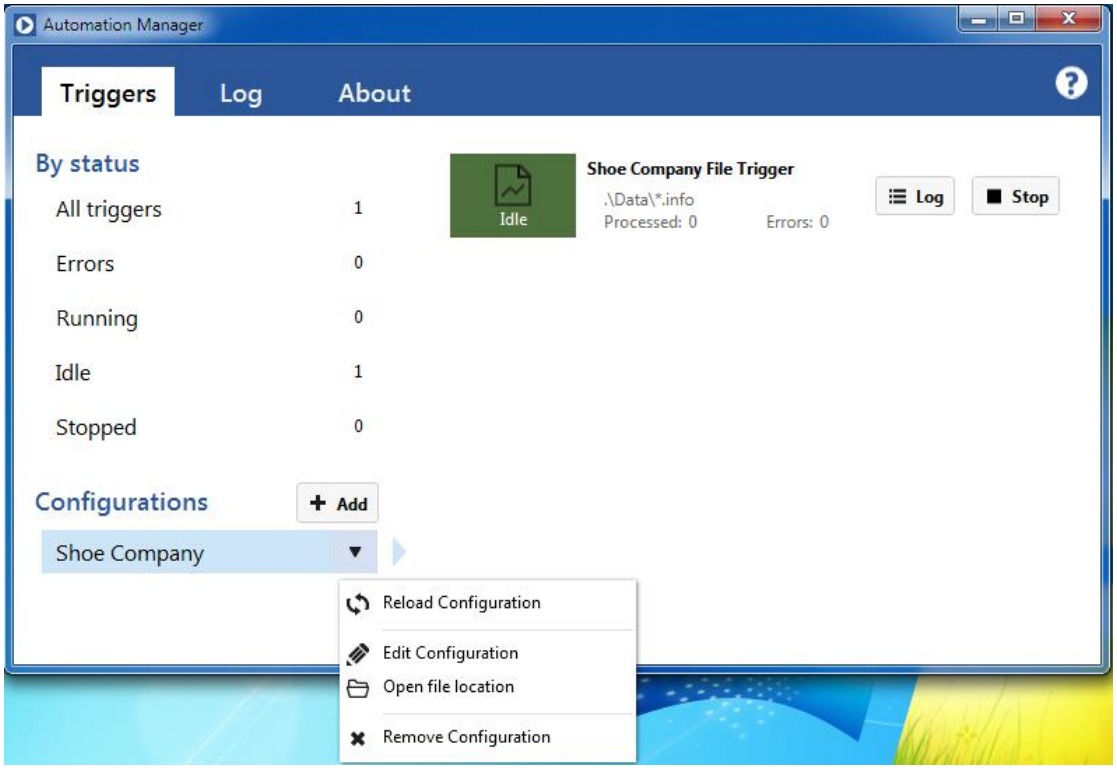


Figure 18 - Automation Manager - Edit Configuration

Opening NiceWatch File

NiceLabel Automation also supports configurations (.mis files) that were built with NiceWatch or NiceWatch Enterprise. To open such files you need to select **Open NiceWatch File** option from the **File** menu.

Note

For more information about the differences between NiceLabel Automation and NiceWatch, see section What Has Changed in Comparison with NiceWatch?

Testing Configuration

When you want to check if your configuration is correctly set, you can use **Run Preview** to see how your filters and actions will work. You can find the **Run Preview** button on the **Trigger** ribbon tab when the trigger is selected.

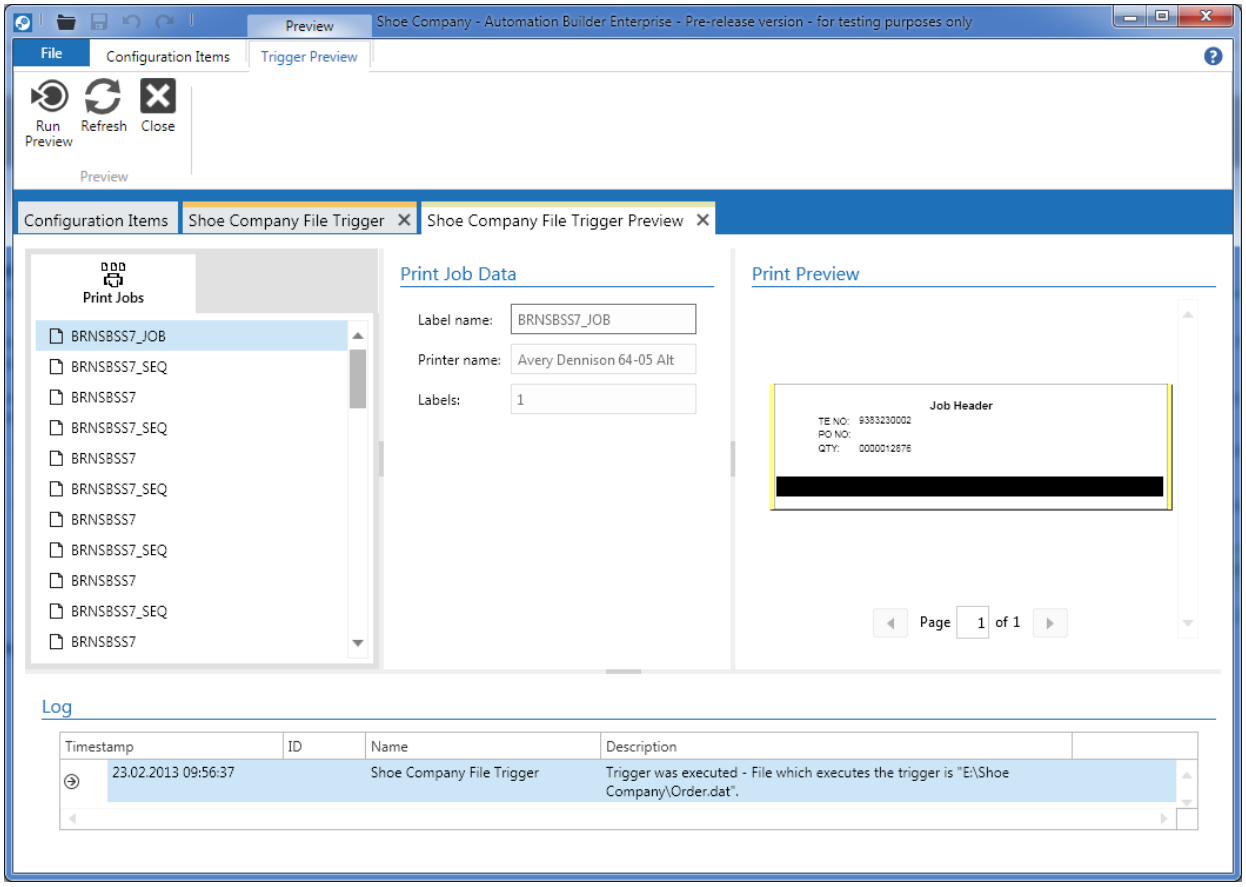


Figure 19 - Automation Builder - Trigger Preview

When you click the **Run Preview**, Automation Builder will prompt you to select a data file. This data file is needed to simulate the data that might otherwise come from a file (if File trigger is used), serial port (if

Serial Port Trigger is used), database (in case of Database trigger), etc. You need to provide the data needed to run preview successfully because Automation Builder does not process / execute triggers, so it cannot communicate for example with TCP/IP, HTTP or Web Service clients.

Trigger preview will show previews of all the labels that would be printed and also display the log of all actions that were executed.

Note

Run Preview button will not be enabled if configuration is not saved or if it is in the error state.

Deploying Configuration

Once you have setup your configuration you need to deploy it to Automation Manager, where triggers and all other configuration items are really used and executed.

To deploy the configuration to Automation Manager, save the configuration and then go to the **Configuration Items** tab in the ribbon and click the **Deploy Configuration** button. If Automation Manager is already running, configuration will be added to the list of configurations in Automation Manager. If it is not running, then it will be opened first, and only then will the configuration be added.

Of course you can always manually open Automation Manager and add the configuration there.

Undoing and Redoing Changes

Automation Builder allows you to undo or redo the changes you have made to the configuration. This mechanism works on all configuration items - triggers, filters, actions and variables. For example if you accidentally remove a trigger, you can reverse the action by simply pressing CTRL + Z or clicking the **Undo** button in the Quick Access Toolbar in the ribbon. The same goes for configuration item properties (e.g. variable name). You can also undo operations like adding new variable or new action to the list.

For every undo operation you have the possibility to redo it by pressing CTRL + Y or clicking the **Redo** button in the Quick Access Toolbar.

Number of undo or redo steps is not limited.

Automation Manager

Introducing Automation Manager

While Automation Builder is the design part of NiceLabel Automation, Automation Manager is the application for managing and controlling configurations and triggers.

There are three tabs in the application:

- Triggers
- Log
- About

Triggers

Triggers tab is where you work with configurations that you have previously designed in Automation Builder, start and stop triggers, view trigger activities and possible errors.

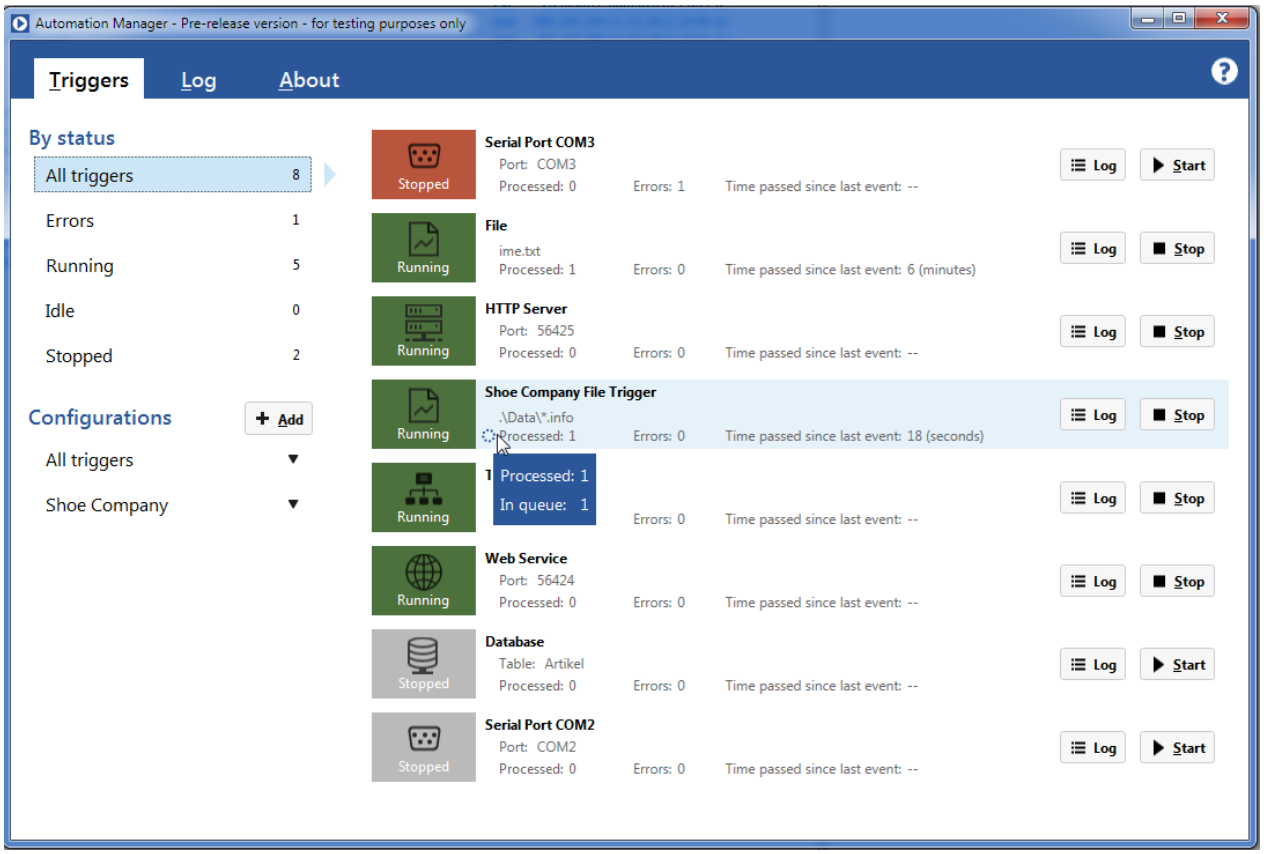


Figure 20 - Automation Manager - Triggers

Adding Configuration

To add a configuration to Automation Manager you can, either deploy it from Automation Builder, or add it manually by clicking the **Add** button and browsing for the .misx file.

Editing Configuration

If you decide that you need to modify configuration that is currently opened in Automation Manager, first select the configuration, and then right click it or click the drop down arrow. A popup window will appear, where you can select **Edit Configuration** to open it in Automation Builder.

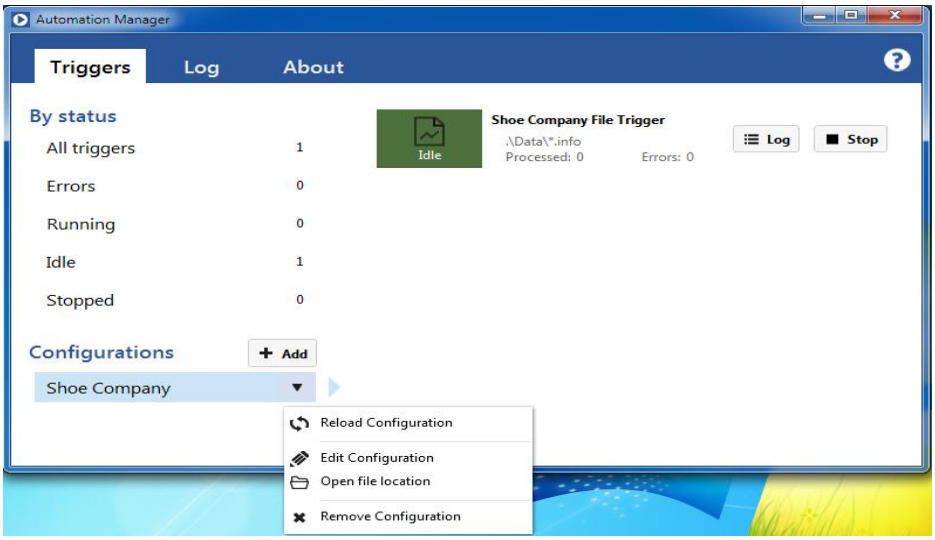


Figure 21 - Automation Manager - Configurations Popup Menu

Reloading Configuration

If you change the configuration in Automation Builder, you need to reload it manually, because Automation Manager and Automation service will not reload it automatically. To do that, select **Reload Configuration** in the popup menu. This was a conscious decision by the NiceLabel Automation team, because in most cases automatically reloading the solution is not a very good idea (e.g. when the application is running in the production).

Removing Configuration

To remove configuration from the list of active configurations just press DELETE key when the configuration is selected, or select **Remove Configuration** from the popup menu.

Before configuration is removed, you will be asked for confirmation of the action.

Opening Folder where Configuration is Stored

If you hover a mouse over the configuration name, a tooltip will appear containing full path to the configuration file. If you want to open the folder where the configuration file is stored, select **Open file location** in the popup menu.

Changing Trigger View

Triggers can be in different states:

- Stopped

- Running
- In error
- Idle (trigger is idle if It hasn't been active within the last 8 hours)

If you have lots of triggers or just want to see some of them (e.g. triggers with errors), you can group them by selecting one of the options in the left part of the screen called **By status**:

- **All triggers**: All triggers are shown in the list.
- **Errors**: Only triggers with errors are displayed.
- **Running**: Triggers that are running without errors.
- **Idle**: Idle triggers are shown.
- **Stopped**: Only inactive triggers are visible in the trigger list.

Starting and Stopping Triggers

If trigger is not started it will not react to any events, process data and execute actions – it will basically be useless. To start a trigger, select it in the list, and click the **Start** button. You can also just press SPACE key once the trigger is selected.

To stop the trigger do exactly the same. Once the trigger is running, the **Start** button will change to **Stop** button.

Tip

To start or stop all triggers, first select all of them by pressing CTRL + A, and then press SPACE key to start or stop them.

Notifications

Notification pane is displayed above trigger list when needed. This part of the Automation Manager window is shown when:

- NiceLabel Automation is running in trial mode.
- Trial mode has expired.
- NiceLabel Automation is getting a license from NiceLabel Enterprise Print Manager but the license cannot be obtained.
- Tracing is enabled.
- Any combination of above possible situations happens.

Opening Log

To open the log tab you can, either click the **Log** tab at the top of the window, or click the **Log** button in the selected trigger. The latter will show log only for the selected trigger.

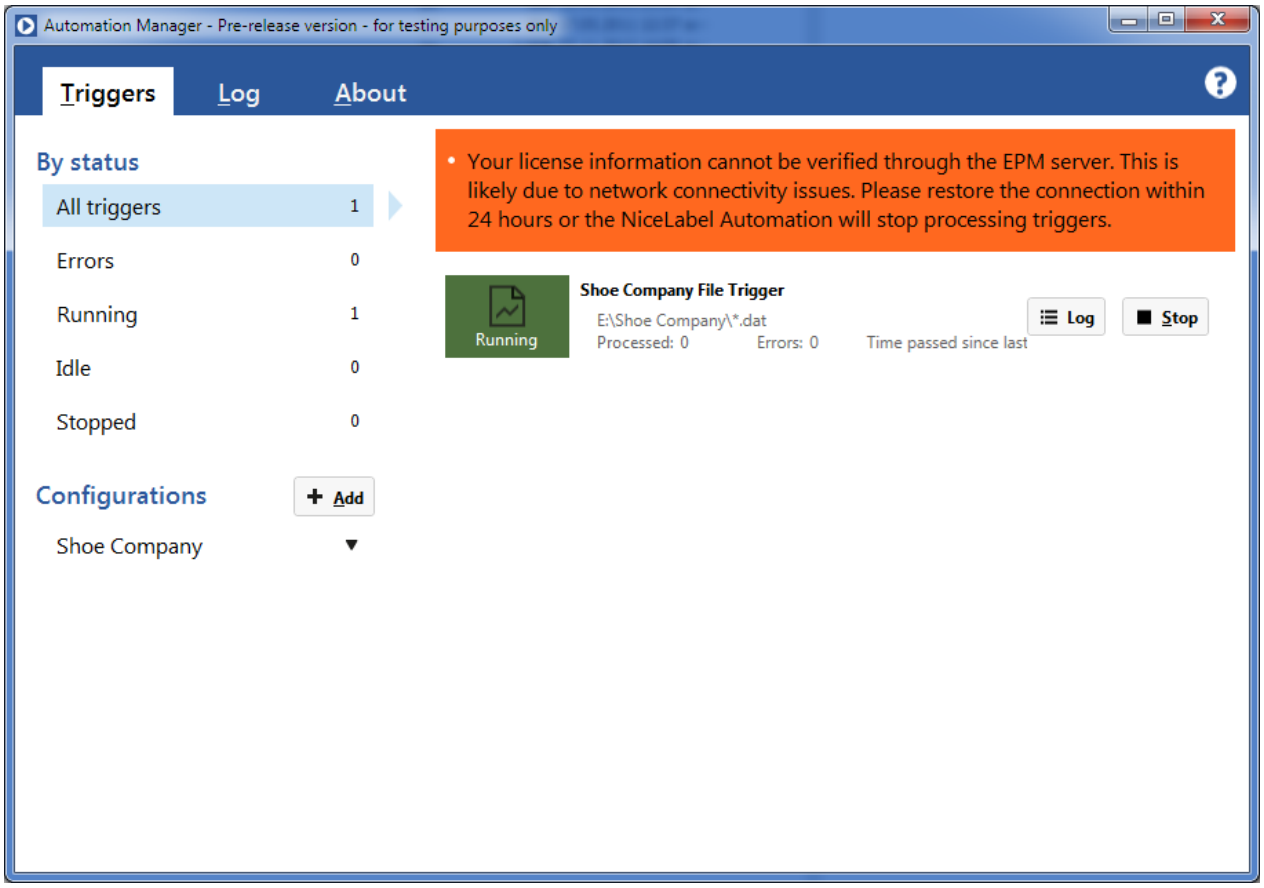


Figure 22 - Automation Manager - Notification Pane

Log

Automation Manager saves information about every trigger and action execution in a log stored in a database. This allows you to see everything that was going on with your configurations and troubleshoot any problems you might have.

For every trigger execution you can find information when the execution happened, which trigger was used and details about what triggered the trigger execution. Additionally the log also contains the date and time of an action execution, ID and name of the action so you can easily find it in Automation Builder, and more detailed description with information about the action.

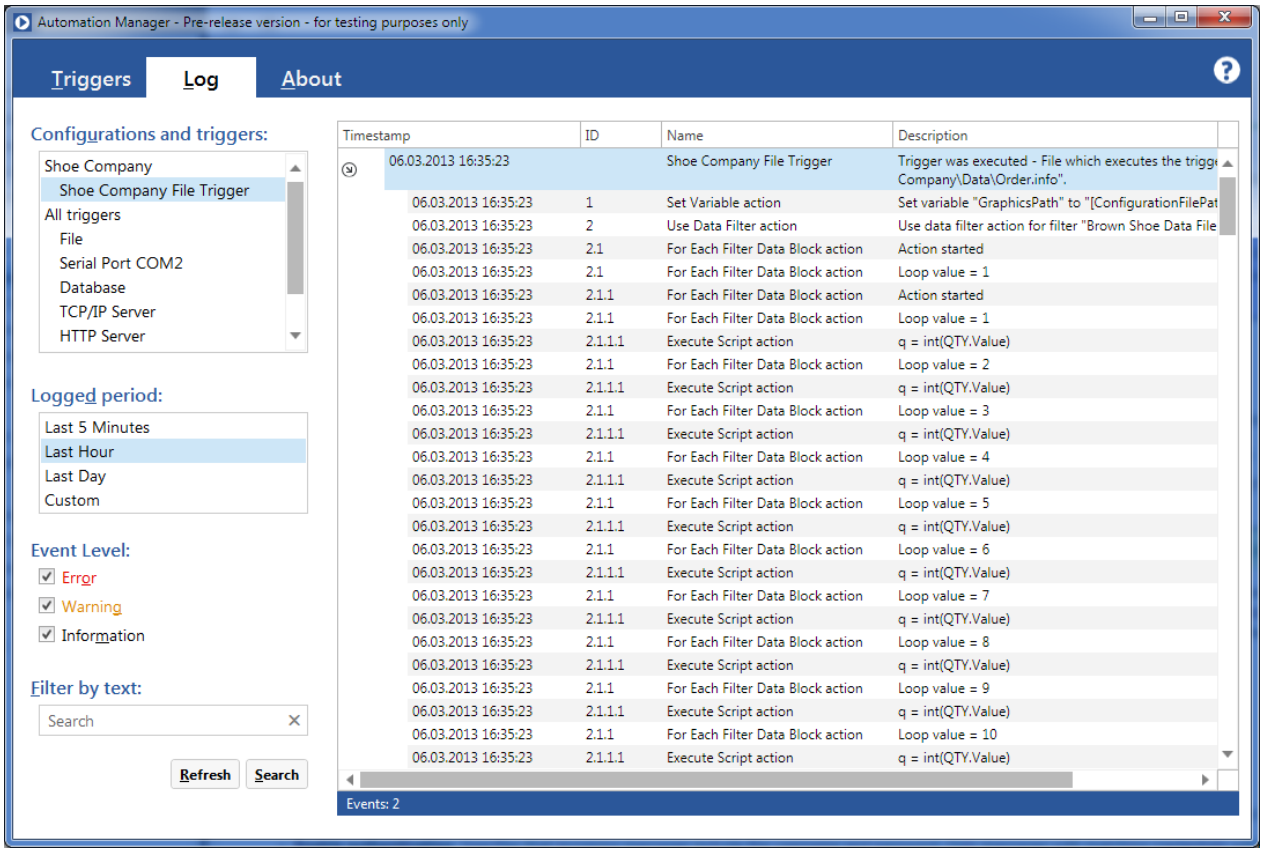


Figure 23 - Automation Manager - Log

Filtering

NiceLabel Automation log can contain thousands and even millions of entries, depending on the number of triggers and actions you have, and how often those triggers and actions are executed. Automation Manager offers several ways of filtering, so you only see information you need to see.

First you can filter them by selecting only the configuration and the trigger you are interested in. Then you can define for which time period entries should be shown. Finally you can select for which event level entries are displayed. There are three levels of entries, each displayed in its own color: errors, warnings and informational entries.

Searching

If there is still too much information in the log, you can also search for specific words. For example if you know the name of the label that is not printed correctly, enter that name in the **Search** field and press ENTER or click **Search** button. Only entries that contain the word you are searching for will be shown.

Clearing Log

Old items are automatically removed from the log. By the default this is done every day and all log entries older than 7 days are removed. You can change this setting in the Automation Configuration.

About

About page displays information about the license and application. Additionally you can also manage license from here by clicking the **Manage license…** button.

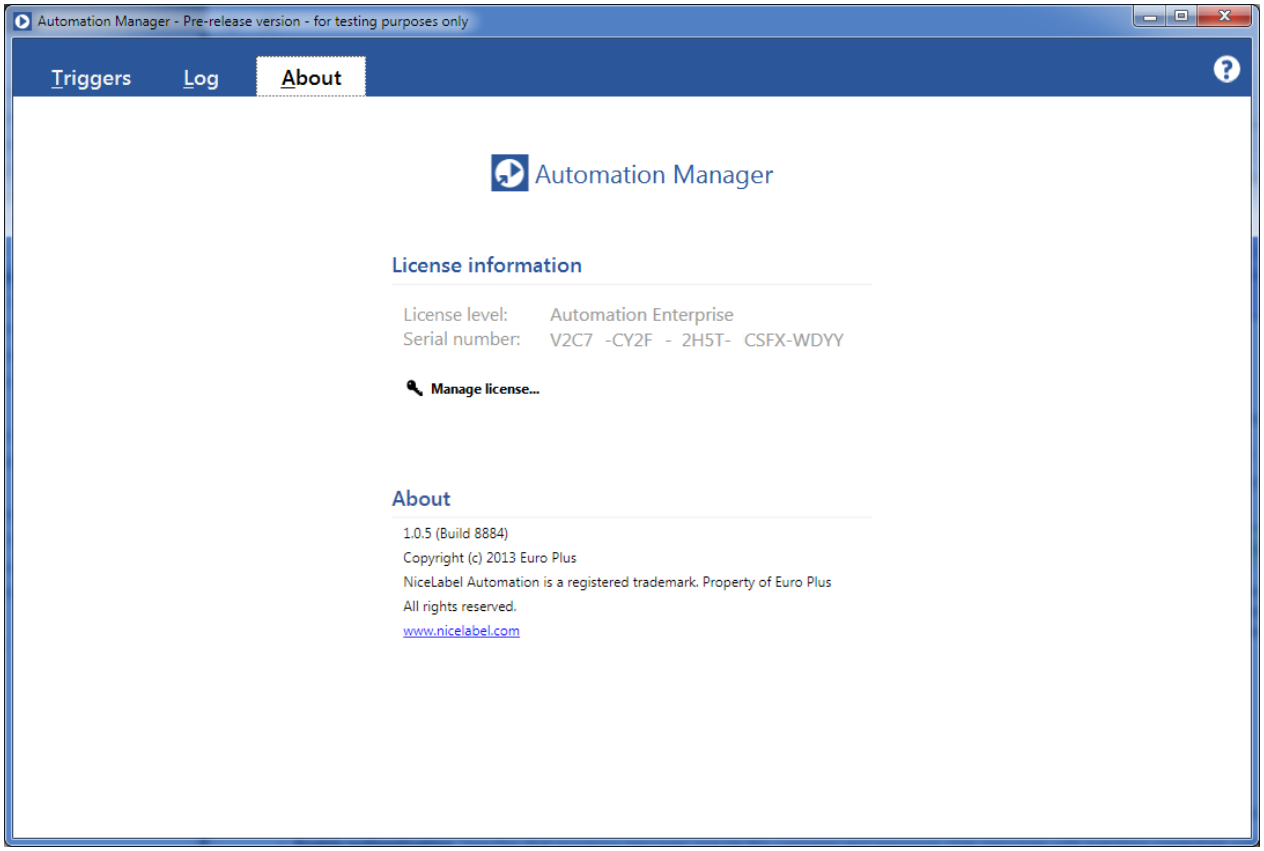


Figure 24 - Automation Manager - About

Help! I Need Somebody

In the top right corner of Automation Manager window you can find the **Help** button (big blue question mark in a white circle). **Help** button gives you access to the

- Local help file (also available by pressing F1 key)
- **Tutorials**: Short videos that introduce NiceLabel Automation and walk you through the basics of creating and deploying triggers.
- **Samples**: NiceLabel Automation comes with several samples. Each sample is in its own folder and contains all necessary files (configuration, data file, labels, etc.) and a short description of what it does in a .PDF format.
- **Forums**: http://forums.nicelabel.com/ is a great source of information. Questions get answered by other users and NiceLabel team.
- **Knowledge Base** is a large information repository in a form of a problem – solution description . Since NiceLabel Automation is a new product there might not be much information about it yet.
- **Technical Support**: If nothing else helps, you can contact the support team. See the Troubleshooting chapter for more details.

Command Line Parameters

NiceLabel Automation Manager also allows you to manage the configurations and triggers from the command line. You can

- Add configuration (if configuration is already loaded, it will be re-loaded)
- Reload configuration
- Remove configuration
- Start trigger
- Stop trigger

Usually Automation Manager will not be shown. If you want the Automation Manager to be visible after the operation has completed, include /SHOWUI in the list of parameters.

Here are a few samples that show how command line parameters for Automation Manager can be used.

Adding Configuration

```
NiceLabelAutomationManager.exe ADD "e:\Shoe Company\Shoe Company.misx"
```

Adding Configuration and Showing Automation Manager

```
NiceLabelAutomationManager.exe ADD "e:\Shoe Company\Shoe Company.misx" /SHOWUI
```

Reloading Configuration

```
NiceLabelAutomationManager.exe RELOAD "e:\Shoe Company\Shoe Company.misx"
```

Removing Configuration

```
NiceLabelAutomationManager.exe REMOVE "e:\Shoe Company\Shoe Company.misx"
```

Starting One Trigger

```
NiceLabelAutomationManager.exe START "e:\Shoe Company\Shoe Company.misx" TriggerName
```

Starting All Triggers

```
NiceLabelAutomationManager.exe START "e:\Shoe Company\Shoe Company.misx"
```

Stopping All Triggers in Configuration

```
NiceLabelAutomationManager.exe STOP "e:\Shoe Company\Shoe Company.misx"
```

Getting Results

If you want to get the return code of the command, you need to execute it a bit differently:

```
start /wait NiceLabelAutomationManager.exe ADD "e:\Shoe Company\Shoe Company.misx"
```

Return code of the execution will be stored in a system variable called ERRORLEVEL. To display its value enter the following command in the command prompt:

```
echo %ERRORLEVEL%
```

Automation Manager can return one of the following exit codes:

- **0**: Everything worked without problems.

- **100**: Configuration file could not be found.
- **101**: Configuration could not be loaded.
- **200**: Trigger could not be found.
- **201**: Trigger could not be started.

Note

You need to include the full path to the Automation Manager executable in command line, otherwise the operating system cannot find the program.

On a 64-bit operating system and if the application is installed in the default location, the path should look something like this

"c:\Program Files (x86)\EuroPlus\NiceLabel Automation\bin.net\NiceLabelAutomationManager.exe"

Variables

Variables

Variable is a facility for storing data. Variables allow you to design flexible and reusable automation solutions.

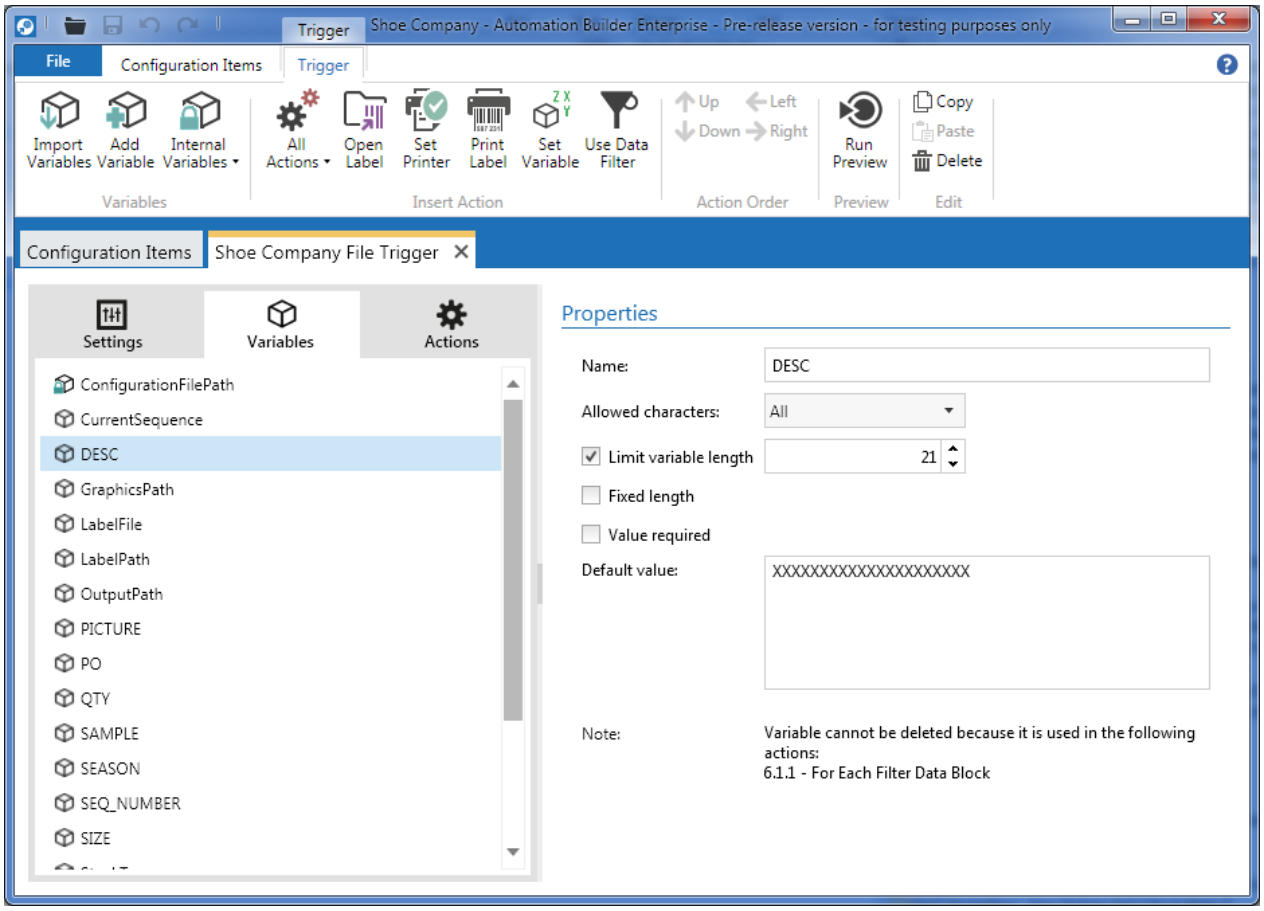


Figure 25 - Automation Builder - Variables

Properties of a Variable

Every variable – and it does not matter if you imported it from a label file, loaded it from NiceWatch .mis configuration or created it manually – has several properties that allow you to control its behavior.

- **Name**: Name of variable can be any string and it can contain digits, letters, spaces and other characters like (,), +, =, etc. Even if almost all characters are allowed in a name, try to give variable a meaningful name so that you will know after a year after you designed a trigger, what you meant with it. For example, “TypeOfProduce” is much better name than “Var123”.
- **Allowed characters**: Three categories of allowed characters are available – **All** (all characters are allowed), **Numeric** (only digits can be used) , and **Binary** (all characters are accepted and no encoding is used when loading or saving such values).
- **Limit variable length**: In NiceLabel, NiceForm and NiceWatch variable always had to have a length defined and the maximum possible length was 4096 characters. In NiceLabel Automation this has changed and now by default a variable has unlimited length. If you need to limit the

variable length (some barcodes for example do not allow data of arbitrary length), you can do that by enabling **Limit variable length** setting and entering the required length.

Tip

If you create a variable in NiceLabel and set its maximum length to 4096 characters, and then import that variable in Automation Builder, the variable in Automation Builder will NOT have a limited length.

- **Fixed length**: If variable length is limited, you can also define it always has to have a value of a fixed length. If this setting is enabled and you try to set variable to a value that does not have a correct length, an error will be reported.
- **Value required**: Variable must have a value, otherwise an error is reported.
- **Default value**: Every variable can have a default value.

Note

If you intend to use variable in a script, in the action condition, or in the Execute SQL Statement action, you must be careful how you name it. I suggest you only use letters and digits, without spaces or any other characters because otherwise script or condition might not work. Here are some samples of variable names that will NOT work in a Python script:

First Name
123 Var
!!Address!!

Importing Existing Variables from Label Files

Automation Builder allows you to import variables from an existing label file. This is very helpful, especially if you have lots of variables, because you do not have to create them manually. To import variables from a label file, click **Import Variables** button in the **Trigger** ribbon that is enabled when you have a trigger opened, and browse for the label file.

Note

Only prompt variables will be imported from the label.

Adding New Variable

To add new variable all you have to do is click the **Add Variable** button in the **Trigger** ribbon. New variable will be added to the list with a name already defined, allowed characters set to **All** and with no default value.

You can also copy and paste an existing variable (CTRL + C, CTRL + V).

Removing Existing Variable

If variable is not used in an action you can also remove it. You can do that by selecting variable in the list and pressing DELETE key, or clicking the **Delete** button in the **Trigger** ribbon.

If variable is used in the action, a note will be displayed (as you can see in the screenshot from the beginning of this section), where you can see in which actions it is used.

Internal Variables

In NiceLabel Automation there are 34 internal variables (listed in alphabetical order below). Not all internal variables are available in all trigger types.

- **ActionLastErrorDesc**: Description of the last error that occurred during the execution of actions.
- **ActionLastErrorID**: ID (number) of the last error that occurred during the execution of actions.
- **BytesOfReceivedData**: Only available in the TCP/IP Server trigger. Contains the number of bytes received by the trigger.
- **ComputerName**: Name of the computer where the trigger is running.
- **ConfigurationFileName**: Full file name of the current configuration (including the path).
- **ConfigurationFilePath**: Path to the current configuration (without the file name).
- **Database**: Only available in the Database trigger. Name of the database used in the trigger.
- **DataFileName:** Full file name of the file where the data received by the trigger is saved to.
- **Date**: Current system date. Date is displayed in a format that is defined on the computer where the trigger is running.
- **DateDay**: Current day.
- **DateMonth**: Current month.
- **DateYear**: Current year.
- **DefaultPrinterName:** Name of the default printer on the computer.
- **DriverType**: Only available in the Database trigger. Type of the database driver used in the trigger definition.
- **HostName:** Only available in the TCP/IP Server trigger. Host name or IP address of a computer that connects to the trigger.
- **LocalIP**: Only available in the TCP/IP Server trigger. IP address of a TCP/IP Server trigger.
- **NumberOfRowsReturned**: Only available in the Database trigger. It contains a number of rows returned from the database when Database trigger is fired.
- **PathDataFileName**: Path to file where the data received by the trigger is saved to.
- **PathTriggerFileName**: Only available in the File trigger. Path to the file that caused the trigger to execute.

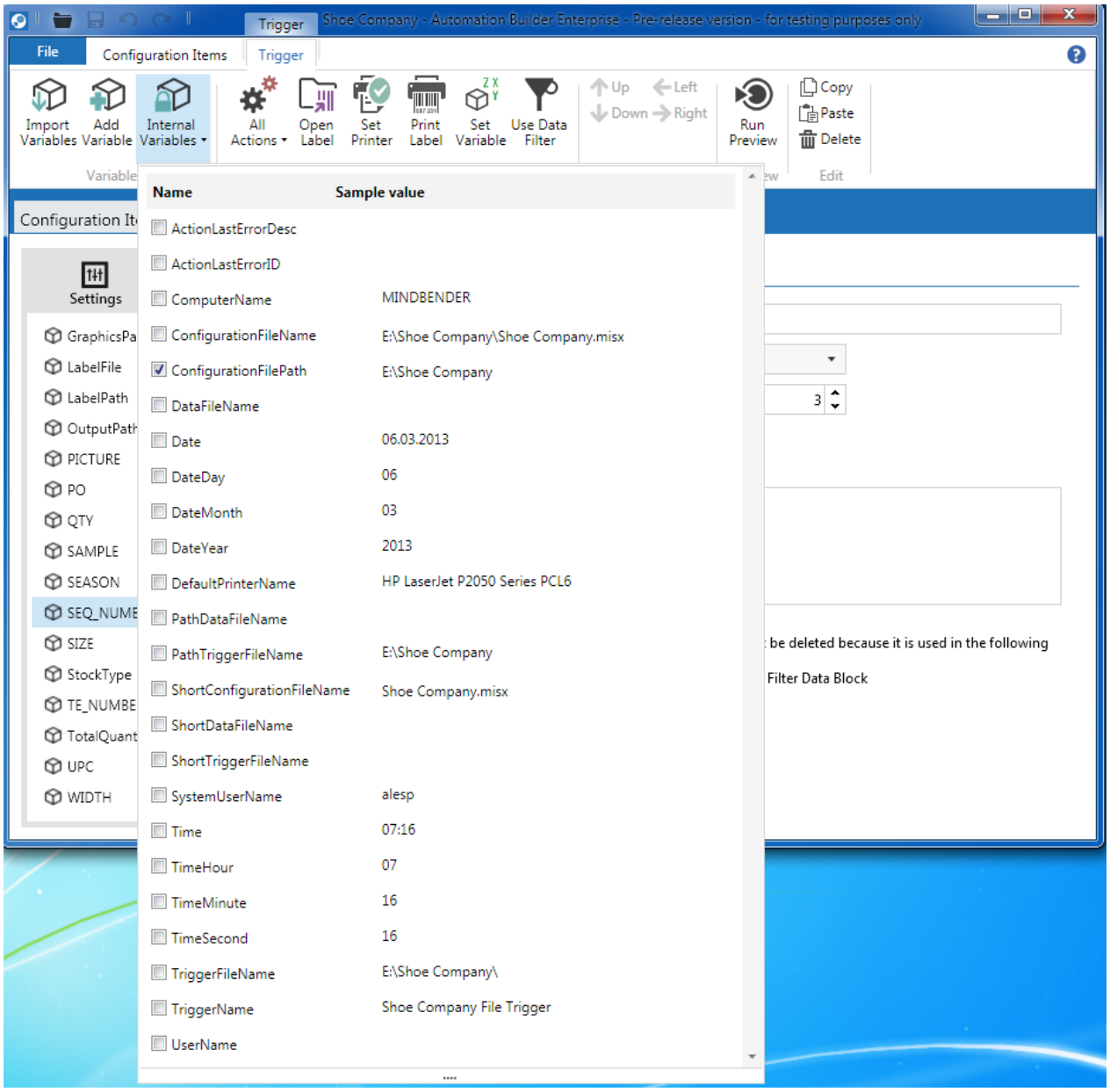


Figure 26 - Automation Builder - Internal Variables

- **Port:** Port number of the client that executed the trigger. Available in Web Service, TCP/IP Server and HTTP Server triggers.
- **RemoteHttpIp**: Only available in the HTTP Server trigger. IP address of a computer that connects to the trigger.
- **RemoteIp:** The IP address of the computer that connected to the trigger. Only available in the Web Service trigger.
- **ShortConfigurationFileName:** Short file name of the current configuration (without the path).

- **ShortDataFileName**: Short file name (without the path) of the file where the data received by the trigger is saved to.
- **ShortTriggerFileName**: Only available in the File trigger. Short name of the file that caused the trigger to execute. It does not contain the path.
- **SystemUserName**: Name of the currently logged-in Windows user.
- **TableName**: Only available in the Database trigger. Name of the table used in the trigger.
- **Time**: Current system time. The value is formatted according to the format defined on the computer where the trigger is running.
- **TimeHour**: Current hour.
- **TimeMinute**: Current minute.
- **TimeSecond**: Current second.
- **TriggerFileName**: Only available in the File trigger. Name of the file that caused the trigger to execute. It also includes the path to the file.
- **TriggerName**: Name of the current trigger.
- **UserName**: Application user name of the currently logged-in user. It only has value if application authentication is enabled.

Tip

If you want to use internal variable in a script or in some other action you need to enable it first. To enable internal variable click Internal Variables button in the Trigger ribbon and check the check box in the drop down list. The variable is then added to the list of variables and it can be distinguished from other variables by a different icon.

Variable	File Trigger	Serial Port Trigger	Database Trigger	TCP/IP Server Trigger	HTTP Server Trigger	Web Service Trigger
ActionLastErrorDesc	X	X	X	X	X	X
ActionLastErrorID	X	X	X	X	X	X
BytesOfReceivedData				X		
ComputerName	X	X	X	X	X	X
ConfigurationFileName	X	X	X	X	X	X
ConfigurationFilePath	X	X	X	X	X	X
Database			X			
DataFileName	X	X	X	X	X	X
Date	X	X	X	X	X	X
DateDay	X	X	X	X	X	X
DateMonth	X	X	X	X	X	X
DateYear	X	X	X	X	X	X
DefaultPrinterName	X	X	X	X	X	X
DriverType			X			
HostName				X		
LocalIP				X		
NumberOfRowsReturned			X			
PathDataFileName	X	X	X	X	X	X
PathTriggerFileName	X					
Port				X	X	X
RemoteHttpIp					X	
RemoteIp						X
ShortConfigurationFileName	X	X	X	X	X	X
ShortDataFileName	X	X	X	X	X	X
ShortTriggerFileName	X					
SystemUserName	X	X	X	X	X	X
TableName			X			
Time	X	X	X	X	X	X
TimeHour	X	X	X	X	X	X
TimeMinute	X	X	X	X	X	X
TimeSecond	X	X	X	X	X	X
TriggerFileName	X					
TriggerName	X	X	X	X	X	X
UserName	X	X	X	X	X	X

Table 1 - Internal Variables and Triggers

Using Variables

In NiceLabel Automation variables can be used in many places: as a source of value for action properties, as data source for objects that are used as part of label design, in database filters, in functions and scripts. Every action has at least one property that can be variable. For example:

- Label name of an **Open Label** action

- Printer name of a **Set Printer** action
- Quantity in **Print Label** action
- File name of a **Run Command File** action
- Start and End value of the **For Loop** action
- Etc.

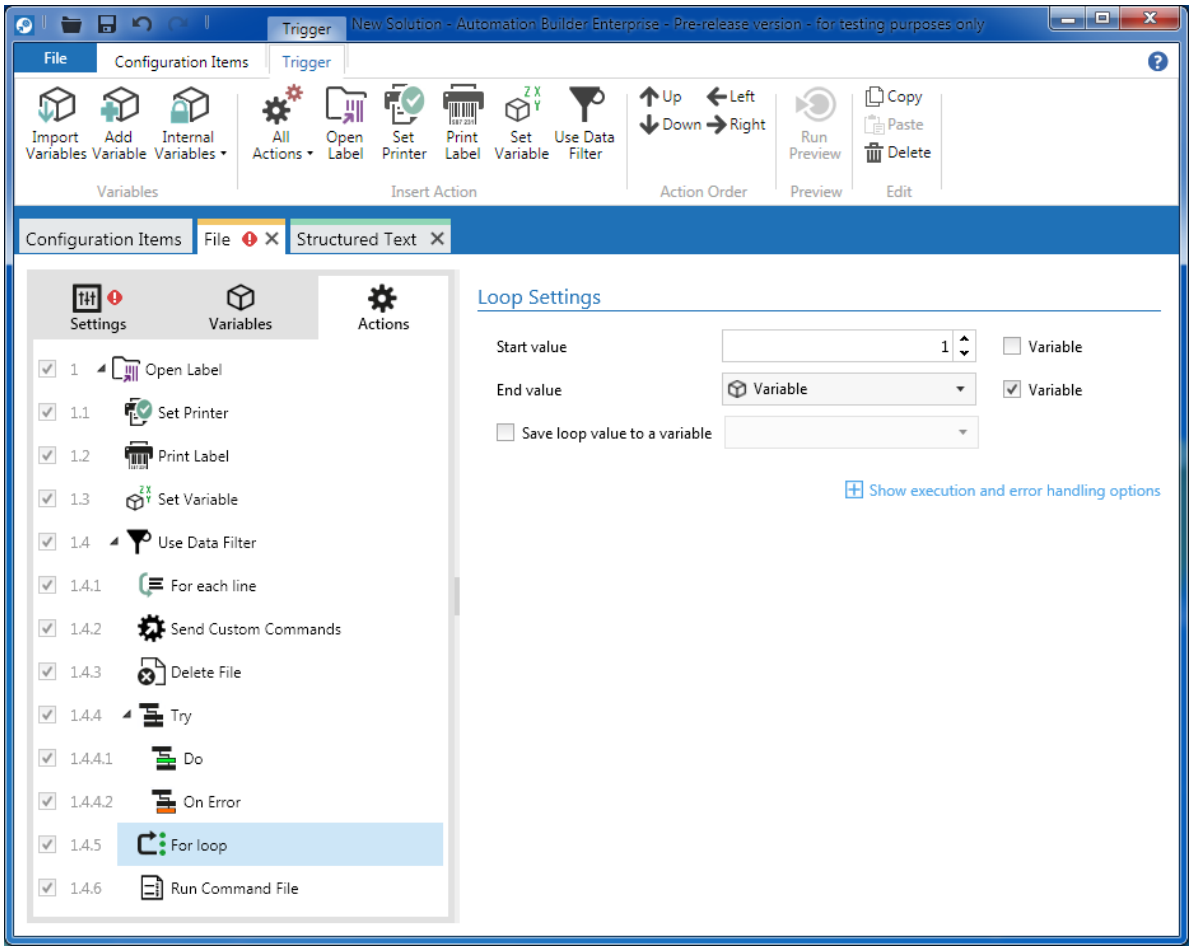


Figure 27 - Automation Builder - Using Variables

Triggers

Triggers

A trigger is a part of NiceLabel Automation that is automatically executed in response to certain events. NiceLabel Automation can react to six different events:

- A change was made to a file, or a file has been added to a folder
- A change was made to a database
- Data was received on a serial port
- Data was received over a TCP/IP port
- Data was received over a port using HTTP protocol
- A call was made to a web service method

As you can see from the table below, not all triggers are available in all editions of NiceLabel Automation.

Trigger	Automation Easy	Automation Pro	Automation Enterprise
File Trigger	X	X	X
Serial Port Trigger	X	X	X
Database Trigger	X	X	X
TCP/IP Server Trigger	X	X	X
HTTP Server Trigger		X	X
Web Service Trigger			X

Table 2 - Triggers in different editions of NiceLabel Automation

Working With Triggers

Adding New Trigger

There are two ways you can add a new trigger in Automation Builder. When configuration is still empty (there are no triggers or filters) you can click one of the trigger buttons on the **Configuration Items** page.

If the configuration already has some items, you can create new trigger by clicking appropriate button in the **Configuration Items** ribbon.

Copying Existing Trigger

To make a copy of a trigger just go to the **Configuration Items** page, select the trigger and press CTRL+C or right click it to open popup menu where you can select **Copy**. This will copy trigger to the clipboard from where you can the paste it, again by the keyboard shortcut (CTRL + V) or popup menu.

Editing Existing Trigger

Trigger can be edited / opened by going to the **Configuration Items** page and double clicking it, or by selecting it and clicking the **Edit** button.

Removing a Trigger

When a trigger is selected in the list, you can delete it by pressing DELETE key, or from a popup menu that is shown when you right click the trigger. Remember, you can always undo the changes you made to the configuration!

Common Trigger Properties

All triggers share some basic properties:

- **Name**: If you have lots of triggers it is important that you choose a descriptive name so that you will be able to easily find a trigger.
- **Description**: It is also smart idea to give a trigger a short description where you describe what the trigger does, especially if the trigger configuration is not a simple one and has lots of variables and actions.
- **Supervised printing**: When you need to send the feedback from the printing process in NiceLabel Automation back to the application that initialized it, you can enable this setting. Enabling supervised printing will enable synchronous printing mode, where execution of other actions is suspended long as the print process is running. The trigger sends the data to the print process and keeps the connection open until the print process is completed. When the print process completes successfully, or if an error occurs, the trigger will be notified about the status. You can use this information in the actions in the trigger.
  By default this option is disabled and printing is done asynchronously, which means that the trigger does not wait for the printing to finish before accepting other requests. That way much better trigger processing performance is possible.
- **Scripting language**: You can choose between two scripting languages - VBScript and Python. Choice of scripting language affects all **Execute script** actions you use in the trigger configuration and in condition properties of actions used. For example if you select Python as your scripting language, then you will have to write all your scripts and conditions for that trigger in Python.
- **Locking**: If you do not want everybody to see how triggers are configured or if you have written some VBScript or Python script that contains some sensitive information, or if you have designed some clever algorithm that you do not want your competition to see, you can protect a trigger with a password.

Tip

VBScript is used for backwards compatibility (if you use configurations made with NiceWatch where VBScript was the only option). Its execution is slower because it has to go through NiceLabelAutomationProxyService. If you have scripts that are executed many times and you face performance issues, you may consider using Python as your scripting language. Python allows for better performance because it is compiled into NiceLabel Automation.

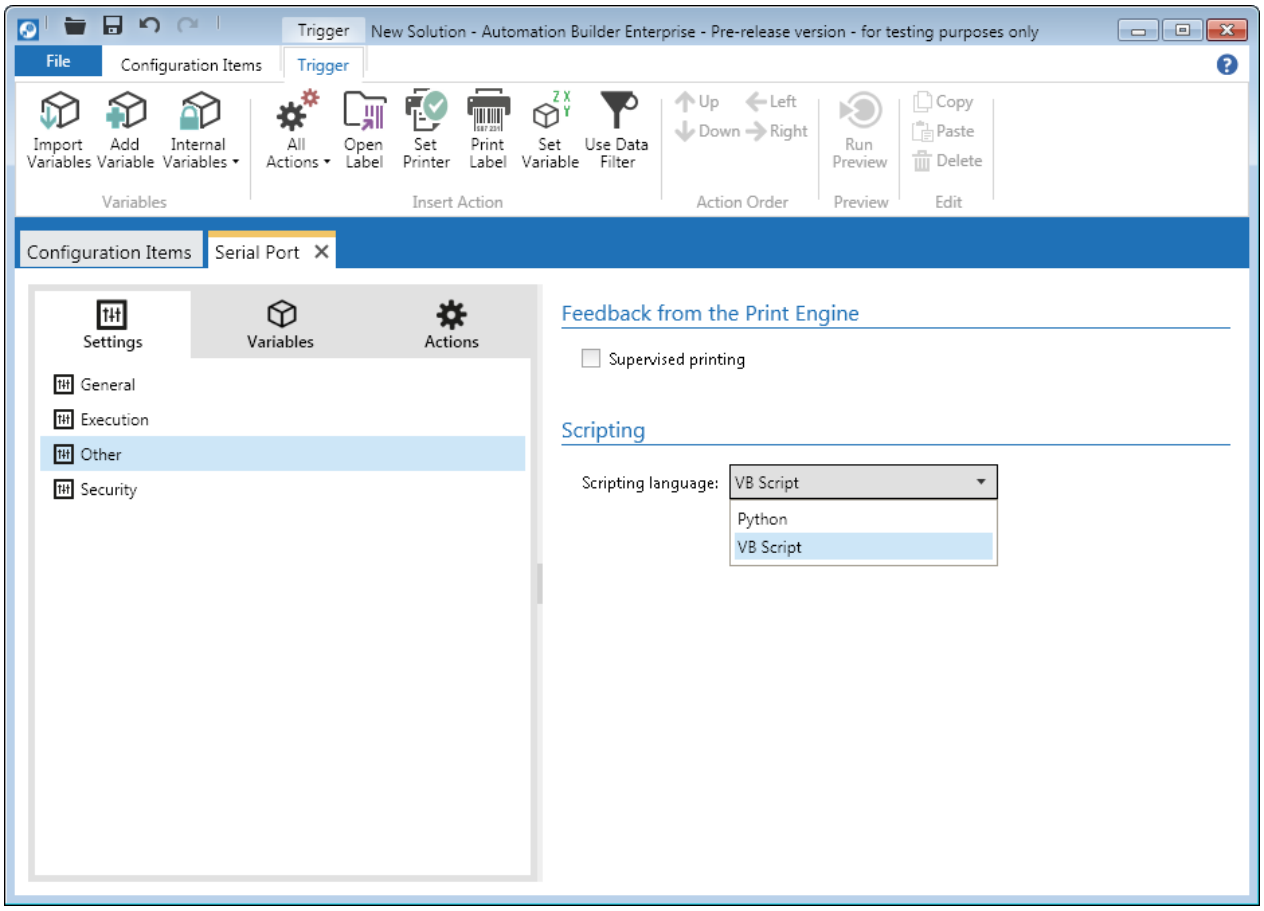


Figure 28 – Automation Builder - Trigger - Other

File Trigger

File trigger is the most commonly used type of trigger because it does not require much infrastructure or hardware (all other types of triggers require some sort of infrastructure like a database, or a special hardware like serial port). All you need to do is to drop a file in a specific folder or modify an existing file and the file trigger will pick up those changes and start filtering data and executing actions. File trigger will detect changes to:

- File name,
- File size,
- Last write time, and
- Creation time

Note

When a change to a file or a folder is detected, a copy of a file is created. This backup file can then be used in actions. You can get the path to the new file by using internal variable DataFileName.

Detecting Changes

File trigger reacts to changes that occur to a specific file or a set of files in a specific folder. From the screenshot below you can see that NiceLabel Automation will automatically react to changes that might happen to any of the files that have .dat extension and are located in the "E:\Shoe Company" folder.

If you always work with one specific file you can also configure the trigger to detect changes only for that file. You can do that by selecting the **Detect the specified file** option in the **Monitoring Options** section and defining a file name that needs to be monitored.

Usually you would configure file trigger to automatically detect changes to the file and this is also set by default when you create new file trigger. However you can also change that setting so that check for file or folder changes are made in a certain time intervals. You will probably only need this option when the file is stored on a network share where automatic detection is not possible (for example when a file is stored in a folder on a Linux computer).

Tip

Relative path is a path relative to the working directory of the application. In NiceLabel Automation you can use relative paths when defining file names or paths. For example, if you enter

.\Data\order.info

into File name property of the File Trigger, then NiceLabel Automation will react to changes made to file order.info that is stored in the Data folder. This Data folder can be located anywhere on your computer as long as it is in the same folder as the configuration file (it is relative to the location of the configuration file). If you setup your configuration files to use relative paths, you can easily transfer them between computers and different disks without any changes.

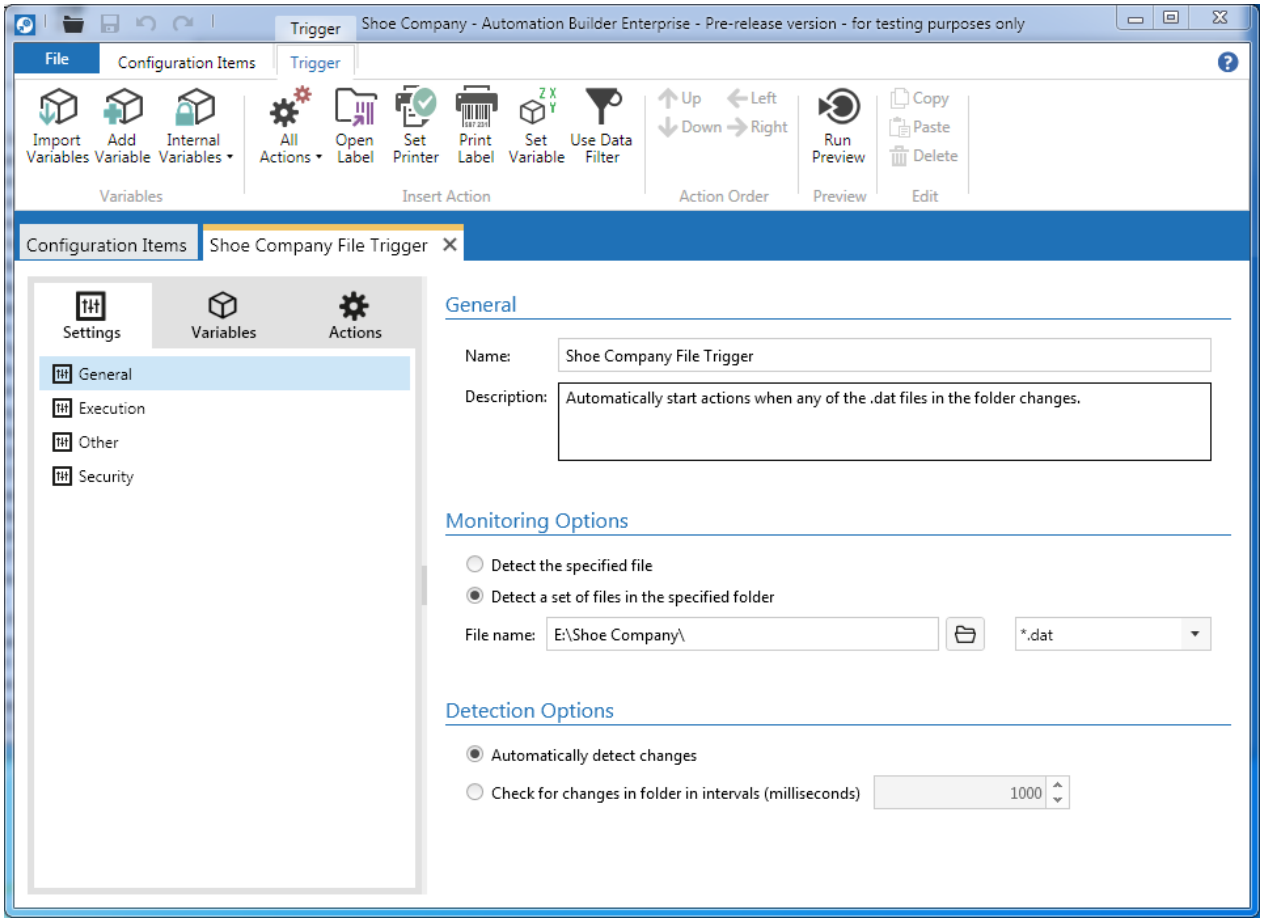


Figure 29 – Automation Builder - File Trigger - General Options

Accessing Files

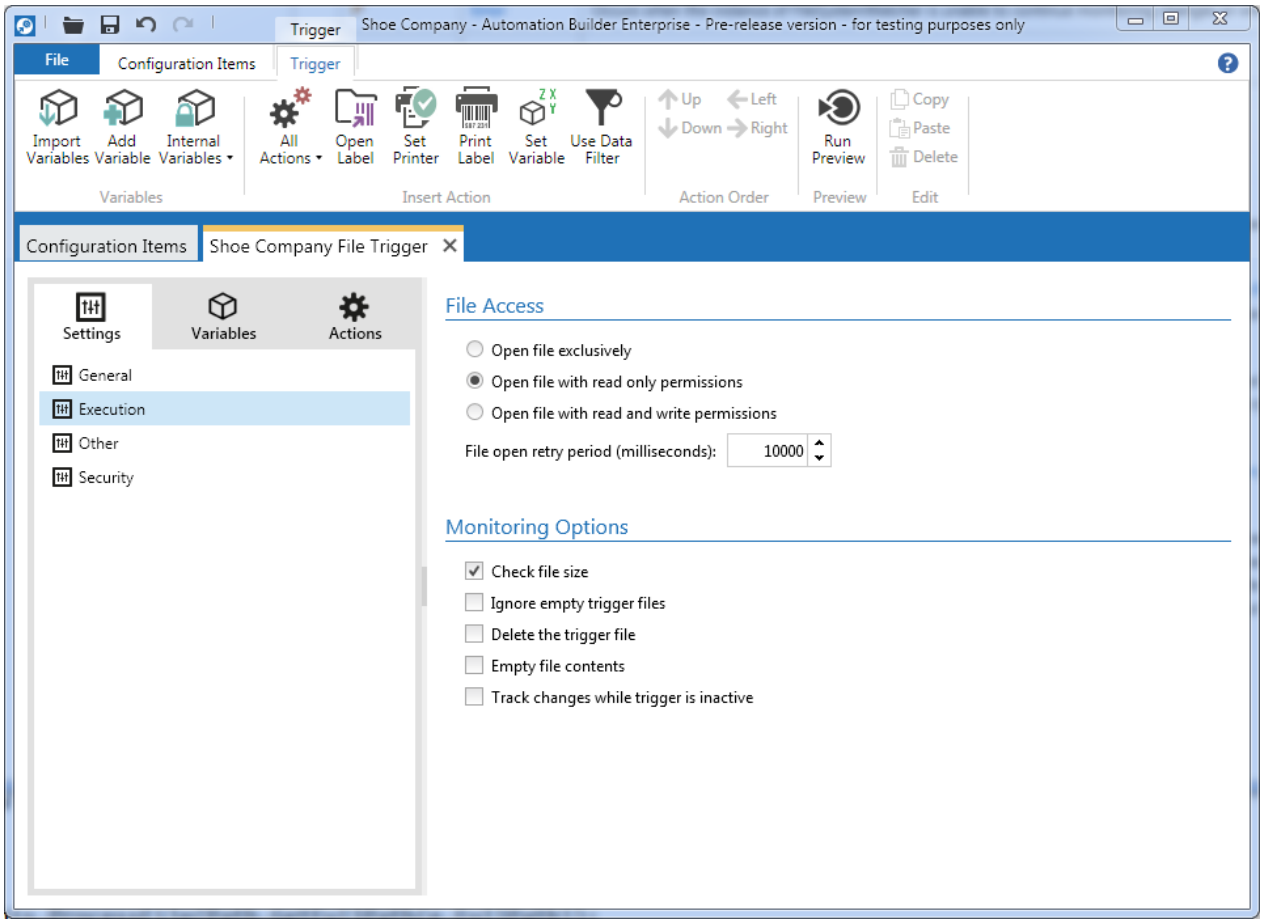


Figure 30 – Automation Builder - File Trigger - Execution Options

When file change is detected, the specified file is opened to read the data from it. You can specify how you want to open the file:

- **Open file exclusively**: When a file is opened in exclusive mode it means that no other process can open that file (for reading or writing) and only NiceLabel Automation has access to it while it reads from it.
- **Open file with read only permissions**: If NiceLabel Automation opens file with read only permissions other programs can still access it and write to it, but NiceLabel Automation can only read from it.
- **Open file with read and write permissions**: When this mode is used, NiceLabel Automation will be able to read from and write to the file.

If NiceLabel Automation cannot open the file it will wait for a period of time and then try again. This retry period can also be set in the **File Access** section. If the file still cannot be accessed after the specified time period, an error will be reported.

Monitoring Options

Monitoring Options section in the **Execution** category of the file trigger properties allows you to cover some more advanced scenarios for using file trigger:

- **Check file size**: If this option is selected, the size of the file is also checked. This option is selected by default when you create a new file trigger.
- **Ignore empty trigger files**: If there is no contents in the file (it has a size of 0 bytes), trigger will not initiate any actions or filter the data.
- **Delete the trigger file**: File is deleted after it is processed. This option helps you keep a folder clean of files that NiceLabel Automation already processed.
- **Empty file contents**: Contents of the file is removed after the data is processed. File will still remain on the disk.
- **Track changes while trigger is inactive**: If the trigger is stopped it can still track file or folder changes. The next time you start such file trigger, all the changes will be processed.

Serial Port Trigger

Serial port trigger connects to the serial port (RS232) and reads data from it. Usually this trigger type is used for processing data from barcode scanners, scales and other devices that know how to send data via serial port.

Serial Port Settings

When you add a serial port trigger you need to select which port should NiceLabel Automation monitor. You can select one of the 32 possible serial ports (COM1 to COM32). Automation Builder does not automatically detect serial ports that are used on the computer where it is running, but always displays a fixed list of 32 serial ports. The reason for this is that you might deploy a configuration to a different computer that might have completely different serial ports installed.

Standard serial port settings needed for two devices to communicate have to be set in order for NiceLabel Automation to work correctly:

- Speed in **bits per second**: The port speed and device speed must match.
- **Data bits**: The number of data bits in each character sent via serial port can be 5, 6, 7, 8, or 9 (which are very rarely used and NiceLabel Automation does not support). 8 data bits are used most of the time.
- **"Parity** is a method of detecting errors in transmission. When parity is used with a serial port, an extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or always even. If a byte is received with the wrong number of 1s, then it must have been corrupted. However, an even number of errors can pass the parity check." (Source: http://en.wikipedia.org/wiki/Serial_port (Wikipedia). The most commonly used setting for parity is **None**, where error detection is handled by the communication protocol.
- **Stop bits** are sent at the end of every character and allow the receiving device to detect the end of character.
- **Flow control** is the process of managing the rate of data transmission between two devices to prevent a fast sender from outrunning a slow receiver. There are hardware (CTS/RTS) and software (XON/XOFF) methods of flow control.

In some situations, usually when you are using virtual serial port, you might want to skip the port initialization phase. To do that just select the **Disable port initialization** property in the **Port Settings** section.

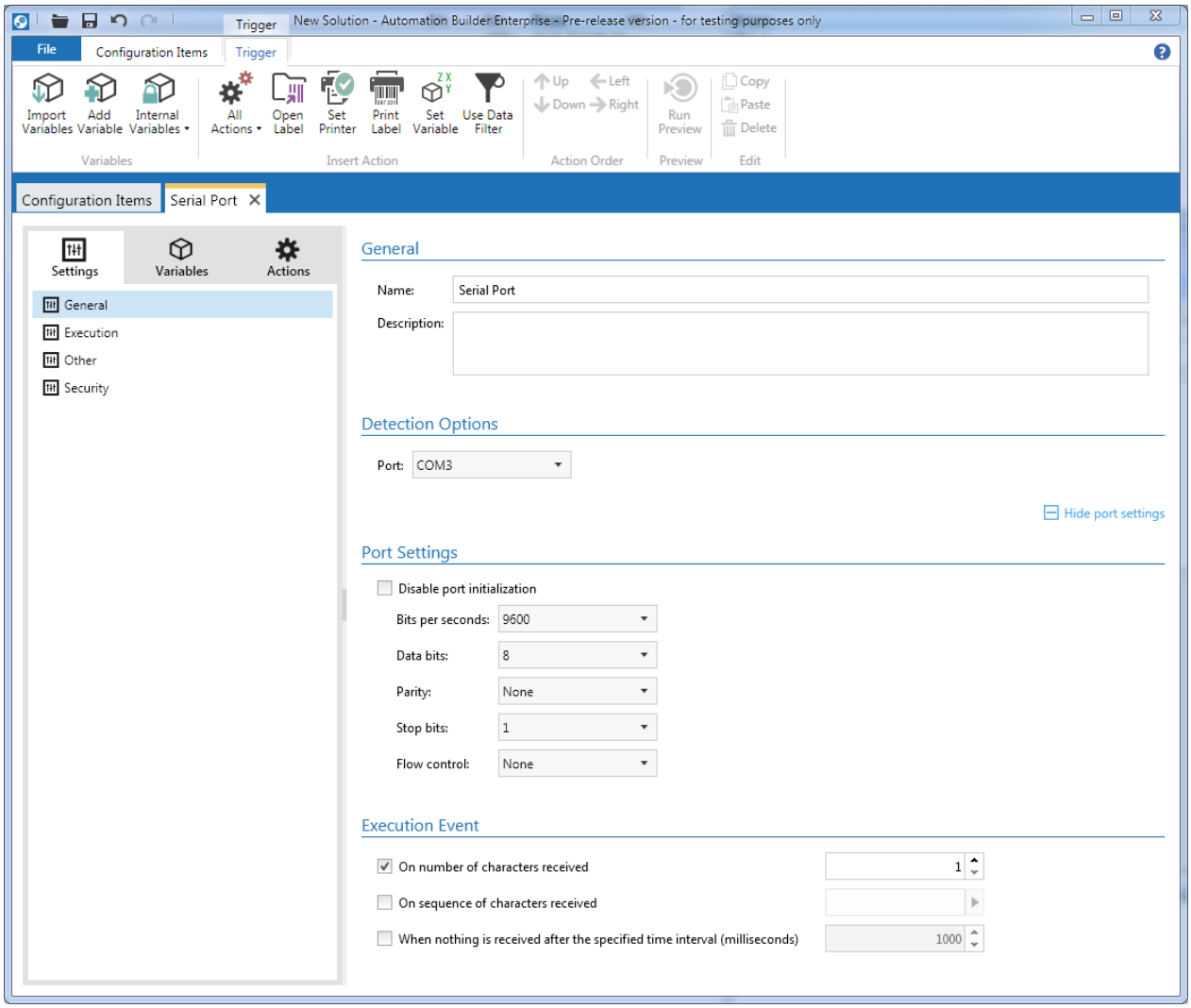


Figure 31 – Automation Builder - Serial Port Trigger - General Settings

Definition

"A virtual serial port is an emulation of the standard serial port. This port is created by software which enable extra serial ports in an operating system without additional hardware installation (such as expansion cards, etc.). It is possible to create a large number of virtual serial ports in a PC. The only limitation is the amount of resources, such as operating memory and computing power, needed to emulate many serial ports at the same time.

Virtual serial ports emulate all hardware serial port functionality, including Baud rate, Data bits, Parity bits, Stop bits, etc. Additionally they allow controlling the data flow, emulating all signal lines (DTR/DSR/CTS/RTS/DCD/RI) and customizing pin out. Virtual serial ports are common with Bluetooth and are the standard way of receiving data from Bluetooth-equipped GPS modules."

Source: http://en.wikipedia.org/wiki/Serial_port (retrieved 2013-03-14) (Wikipedia)

Defining When to Read Data from the Port

NiceLabel Automation can react to three serial port events and read data from it.

- **On number of characters received**: When a certain number of characters is received from the serial port, the contents of the serial port buffer will be saved into a file that can be used in actions. Just like for file trigger this file can be identified by the **DataFileName** variable.
- **On sequence of characters received**: When a certain sequence of characters is received, the trigger will read everything up to that sequence from the port. Sequence of characters that starts the process of reading the data will not be saved to the data file.
- **When nothing is received after the specified time interval**: Serial port trigger waits a specified time interval and then reads the data available on the serial port.

All three options can be selected at the same time.

Initializing Serial Port

If a device connected to the other side of a serial connection (like a weigh or a barcode scanner) needs some sort of a nudge to start sending data, you can send some initialization data to it. This initialization data is sent when the trigger is started.

Data Polling

Sometimes constant communication with a serial device is needed. In such cases data polling functionality of the serial port trigger can come in handy. If you enable **Use data polling**, and enter some data in the **Content** field, serial port trigger will send that data every time the provided time interval elapses.

Note

If some other application or service is already connected to the same serial port the trigger is trying to use, you will get an error, and the trigger will not start.

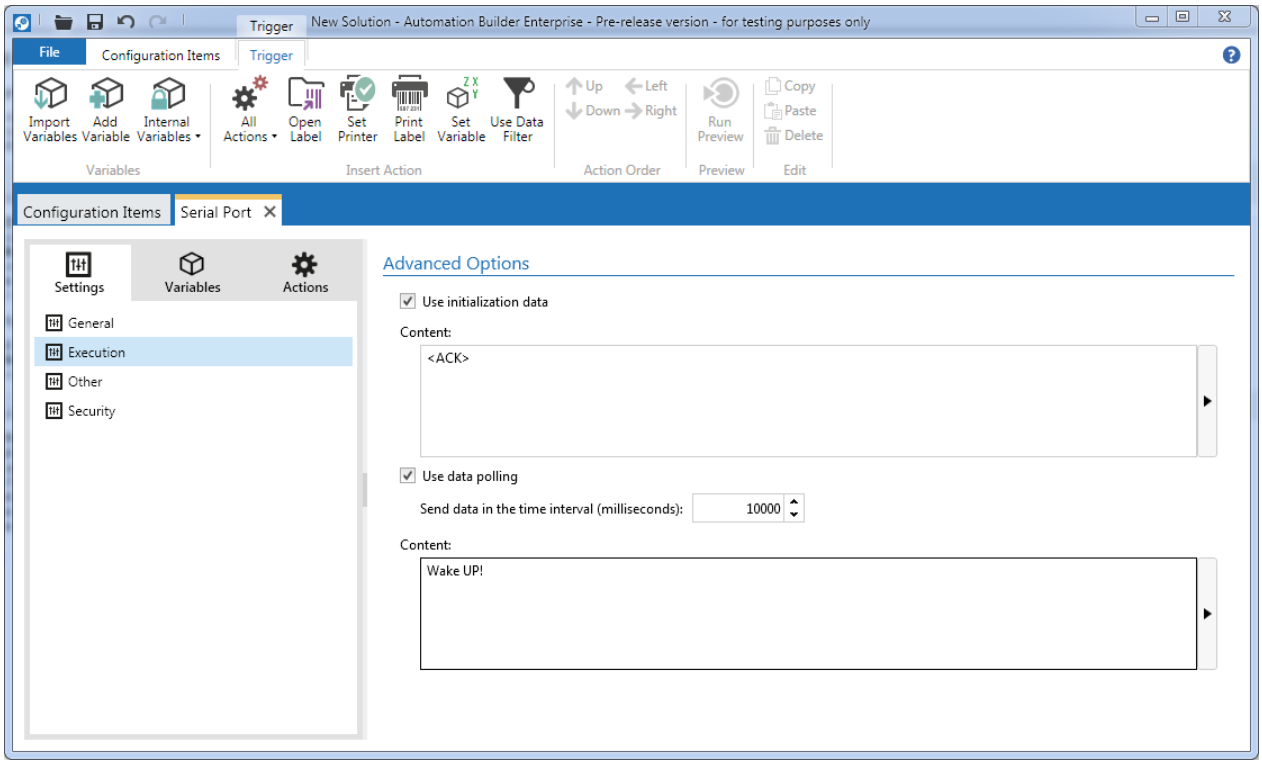


Figure 32 – Automation Builder - Serial Port Trigger - Execution Settings

Database Trigger

Database trigger reacts to changes in a database, for example adding a record , or updating an existing one. Since automatic detection of database changes is not possible, it queries the database for changes in specified time intervals. Because the trigger needs to query a database for changes, only databases that allow such queries, are supported. This means that some file based databases like CSV files or Excel files cannot be used.

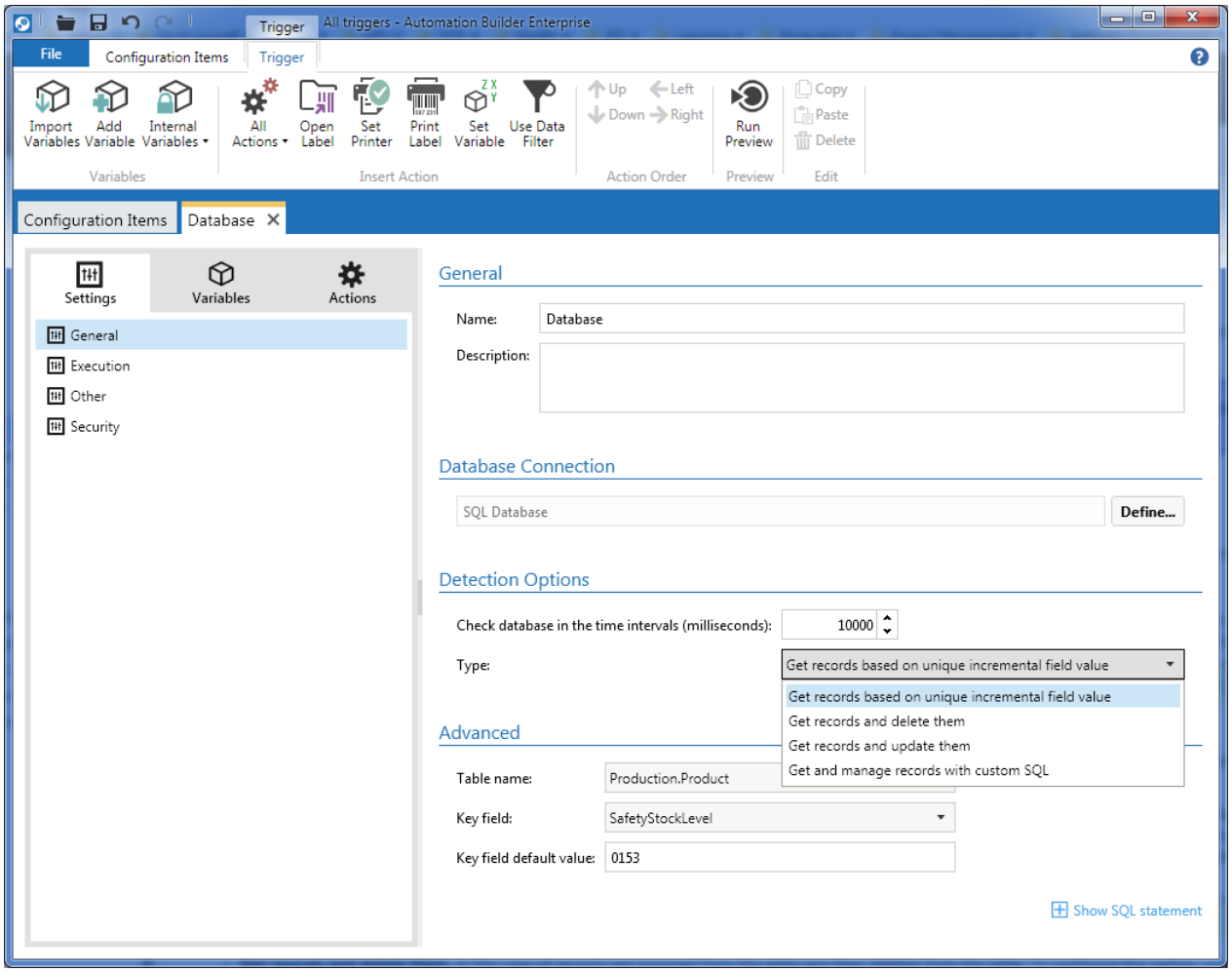


Figure 33 - Automation Builder - Database Trigger - General

Specifying Database Connection and Table Properties

First thing the database trigger needs is a connection to the database. To define a database connection, click the **Define…** button in the **Database Connection** section of the **General** category on the **Settings** tab.

A **Database** window is displayed, where you can select the database type you would like to use. In a screenshot below you can see I selected MS SQL Server. After you have selected the database driver, you will have to enter some information regarding the database, like database server location, user name and password if the database requires authentication, etc.

For the Microsoft SQL Server I clicked the **Define** button which opened another window, where I selected the server name, defined which authentication I want to use (since I used Windows authentication I did not have to enter user name and password), and selected a database name. Click on the **Test Connection** button confirmed the connection is correctly set.

Since every database type has its own set of settings that need to be configured, the user interface for your particular type of database might look differently.

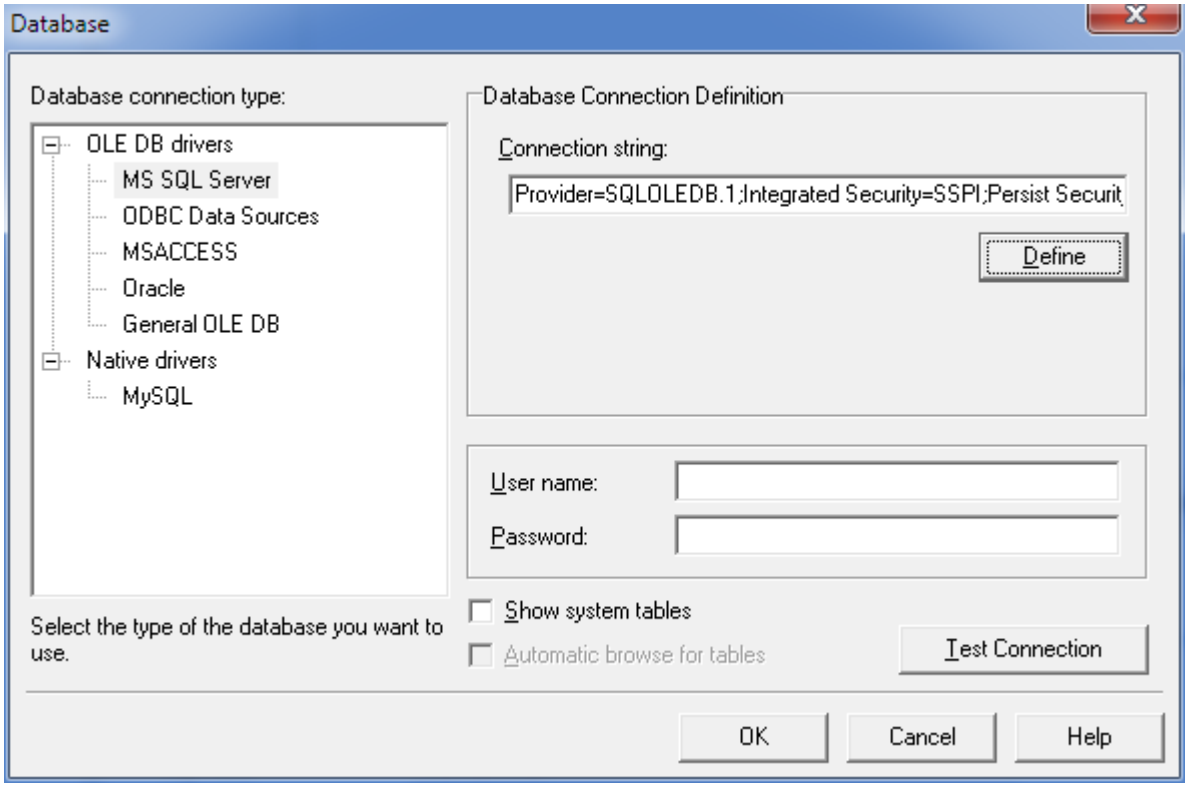


Figure 34 - Automation Builder - Database Trigger - Selecting Database Type

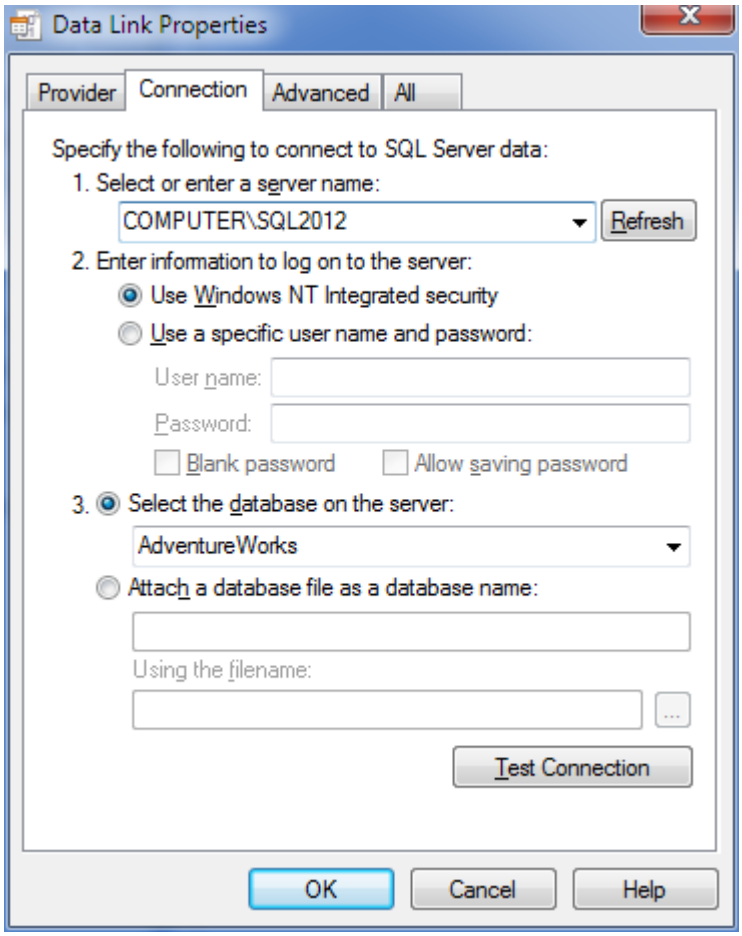


Figure 35 - Automation Builder - Database Trigger - Database Connection Properties

Detecting Changes

As mentioned previously, NiceLabel Automation needs to check the database periodically in regular intervals, because automatic detection is not possible.

To define the time interval for database check, enter the appropriate value In the **Check database in time intervals (milliseconds)** field. The value accepts milliseconds, so if you want to poll the database every 5 seconds, enter 5000 (milliseconds).

When the interval elapses, NiceLabel Automation queries the database. There are four different detection types and possible consequences:

- **Get records based on unique incremental field value**: In this case the database trigger will monitor auto-incremental numeric field in the table that you defined as the key field. NiceLabel Automation will remember the field's value for the last processed record. At the next interval only the records with values greater than the previously used will be returned. To correctly configure this option, select the table name, choose the auto-incremental key field and enter the starting value for the field (Key field default value).

Note

Internal variable KeyField is used as a reference to the current value of key field in SQL statements.

- **Get records and delete them**: All records that match the criteria will be returned and then deleted from the database. When this option is selected all you have to select is the table name. Key fields are displayed for informational purposes only.
- **Get records and update them**: Records will be returned and before/after the actions have finished the execution, they are updated. This option allows you to for example update a field in the database that indicates whether or not certain record was already processed / printed. When using this type of database trigger, you have to select the update field and the update value.
- **Get and manage records with custom SQL**: For all three previous options NiceLabel Automation automatically generates a SQL statement and you cannot change it. This option allows you to manually enter search and update SQL statements. Even though the property is called **Update SQL statement**, you can still use the delete statement.

Data Preview

Database trigger has a nice little feature that allows you to preview which records trigger will actually affect.

When you click the **Show SQL statement** link at the right bottom corner, a SQL statement section is displayed. The contents of this section depends on the type of database trigger you have selected previously. For **Get records based on unique incremental field value** you only get the **Search SQL statement** field, while for the other three options you also get the **Update SQL statement**.

You can save and test SQL statements. If you click the **Test** button, **Data Preview** pane will be shown on the right side of the window. There you enter the test values for the key fields and execute the SQL statement you want to test. If you check the **Simulate execution**, database records will not be really updated. A list of records will be displayed in a grid and below the grid the result of a query will also be shown. Result can be a message indicating that SQL statement was successfully executed and how many records were returned or affected. Or an error message will be displayed hinting what went wrong. For example you might get an error like this:

An error occurred in the execution of the SQL statement.

System error message: The DELETE statement conflicted with the REFERENCE constraint “FK_ProductInventory_Product_ProductID”. The conflict occurred in database “AdventureWorks”, table “Production.ProductInventory”, column ‘ProductID’. The statement has been terminated.

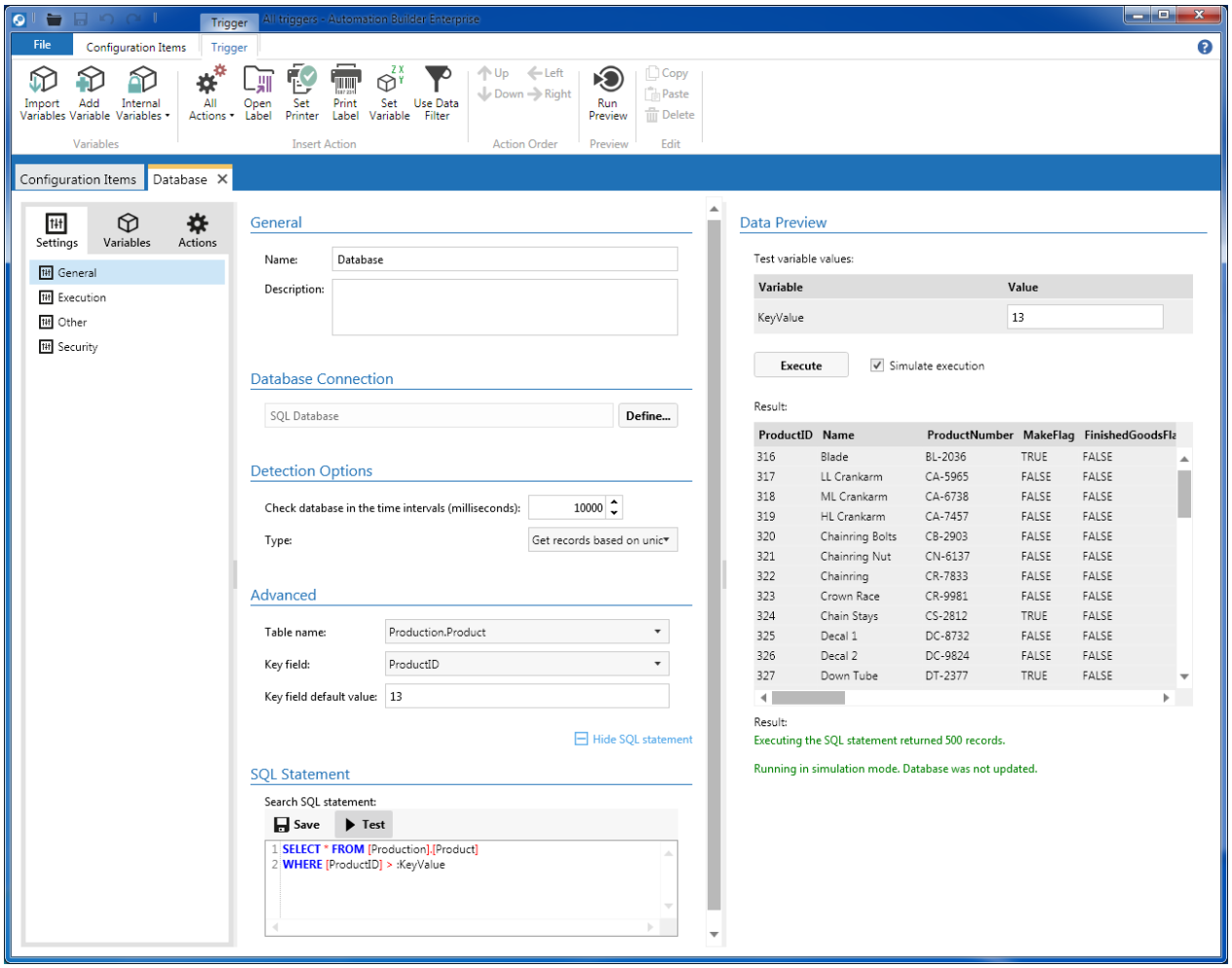


Figure 36 - Automation Builder - Database Trigger - Data Preview

Defining When Database Should be Updated

In the **Execution** category of the database trigger properties you can select when the database should be updated. You have two possibilities:

- Before processing actions
- After processing actions

The success of action execution does not affect the process of updating the database. If an error occurred during the print process, the database would still be updated. If you want to update the database only after the actions were executed without errors, you should update the database manually by using **Execute SQL statement** action.

TCP/IP Server Trigger

TCP / IP Server trigger uses Transmission Control Protocol/Internet Protocol (TCP/IP) to receive data that can then be used in filters and actions.

The most important property of the trigger is certainly the **Port** number. Port number represents the address of the TCP/IP trigger. Automation Builder provides a default value when you create new TCP/IP trigger, but you can always change it. If you have firewall enabled (and you should have), then you have to make sure that selected port is opened in the firewall configuration.

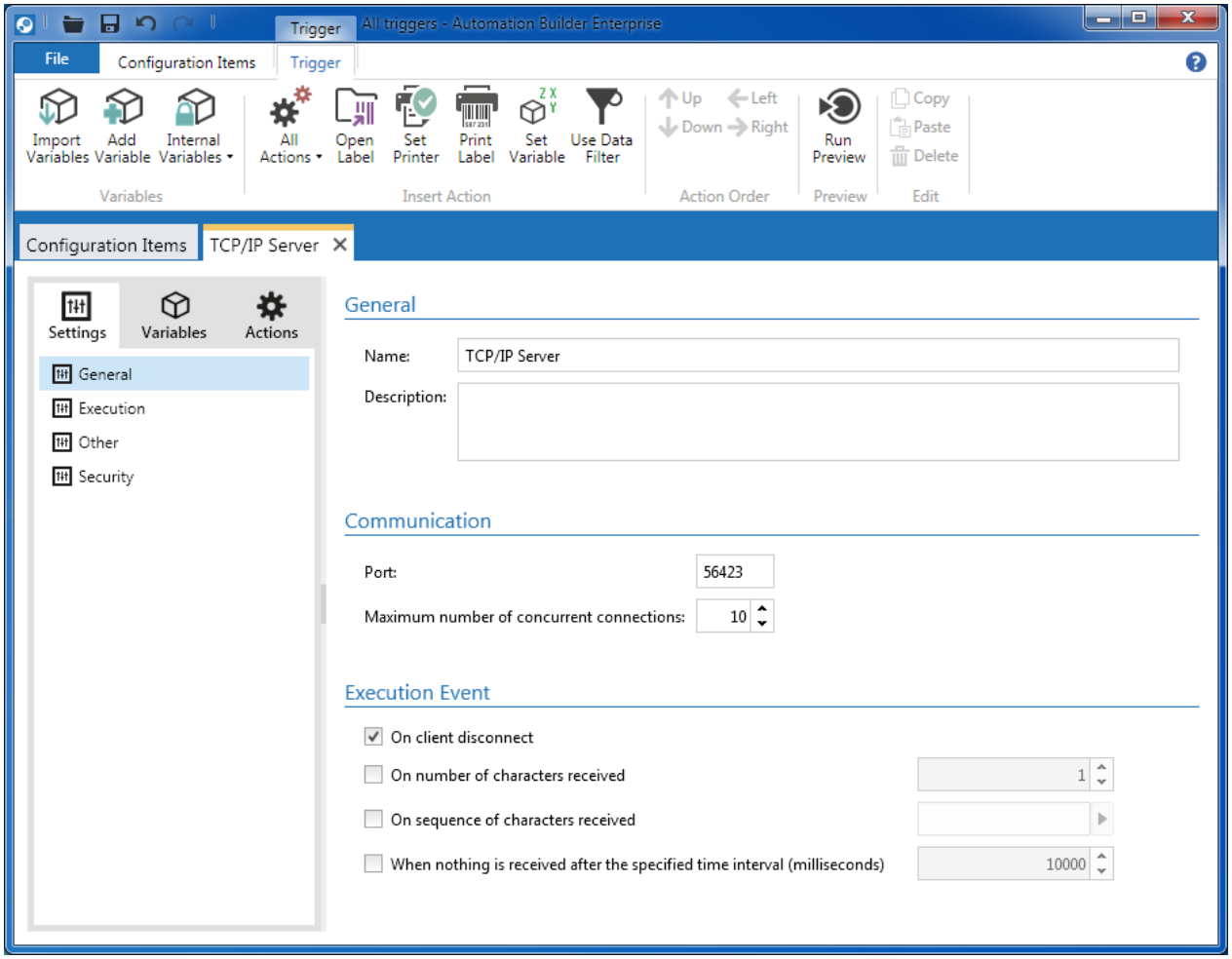


Figure 37 - Automation Builder - TCP/IP Trigger - General Settings

Tip

Some of the port numbers are so called well-known ports and registered ports. These port numbers are used by different system services and applications that registered their usage, so it might be good idea to avoid using them. For more information see:

- http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers (Wikipedia)
- https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt (Internet Assigned Numbers Authority (IANA))

Note

If the computer you are running NiceLabel Automation on, supports multi-homing (several IP addresses on one or more network cards), then NiceLabel Automation will react to events on specified port on all IP addresses. This is true for TCP/IP Server trigger, HTTP Server trigger, and Web Service trigger.

Restricting Number of Connections

Depending on the hardware capabilities of the computer running NiceLabel Automation you might want to restrict the number of concurrent connections to the trigger. **Maximum number of concurrent connections** field in the **Communication** section is used for these purposes. By default NiceLabel Automation allows 10 concurrent connections.

Defining When to Read Data from the TCP/IP Port

NiceLabel Automation can react to four TCP/IP port events and read data from it.

- **On client disconnect**: When client breaks the connection to the trigger, data is read from the port. If you want to send some feedback to the client during action execution (using **Send data to TCP/IP** action and setting its **Reply to sender** property), then do not use this option.
- **On number of characters received**: When a certain number of characters is received from the port, the contents of the port buffer will be saved into a file that can be used in actions. Just like for file trigger this file can be identified by the **DataFileName** variable.
- **On sequence of characters received**: When a certain sequence of characters is received, the trigger will read everything up to that sequence from the port. Sequence of characters that starts the process of reading the data will not be saved to the data file.
- **When nothing is received after the specified time interval**: TCP/IP trigger waits a specified time interval and then reads the data available on the port.

All four options can be selected at the same time.

Restricting Access

If you want to restrict access to the port you have used in your TCP/IP trigger for security or other reasons, you can define which hosts are allowed or denied connection to the trigger.

Both lists allow you to enter IP addresses or host names where each entry needs to be in its own line.

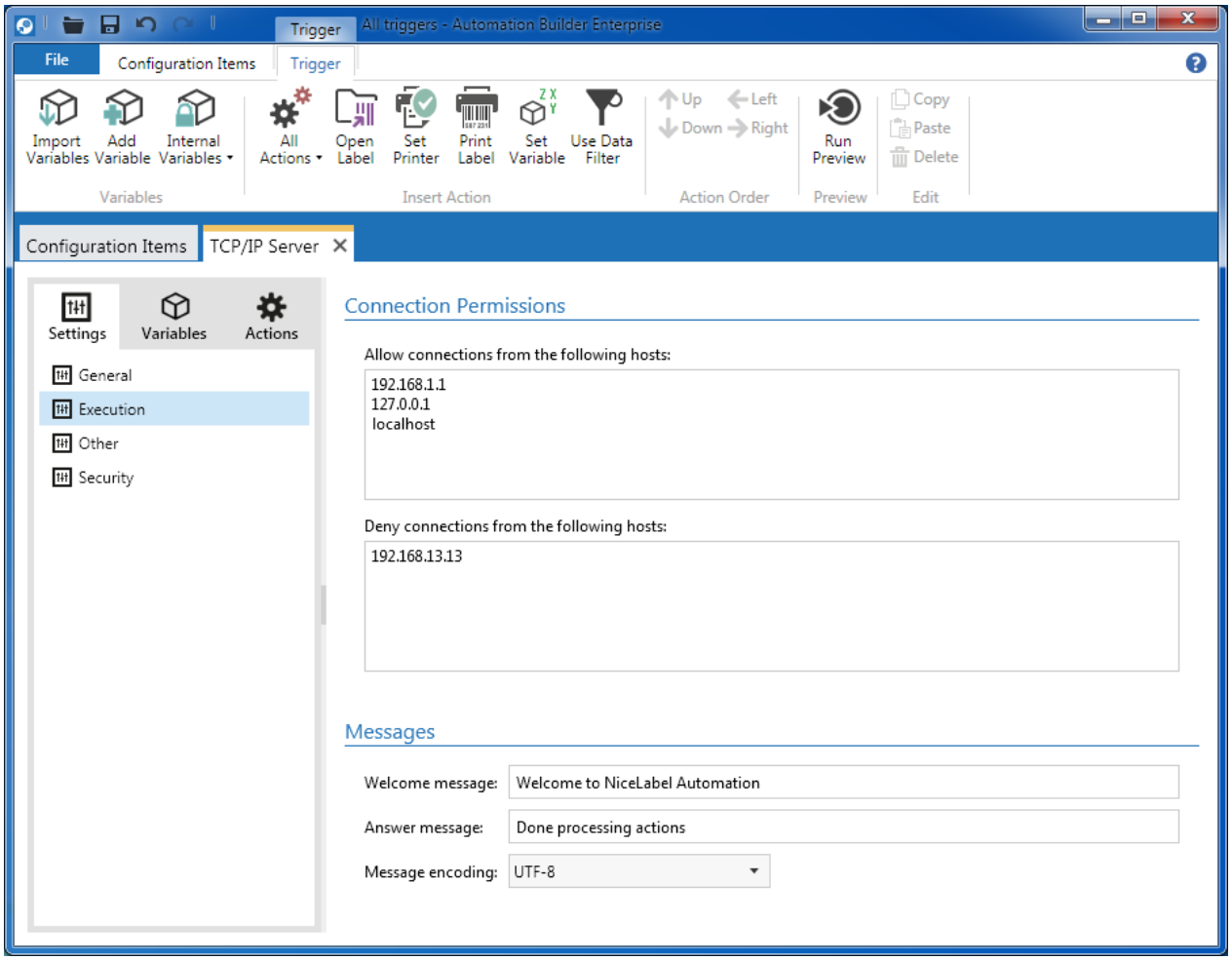


Figure 38 - Automation Builder - TCP/IP Trigger - Execution Settings

Communication with Clients

If you specify a **Welcome message**, then NiceLabel Automation will send that message to the client every time it connects to the TCP/IP trigger.

Answer message is sent to client when the actions have finished with the execution. Answer message can be sent only if the client has kept the connection to the trigger open.

Note

Welcome message and Answer message can only be fixed values (you cannot use variable values).

Messages can be encoded as UTF-8 (default), UTF-16 or ASCII characters.

HTTP Server Trigger

HTTP Server trigger uses Hypertext Transfer Protocol (HTTP) to receive data and provide feedback. Just like with TCP/IP port be careful which port number will you use, as some might already be taken. Web servers usually listen on ports 80 or 443.

When using port other than 80 (usually this port is open, as you need it for web browsing), make sure that firewall allows traffic through that port.

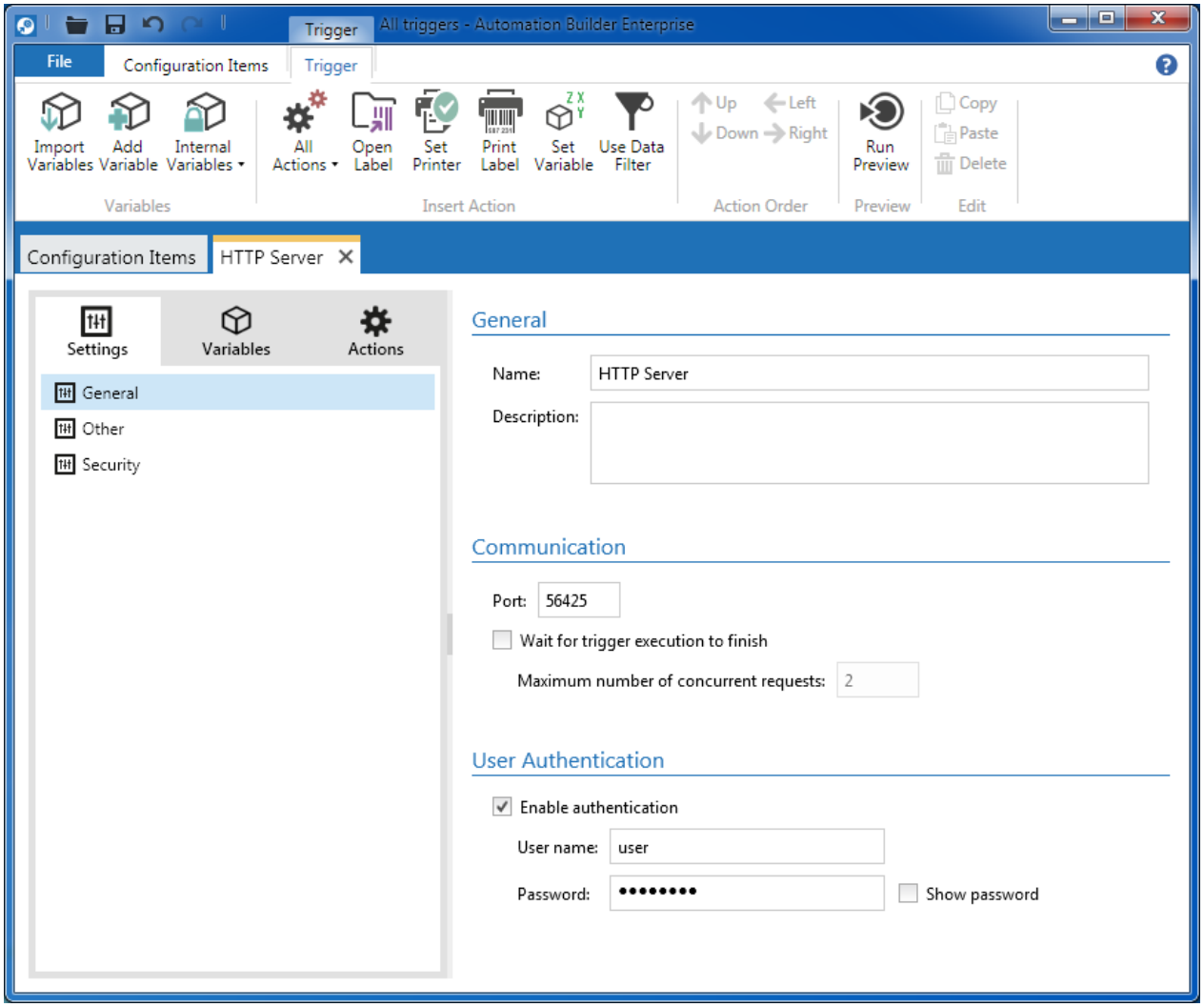


Figure 39 - Automation Builder - HTTP Trigger

Request Response Mechanism

Definition

"HTTP functions as a request-response protocol in the client-server computing model. A web browser,

for example, may be the client and an application running on a computer hosting a web site may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body."

Source: http://en.wikipedia.org/wiki/Http (Wikipedia)

NiceLabel Automation acts as the HTTP server, so it needs to send a numeric response code back to the client. If you check **Wait for trigger execution** to finish, NiceLabel Automation will wait for actions to finish execution and only then return response code. Make sure you checked **Supervised printing** in the **Other** category of the trigger properties, if you want for NiceLabel Automation to wait for the printing process to finish as well, before sending the response.

If everything goes well, response code is 200 (OK), if there is an error during the action execution, response code is 500 (Internal Server Error), and if user authentication is enabled and NiceLabel Automation cannot authenticate the user, response code 401 (Unauthorized) is returned. (Wikipedia)

Restricting Number of Connections

Maximum number of concurrent requests field in the **Communication** section is used for restricting the number of clients trying to connect to the trigger at the same time. By default NiceLabel Automation allows two concurrent connections. You can set the limit only when **Wait for trigger execution to finish** is checked because otherwise response is sent to the client immediately and connection for the next client is available very quickly, so there is no need for the concurrent requests. How high you set the limit of concurrent requests depends on the hardware capabilities of the computer running NiceLabel Automation and the configuration of the trigger. If complex data filters are involved and/or there are long-running actions that need to be executed, you might want to keep the number of concurrent requests low.

Authenticating Users

To protect the trigger with user name and password so that only clients that have this information can connect, check the **Enable authentication** flag in the **User Authentication** section.

Web Service Trigger

Web Service Trigger allows NiceLabel Automation to react to data coming via SOAP (Simple Object Access Protocol - (Wikipedia)) web service. It relies on XML as its message format and its interface is described in the WSDL (Web Services Description Language) document that is available for every Web Service trigger.

Like TCP/IP Server trigger and HTTP trigger, Web Service trigger also allows you to define a maximum number of concurrent calls. When you add new Web Service trigger to the configuration the maximum number of concurrent calls is set to 10.

Web Service trigger supports two WSDL operation styles:

- **RPC**
- **Document**

The operation style indicates whether the operation is RPC-oriented (messages containing parameters and return values) or document-oriented (message containing document(s)). See http://www.w3.org/TR/wsdl (World Wide Web Consortium (W3C)) for more information about SOAP operation styles.

Definition

"A web service is a method of communication between two electronic devices over the World Wide Web. A web service is a software function provided at a network address over the web or the cloud, it is a service that is "always on" as in the concept of utility computing."

Source: http://en.wikipedia.org/wiki/Web_service (Wikipedia)

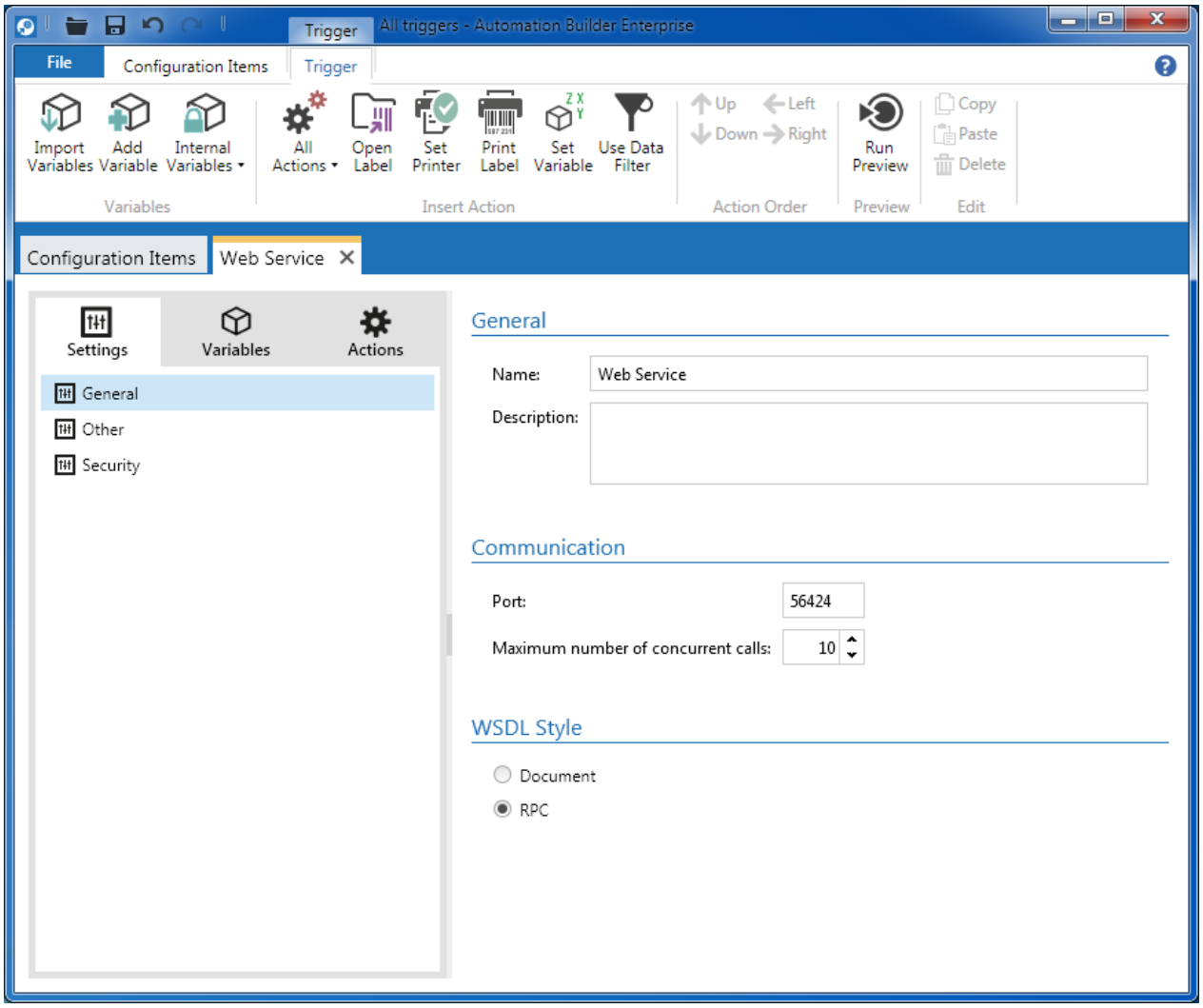


Figure 40 - Automation Builder - Web Service Trigger

If you configure the trigger to run on port 56424 and then start the trigger in Automation Manager, you can check the web service WSDL document by entering the following address in your favorite web browser: http://localhost:56424/.

Web service in NiceLabel Automation has only one method – **ExecuteTrigger** - that takes two parameters:

- **Text**: This string parameter should contain the data that NiceLabel Automation should process and can be structured as regular CSV values, XML or some other format that you can parse in NiceLabel Automation using data filters.
- **Wait**: Boolean parameter indicates to the trigger whether the client application wants to wait for the trigger execution to finish and to provide feedback to the client.

Method also has two optional output parameters that have a value if **Wait** parameter is set to true:

- **ExecuteTriggerResult** contains an integer value indicating if an error occurred. If there are no errors, 0 is returned.

- **ErrorText**: If an error occurred during action execution, this parameter should contain a descriptive error message, otherwise it is empty.

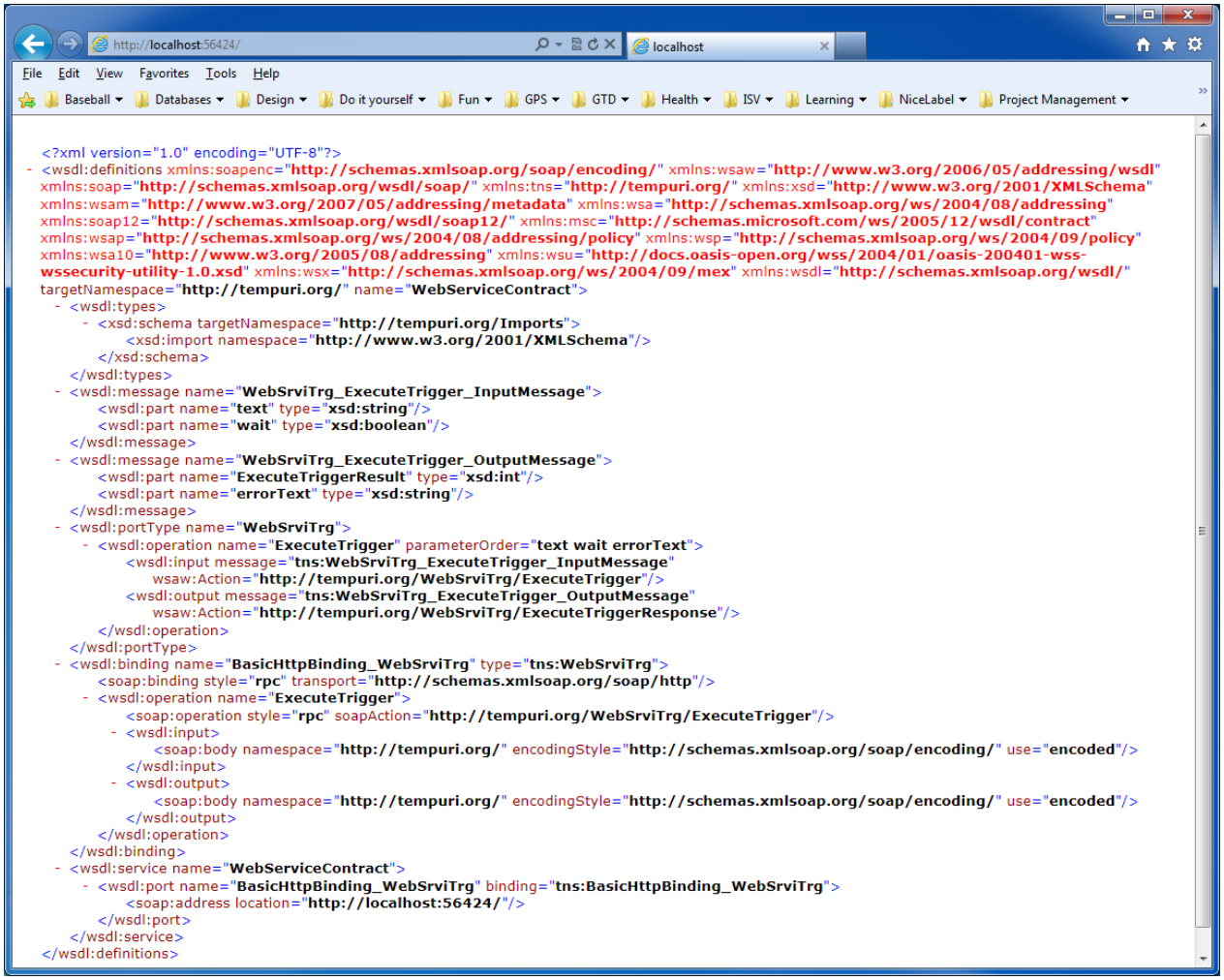


Figure 41 - Web Service Trigger - WSDL Document

How Can I Communicate with Web Service Trigger from My Application

Below you can see a short Python script that sends a sequence of NiceCommands to the web service, waits for the execution to finish but does not check the result of the call to the web service.

Note

Python script uses suds module. For more information about it check https://fedorahosted.org/suds/

```python
from suds.client import Client

msg = 'LABEL "E:\\Triggers\\ISBN.lbl"\nSET "5 Digit"="55999"\nSET "ISBN Num"="9781935182610"\nPRINTER "CAB A4+M 300DPI"\nPRINT 100'
url = 'http://localhost:56424/?wsdl'

client = Client(url)
r = client.service.ExecuteTrigger(msg, True)

print r
```

Figure 42 - Python Script - Call to Web Service Trigger

Filters

Filters

Filters in NiceLabel Automation process data that comes into the application via triggers and only extract data that is useful for the user and can be used in actions or printed on labels. NiceLabel Automation can process structured text like CSV files (**Structured Text Filter**), unstructured data (**Unstructured Data Filter**), and XML formatted data (**XML Filter**) .

Filter	Automation Easy	Automation Pro	Automation Enterprise
Structured Text Data Filter	X	X	X
Unstructured Data Filter	X	X	X
XML Data Filter		X	X

Table 3 - Filters in different editions of NiceLabel Automation

Understanding Filters

In my opinion filters are the most complex part of NiceLabel Automation and as such require a bit more effort to understand them. Besides reading NiceLabel Automation help file, user guide, and this book, I would also suggest studying the samples that come with NiceLabel Automation installation.

Structured Text Filter

Structured Text Filter allows you to parse and transform data from a comma-separated values (CSV) files or files with fixed column widths.

Definition

"A comma-separated values (CSV) file stores tabular data (numbers and text) in plain-text form. Plain text means that the file is a sequence of characters, with no data that has to be interpreted instead, as binary numbers. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by some other character or string, most commonly a literal comma or tab. Usually, all records have an identical sequence of fields."

Source: http://en.wikipedia.org/wiki/Comma-separated_values (Wikipedia)

Below you can see a sample of both, a CSV file and fixed column width file.

CSV File

```
Product_ID;Code_EAN;Product_desc;Package
CAS006;8021228110014;CASONCELLI ALLA CARNE 250G;6
PAS501;8021228310001;BIGOLI 250G;6
PAS502GI;8021228310018;TAGLIATELLE 250G;6
PAS503GI;8021228310025;TAGLIOLINI 250G;6
PAS504;8021228310032;CAPELLI D'ANGELO 250G;6
PAS505;8021228310049;PAPPARDELLE 250G;6
PAS506GI;8021228310056;SFOG.LASAGNE 250G;6
```

Fixed Column Width File

```
NiceLabel           1       Label design and printing
NiceForm            2       Data-entry application design
NiceWatch           3       Data-detection and connectivity
NiceData            4       Database management
NiceMemMaster       5       Font and graphics download
NiceDrivers         6       True Windows thermal-transfer printer drivers
```

Adding New Structured Text Filter

There are two ways to create new Structured Text Filter. You can click the **Structured Text Data Filter** button on the **Configuration Items** tab when there are no items yet in the configuration. Or you can click **Structured Text** button on the **Configuration Items** page in the ribbon.

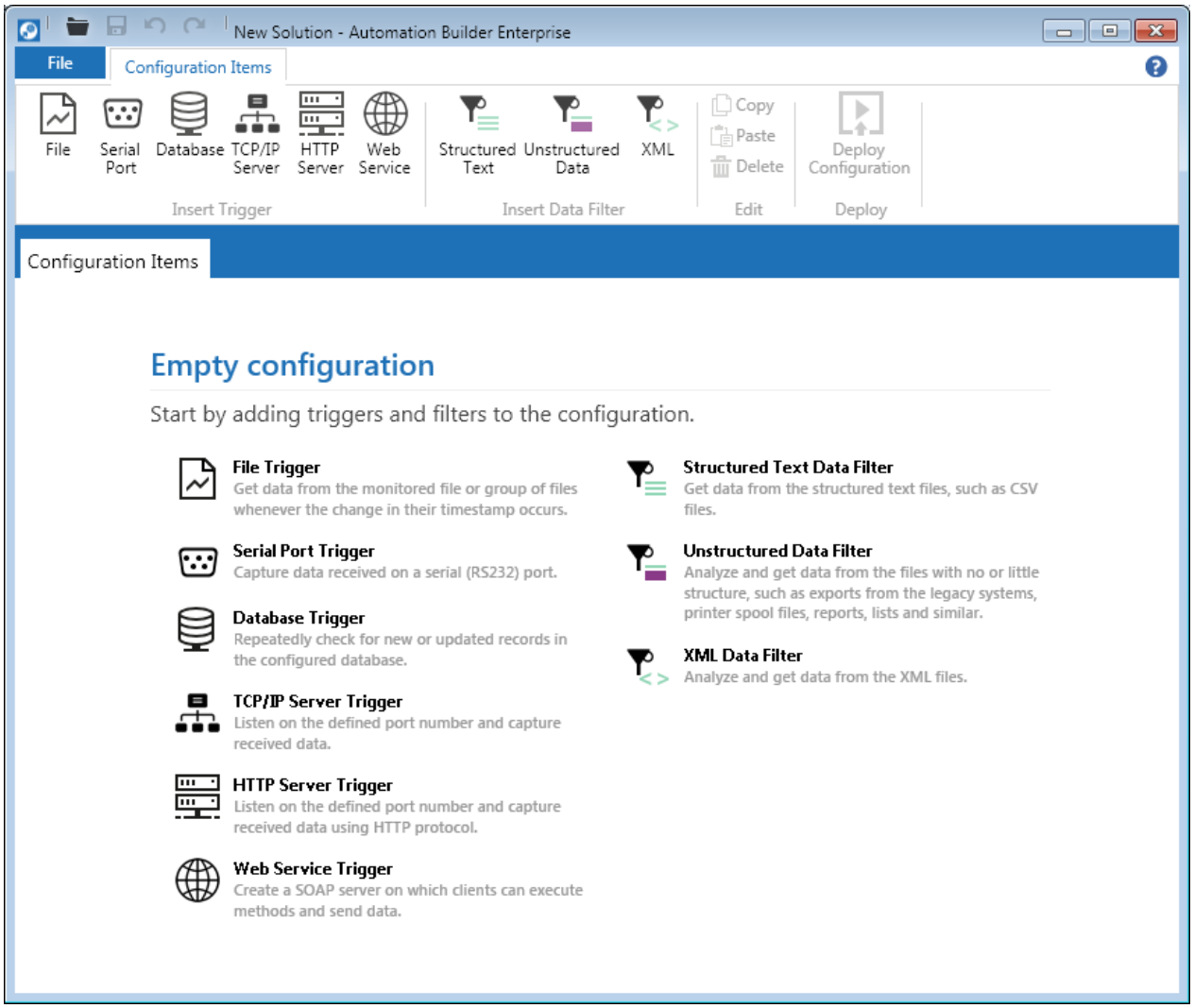


Figure 43 - Automation Builder - Configuration Items

Every filter (just like trigger and variable) has some basic properties like name and description.

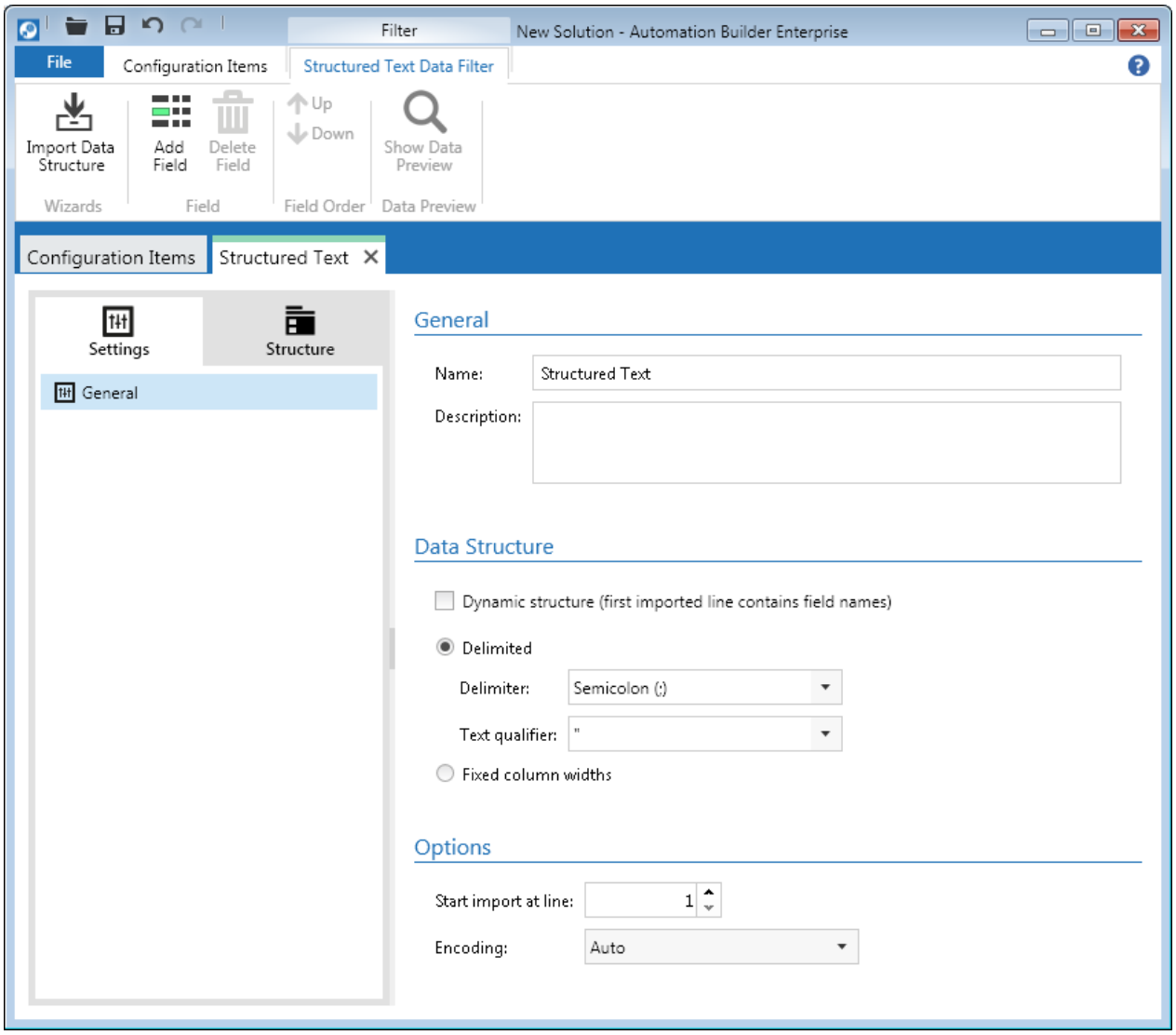


Figure 44 - Automation Builder - Structured Text Data Filter - General Settings

Defining Data Structure

Data structure is described by the fields, delimiter and text qualifier (when CSV file is used) or by the fields and their widths (when fixed column width file is used).

Automation Builder supports three ways of defining the data structure, that the Structured Text Filter can use.

Using Import Data Structure Wizard

Using Text File Wizard is the easiest way to define data structure. Wizard supports both types of files (CSV and fixed column width files). To start it click the **Import Data Structure** button on the **Structured Text Data Filter** tab in the ribbon.

In a series of screenshots below you can see how the wizard imports structure from a CSV file that uses semicolon as delimiter and double quotes as text qualifier.

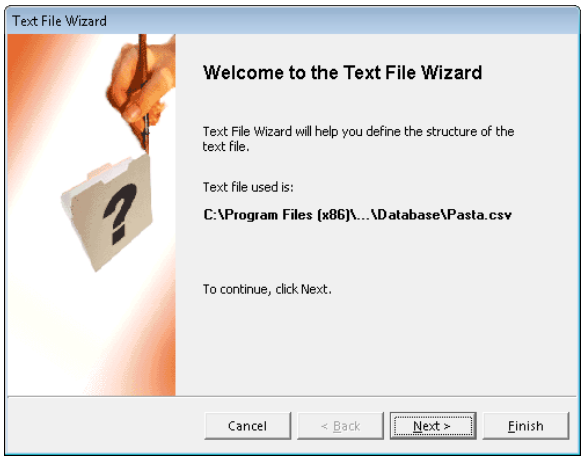


Figure 45 - Text File Wizard – Step 1 - Welcome

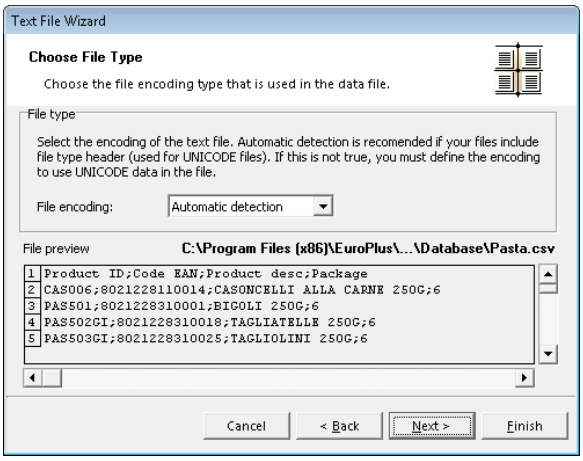


Figure 46 - Text File Wizard - Step 2 - Select File Type

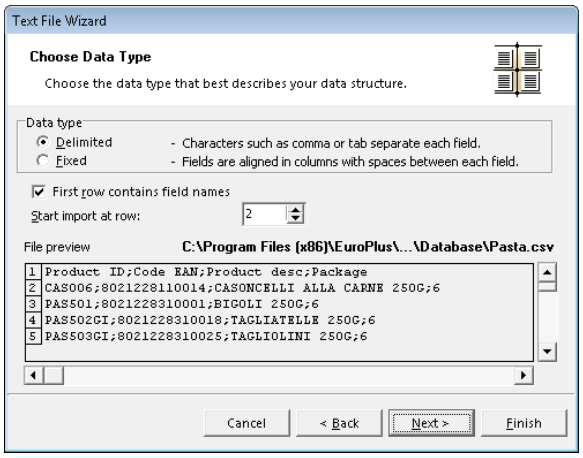


Figure 47 - Text File Wizard - Step 3 - Select Data Type

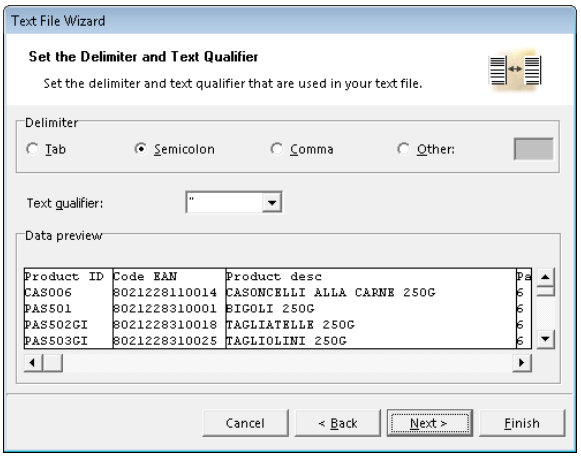


Figure 48 - Text File Wizard - Step 4 - Set Delimiter and Text Qualifier

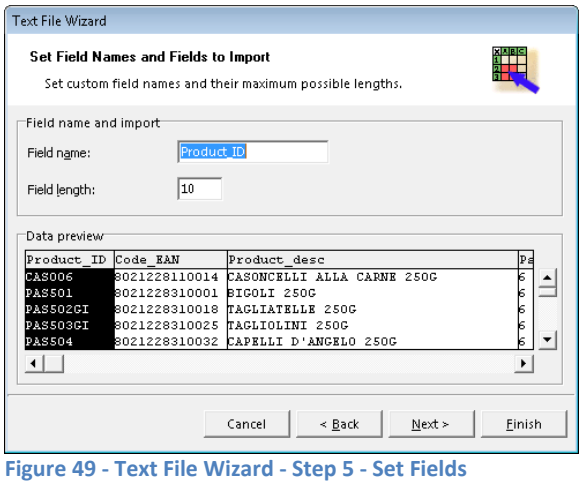


Figure 49 - Text File Wizard - Step 5 - Set Fields

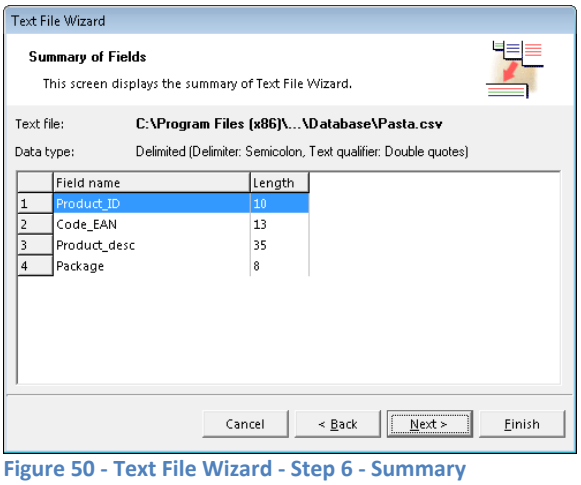


Figure 50 - Text File Wizard - Step 6 - Summary

After you click the **Finish** button you get a list of fields on the **Structure** pane.

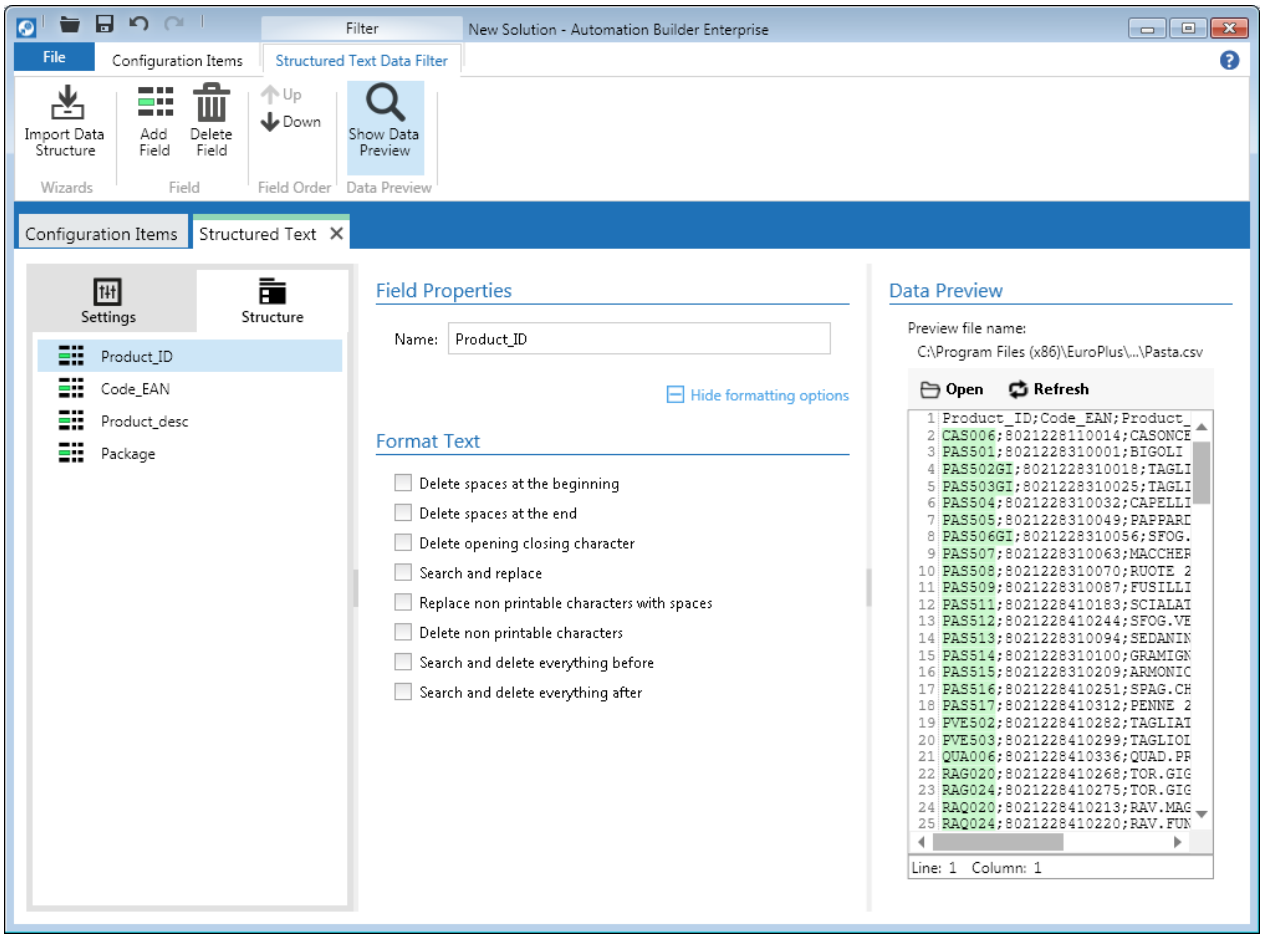


Figure 51 - Automation Builder - Structured Text Filter - Structure

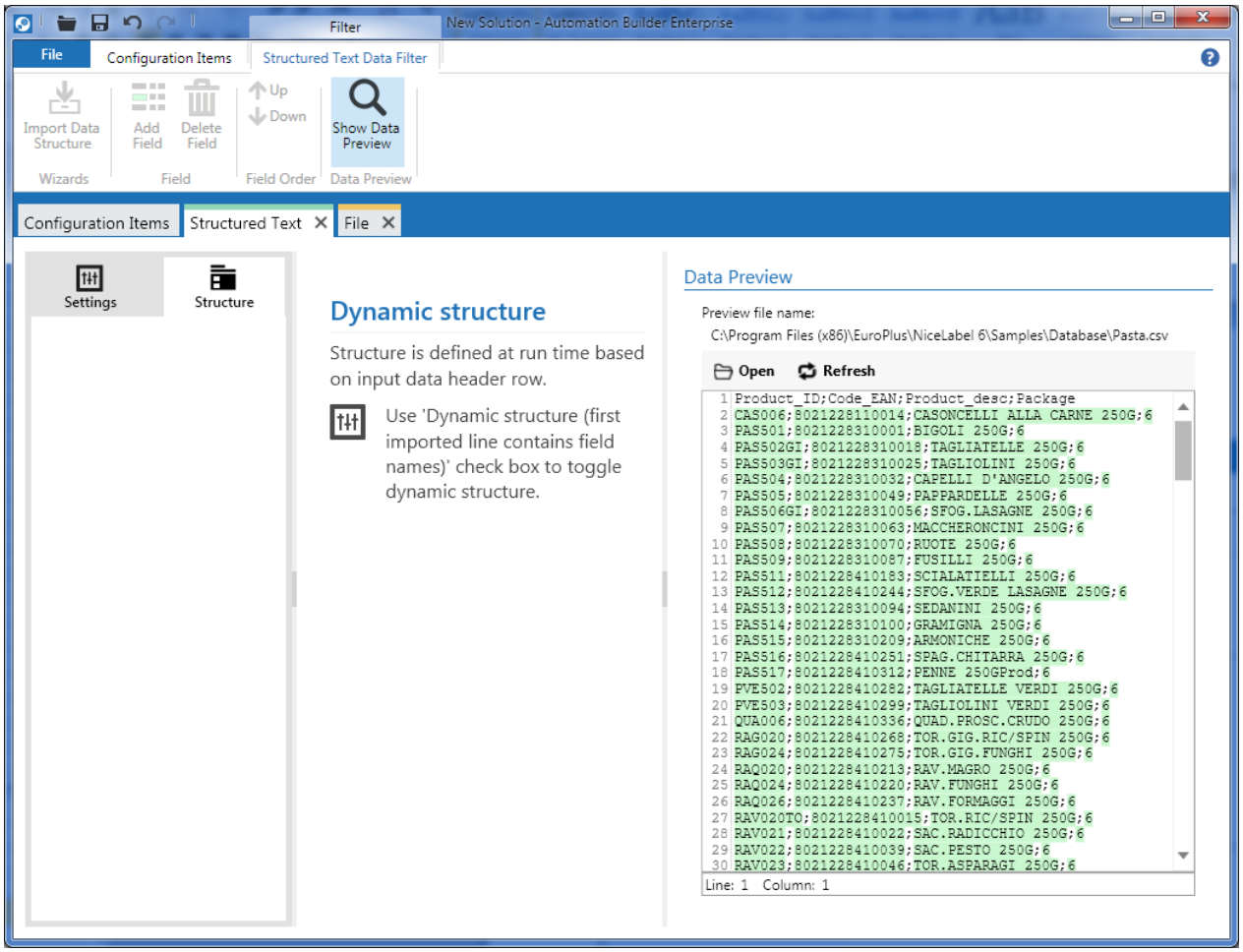
Dynamic Structure

When you want to use Structured Text Data Filter with different data files that do not have a fixed structure and number of fields change, you can use the **Dynamic structure (first imported line contains field names)** option. When this option is enabled filter will automatically figure out the structure of the file by reading the first line in the file that should contain field names. Fields in such files should be delimited (fixed column width files are not supported in this case).

When Structured Text Filter with dynamic structure is used in the **Use Data Filter** action you cannot map the fields with variables as this mapping will be done automatically. You just have to make sure that you give variables the same names that fields in the data file have. When **Print Label** action is then used as a child action under the **Use Data Filter** action, label variables will automatically get values from the filter fields if the names match.

Note

All other action that use variables will NOT automatically get their values. Automatic mapping works for Print Label action only.



Manually Defining Structure

You can also manually add fields to the structure by using the **Add Field** button in the ribbon. To delete a field from the list, select the field and press DELETE key or click **Delete Field** button in the ribbon.

Formatting Text

For every field in the data structure, NiceLabel Automation can execute several text formatting operations.

- **Delete spaces at the beginning**.
- **Delete spaces at the end**.
- **Delete opening** and **closing character**. If you have specified "(" as the opening character and ")" as the closing character, and field contains "*(_some data_)*", you will get "*_some data_*" as the value of the field.
- **Search and replace** strings. Just like in word processing applications like Microsoft Word, you define what you want to search for and what it should be replaced with. Additional options are available that narrow the search results:

- **Match case**: Letter case is taken into consideration while searching. For example "AAAA" is not the same as "aaaa".
- **Find whole words only**: searching for "end" will also find "pretend", "intended", "attend", "endless", if this option is not selected.
- **First occurrence only**: Only the first occurrence of the text you are looking for will be replaced with the new value. Other occurrences will be left alone.
- **Regular expressions**: Regular expressions provide a flexible and powerful method for processing text. There are lots of different implementations of regular expressions that usually differ in little nuances. Since NiceLabel Automation uses Microsoft .NET framework, it supports the same regular expressions as Microsoft .NET does. Be careful though, regular expressions are not for the faint of heart.
  If you click the **What are regular expressions?**, you will be taken to a Microsoft web site where you can read all about regular expressions in .NET Framework. (Microsoft)

- **Replace non printable characters with space**
- **Delete non printable characters**. Non printable or control characters are used to control the interpretation or display of text but do not have a visual or spatial representation. All the data in NiceLabel Automation is Unicode encoded and the Unicode standard assigns the following code points to control characters: from \U0000 to \U001F, \U007F, and from \U0080 to \U009F.
- **Search and delete everything before** a specified value. You can also remove the search string.
- **Search and delete everything after** the specified value. Search string can also be deleted.

If you select more than one option, text formatting will be performed in the same order as listed in the Automation Builder.

Note

Text formatting is NOT possible when dynamic structure is used.

Data Preview

Automation Builder also offers preview for selected data file. This way you can verify that filter is configured correctly as the **Data Preview** visually shows you value in the file for selected field.

To show or hide **Data Preview** pane, click **Show Data Preview** button in the ribbon.

Unstructured Data Filter

Unstructured data refers to information that does not have a predefined data model or does not fit into relational tables. Today lots of potentially usable business information originates in unstructured form (some say even 80 percent; (Wikipedia)). Examples of such unstructured data are:

- Email body
- Reports
- Documents
- Health Records
- Web Pages
- Print streams

NiceLabel Automation allows you to deal with such information using **Unstructured Data Filter**.

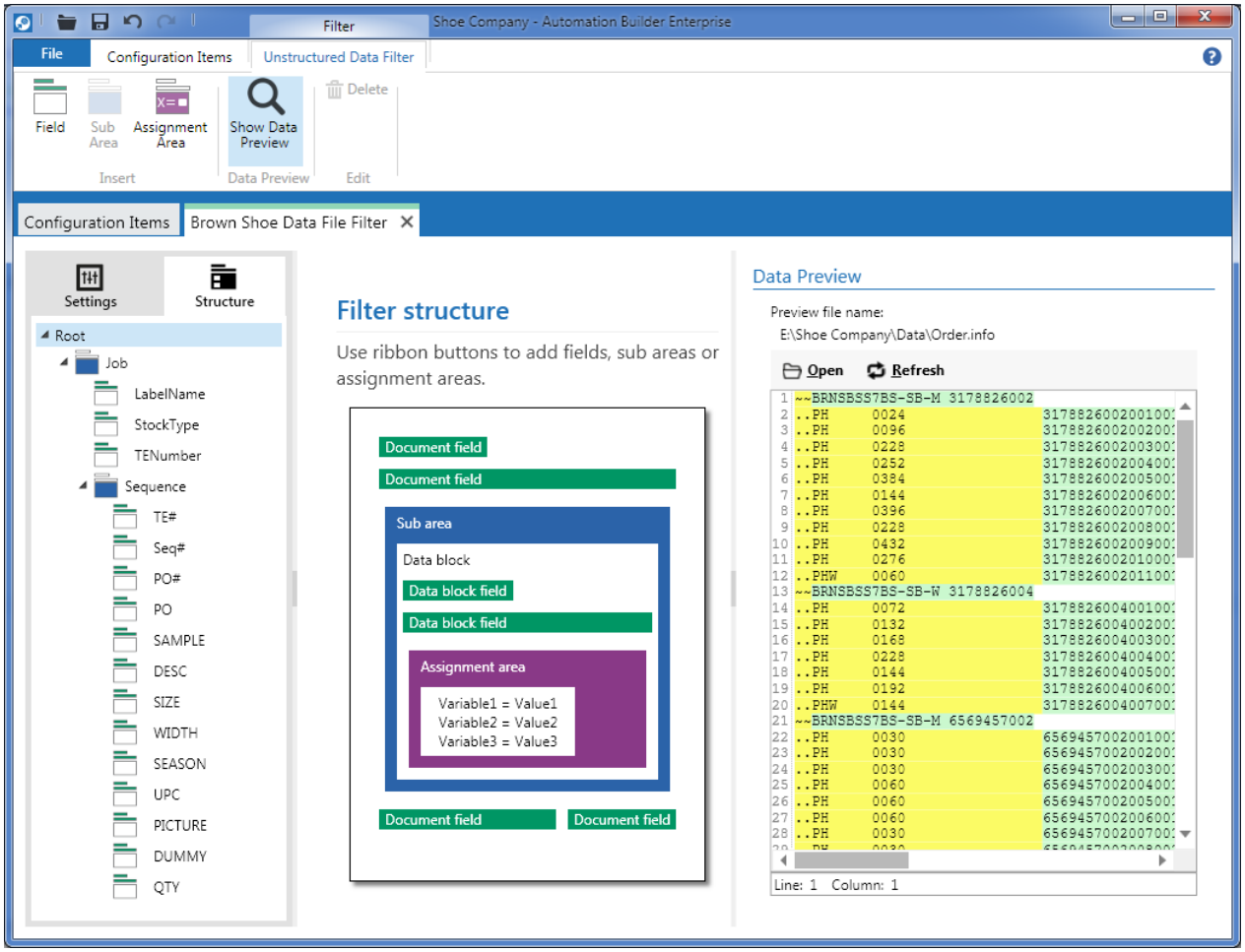


Figure 52 - Unstructured Data Filter

NiceLabel Automation uses some basic terms to describe parts of unstructured data documents.

Data Block

Data block is a sequence of characters or bytes inside a sub area or assignment area. The sequence can be inside one line or it can span across several lines.

Field

Field or document field is a piece of data (text or binary) defined by start and end position. It can be positioned anywhere in the document or inside a data block in a sub area or assignment area. To use value from a field (defined by the filter) in actions, you need to map it to a variable. Variable can then be used in actions.

There are three possibilities to define field start:

- **Position in document or data block**: Field starts at fixed character in fixed line number. If the field is defined at the root of the filter structure then the line and character number are relative to the document. If the field is defined inside a data block then the row and character numbers are relative to that data block.
- **End of document or data block**: Field can be found at a certain position from the end of document or data block. So if the field is located at the fifth character, two lines from the end of the document, you would enter 2 in the **Lines** field and 5 in the **Characters** field.
- **Find string from start of document or data block**: Start position is defined by the position of a certain string you are looking for. When the string Is found in the provided data, the next character (or byte for binary contents) after the string determines the start position of the field. The string that filter is searching for, is not included in the extracted data. Searching can start from the beginning of the document/data block (line 1 and character 1), or from some other location inside document/data block that is defined by the line and character number. If there are more than one occurrences of the string, you can also define which one should be used by setting the **Occurrence** property. You can also set the **offset** from the string where the field should be extracted. The offset can be a positive or a negative number. If you enter a negative number, filter will extract also data in front of the string or the string itself.

Automation Builder offers six possibilities to define the field end position:

- **Position in document or data block**: Field ends at fixed character in fixed line number. If the field is defined at the root of the filter structure then the line and character number are relative to the document, otherwise they are relative to the data block.
- **Length**: Field value is set by extracting the specified number of lines and characters from the field start position.
- **Find string from start of document or data block**: End position of the field is defined by the position of a certain string from the start of document or data block. When this string Is found in the provided data, the next character after the string determines the end position of the field. The string that filter is searching for, is not included in the extracted data. Searching can start from the beginning of the document/data block (line 1 and character 1), or from some other location inside document/data block that is defined by the line and character number. If there are more than one occurrences of the string, you can also define which one should be used by

setting the **Occurrence** property. You can also set the **offset** from the string where the field should be extracted. The offset can be a positive or a negative number. If you enter a negative number filter will extract also data in front of the string or the string itself.

- **Find string after field start**: End position of the field is defined by the position of a specified string after the field start position. When this string Is found in the provided data, the next character after the string determines the end position of the field. The string that filter is searching for, is not included in the extracted data. If there are more than one occurrences of the string, you can also define which one should be used by setting the **Occurrence** property. You can also set the **offset** from the string where the field should be extracted. The offset can be a positive or a negative number. If you enter a negative number filter will extract also data in front of the string or the string itself.
- **End of the line**: Value for the field is obtained from the start of the field to the end of the line. You can also define the offset from the end of the line (a negative number) . For example entering -3 in the **Characters** field will result in the field that will end 3 characters before the end of line.
- **End of document or data block**: End of the field is calculated by using the line and character offset. Both of these are negative numbers meaning that they define how many lines and characters from the end of document or data block backwards should the field end.

Formatting Field Values

Like with Structured Text Filter you can apply string manipulations on the field values. For more details see the Formatting Text section in the chapter about Structured Text Filter.

Binary Fields

By enabling **Field has binary data** property in the **Field Properties** you tell NiceLabel Automation to process binary data, like files created by printing to a file.

Note

NiceLabel Automation does not allow text formatting operations on binary fields.

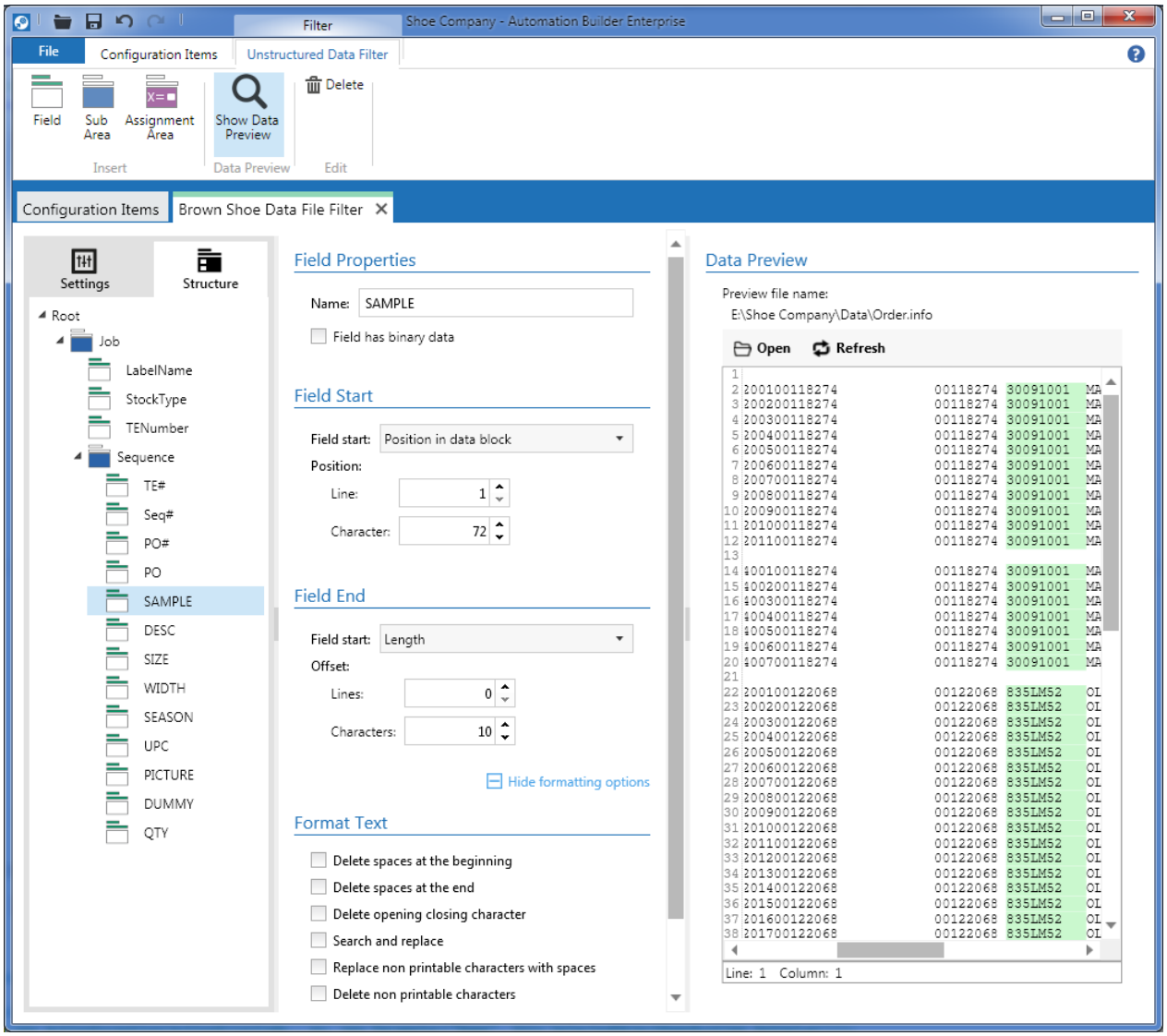


Figure 53 - Automation Builder - Unstructured Data Filter – Field

Sub Area

Sub area is a chunk of data inside a document. Sub area can be repeated throughout the document and it contains at least one data block. As mentioned earlier, data blocks contain document fields. Sub areas can contain other sub areas, which means you can define pretty complex filters to parse complex data structures.

For each sub area in a filter, **Use Data Filter** action gets a new **For each data block** sub action where you map fields to variables.

To describe a sub area in Automation Builder you need to give it a name, define how the data blocks inside the area start, where the first data block begins and when the last block ends.

Defining Data Blocks Inside Sub Area

Automation Builder offers several options to describe data blocks inside the sub area:

- Select **Each block contains fixed number of lines** when you know in advance how many lines will each block occupy.
- Use **Blocks start with a string** when each block starts with a certain string or a sequence of special characters. NiceLabel Automation will mark everything between the occurrences of the provided string as a data block.
- When each block ends with specific set of characters or bytes (special characters are allowed) choose **Blocks end with a string**.
- **Blocks are separated by a string** can be used when blocks do not start or end with a specific string, but are separated with a sequence of one or more characters.

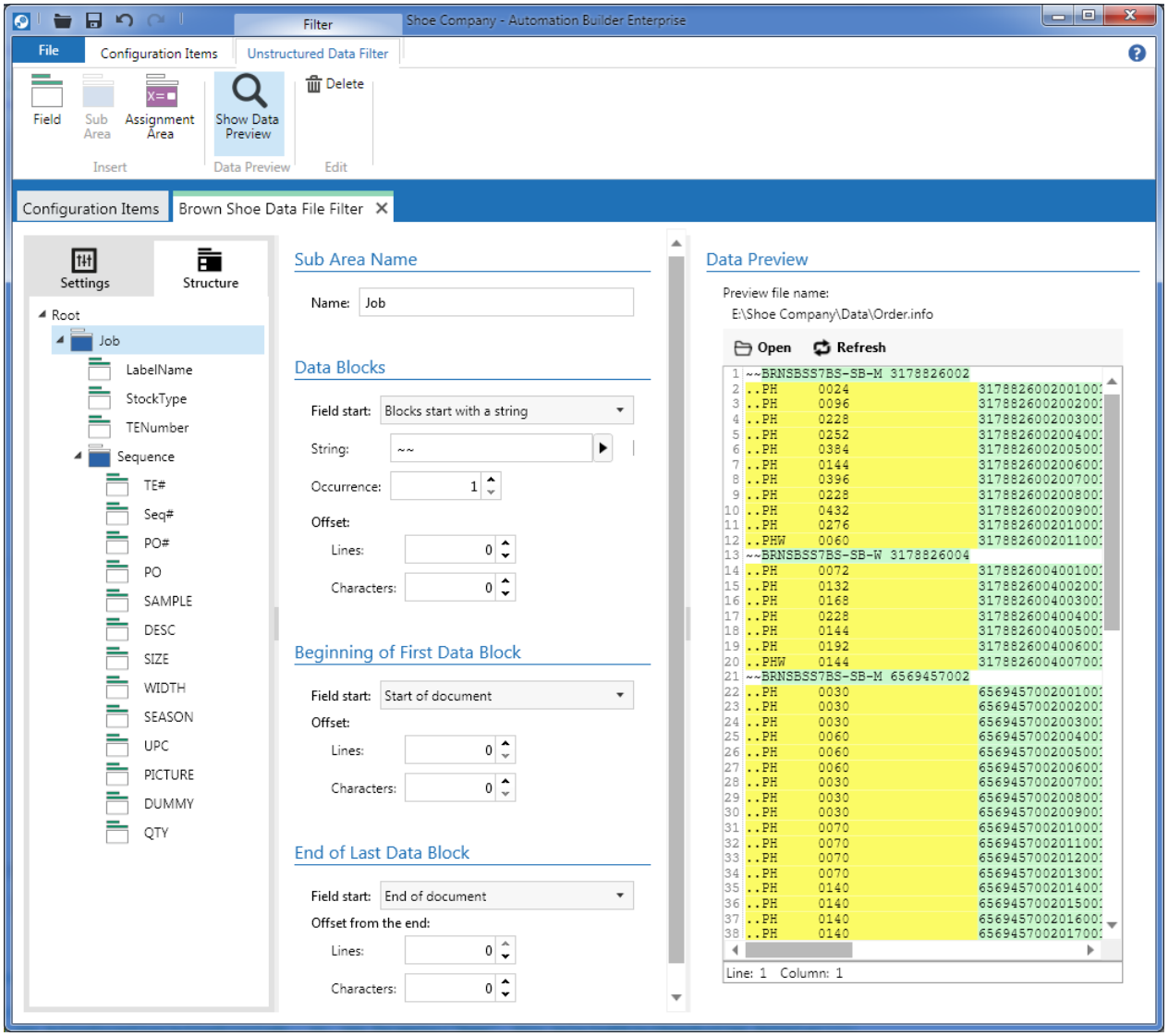


Figure 54 - Automation Builder - Unstructured Data Filter - Sub Area

Beginning of First Data Block

Sub area starts with the beginning of the first data block. Automation Builder allows you to select one of three options as the beginning of the first block. Each of the following options has the same meaning and settings as in document field start definition, so I will not describe them again here.

- **Start of document**
- **End of document**
- **Find string from start of document**

End Of Last Data Block

NiceLabel Automation also needs to know when the sub area ends. End of the sub area is defined by the end of the last data block. Again, the end of the last block has the same parameters as the field end position definition, so please check the section about document fields.

- **Position in document**
- **Length**
- **Find string from start of document**
- **Find string after field start**
- **End of the line**
- **End of the document**

Adding Sub Areas

To add a sub area, first select an existing sub area or the **Root** element in the **Structure** pane in Automation Builder and click the **Sub Area** button in the ribbon. Every level in the filter structure can have only one sub area, but you can have as many levels as you want.

Removing Sub Areas

To remove a sub area, select it and press DELETE key or click **Delete** button in the ribbon. Because Automation Builder supports undo, there is no confirmation window popping up, just the sub area and all of its fields and sub areas are deleted.

Adding Fields to Sub Area

Once the sub area is defined you can define the fields. To add a field to the sub area, select the sub area and click the **Field** button in the **Unstructured Data Filter** page in the ribbon. You can have as many fields as you need.

Assignment Area

Assignment areas in NiceLabel Automation give you the ability to automatically identify fields in unstructured data documents and map them to variables.

In order for this automatic mapping to work the data needs to adhere to certain rules. Fields and their values need to be in name/value pair form. For example, the following values would fit into such category:

```
DonationHR=G095605
Full Name: Lionel Messi
```

```
AccountNr...12983-323-323-23-32
```

Just like with sub area you also need to outline how data blocks are separated, where the first data block starts, and when the last block ends. For more details about this, please see the section about sub areas.

When you add the assignment area to the filter structure, NiceLabel Automation will automatically create two special fields that cannot be removed (unless the whole assignment area is removed):

- Value of **Variable name** field tells NiceLabel Automation which variable should be set.
- **Variable value** field contains the value for the variable defined in the **Variable name** field.

Both fields have the same properties as any other filter field and can also hold binary data.

If you add assignment area to the **Root** element, it will be "executed" only once per document, but if you add it to a sub area it will be processed for each data block in the selected sub area.

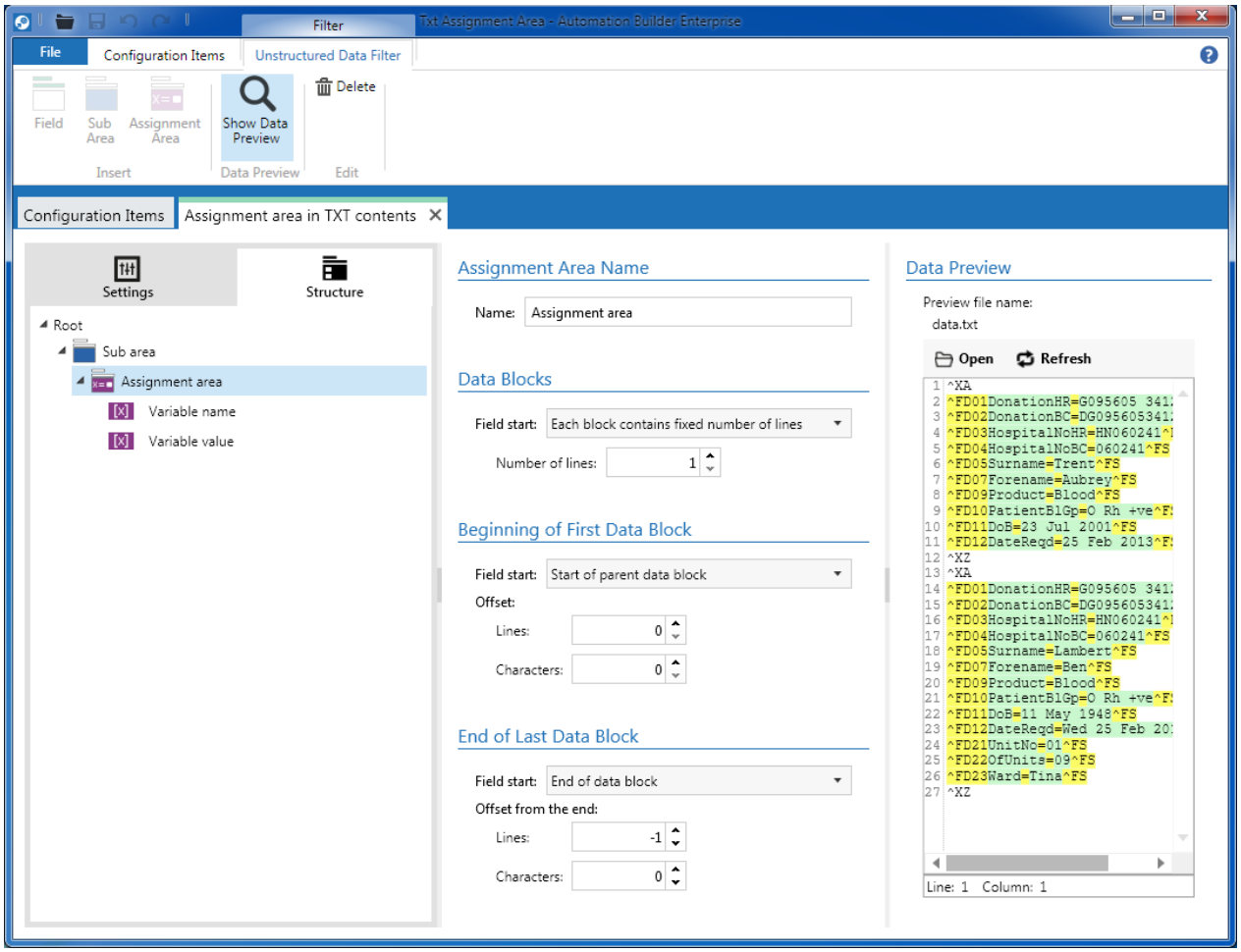


Figure 55 - Automation Builder - Unstructured Data Filter - Assignment Area

As I already mentioned, fields from a filter that uses an assignment area, and variable values are automatically mapped when **Use Data Filter** action is executed. A logical consequence of this fact is that

Data Mapping section of the **For each data block …** sub action does not display or allow you to change any of the mappings between fields and variables.

Note

If variable with the name that is defined in the Variable name **field does not exist there will be no errors raised, when trying to map fields with variables.**

XML Filter

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. Today, hundreds of document formats using XML syntax have been developed, including RSS, Atom, SOAP, and XHTML. XML-based formats have become the default for many office-productivity tools, including Microsoft Office, OpenOffice.org and LibreOffice, and Apple's iWork. XML has also been employed as the base language for communication protocols, such as XMPP.

NiceLabel Automation allows you to extract data from such documents by using **XML Data Filter** and then print it on a label or use it in any other way.

Definition

XML Tag is a markup construct that begins with < and ends with >. There are three types of tags:

- **Start tags: <node>**
- **End tags: </node>**
- **Empty element tags: <node />**

XML Element is a logical document component which either begins with a start tag and ends with a matching end tag or consists only of an empty element tag. The characters between the start and end tags, if any, are the element's content, and may contain markup, including other elements, which are called child elements. An example of an element is <Greeting>Hello, world.</Greeting>. Another is <line-break />.

XML Attribute is a markup construct consisting of a name/value pair that exists within a start tag or empty element tag. In the example (below) the element img has two attributes, src and alt

****

Source: http://en.wikipedia.org/wiki/XML (Wikipedia)

You can define the structure of the XML Data Filter in two ways. You can either import an existing XML file and let NiceLabel Automation determine the structure of the XML document automatically by clicking the **Import Data Structure** button in the ribbon. Or you can define the structure manually by using the **Add Element** and **Add Attribute** buttons.

Every element can be used as a **variable value** or as a **data block**.

Automation Builder will display the name of the element in the tree view in bold when you define it as **variable value**. Also if the **Data Preview** is enabled, the value of selected element will be highlighted in the preview.

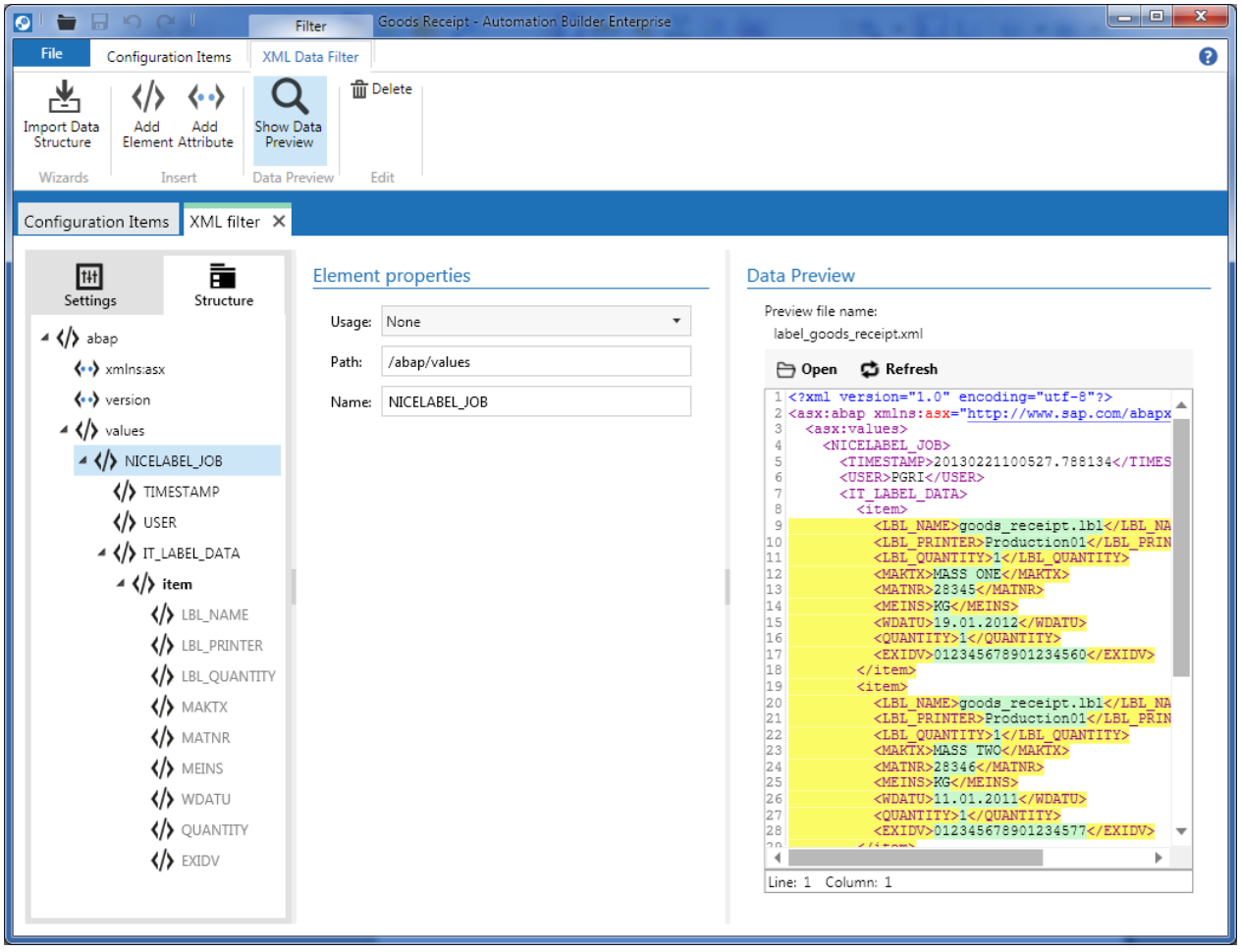


Figure 56 - Automation Builder - XML Data Filter

Data block is an XML element that is repeated multiple times throughout the XML document. Every data block can be defined as repeatable element or as an assignment area, or both.

If data block is marked as **repeatable element**, NiceLabel Automation will process all occurrences of the XML element and not just the first one.

Assignment area is a data block that contains both variable name and variable value (name/value pair). Variable name can be defined by the XML element name, element value or a value of an XML attribute. Variable value can come from the element value or from the attribute value.

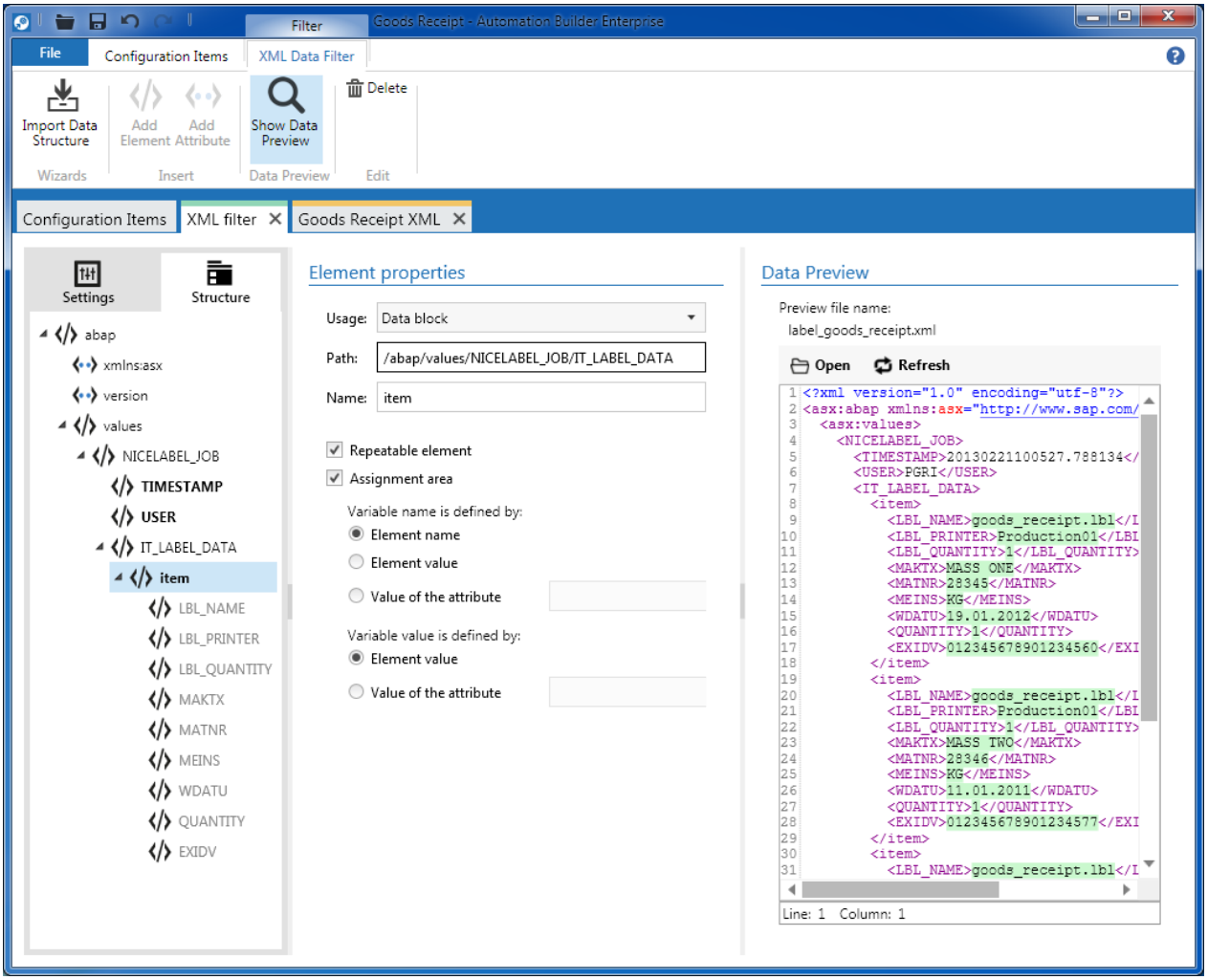


Figure 57 - Automation Builder - XML Data Filter - Data Block

Here are two examples of XML assignment areas.

Assignment Area That Gets Variable Name and Variable Value From XML Attributes

For the following XML document you would define *item* as data block that is both repeatable element and assignment area. Assignment area would get the name of variable from attribute *name*, and the value from attribute *value* (both attributes are part of the *item* element).

```
<purchaseOrder>
    <job>
        <label>
            <item name="Product" value="NiceLabel Automation">
                <quantity>100</quantity>
                <printer>TOSHIBA B-SA4TP TS</printer>
            </item>
```

```
        </label>
        <label>
            <item name="Product" value="NiceLabel Designer Standard">
                <quantity>105</quantity>
                <printer>SATO TG312</printer>
            </item>
        </label>
    </job>
</purchaseOrder>
```

Assignment Area Where Variable Name is Defined by Element Name and Variable Value is Defined by Element Value

XML element *item* is again a repeatable element and assignment area where variable name is defined by element name, and variable value comes from the element value. For the first item in the document, variable 'product' would get 'NiceLabel Automation' as its value, variable 'quantity' would be set to '100', and 'printer' would be set to 'TOSHIBA B-SA4TP TS'.

```
<purchaseOrder>
    <job>
        <item>
            <product>NiceLabel Automation</product>
            <quantity>100</quantity>
            <printer>TOSHIBA B-SA4TP TS</printer>
        </item>
        <item>
            <product>NiceLabel Designer Standard</product>
            <quantity>105</quantity>
            <printer>SATO TG312</printer>
        </item>
    </job>
</purchaseOrder>
```

Data Preview

Automation Builder also offers live preview for the filter. The preview highlights parts of the document when you select elements of the document in the structure tree view. This way you can see if the filter is correctly configured.

Using Data from Filter in Actions

To transfer data from the XML file to the actions (e.g. **Print Label**), you need to have **Use Data Filter** action and map field names from the filter to variables (see **Data Mapping** section in **Use Data Filter** action properties).

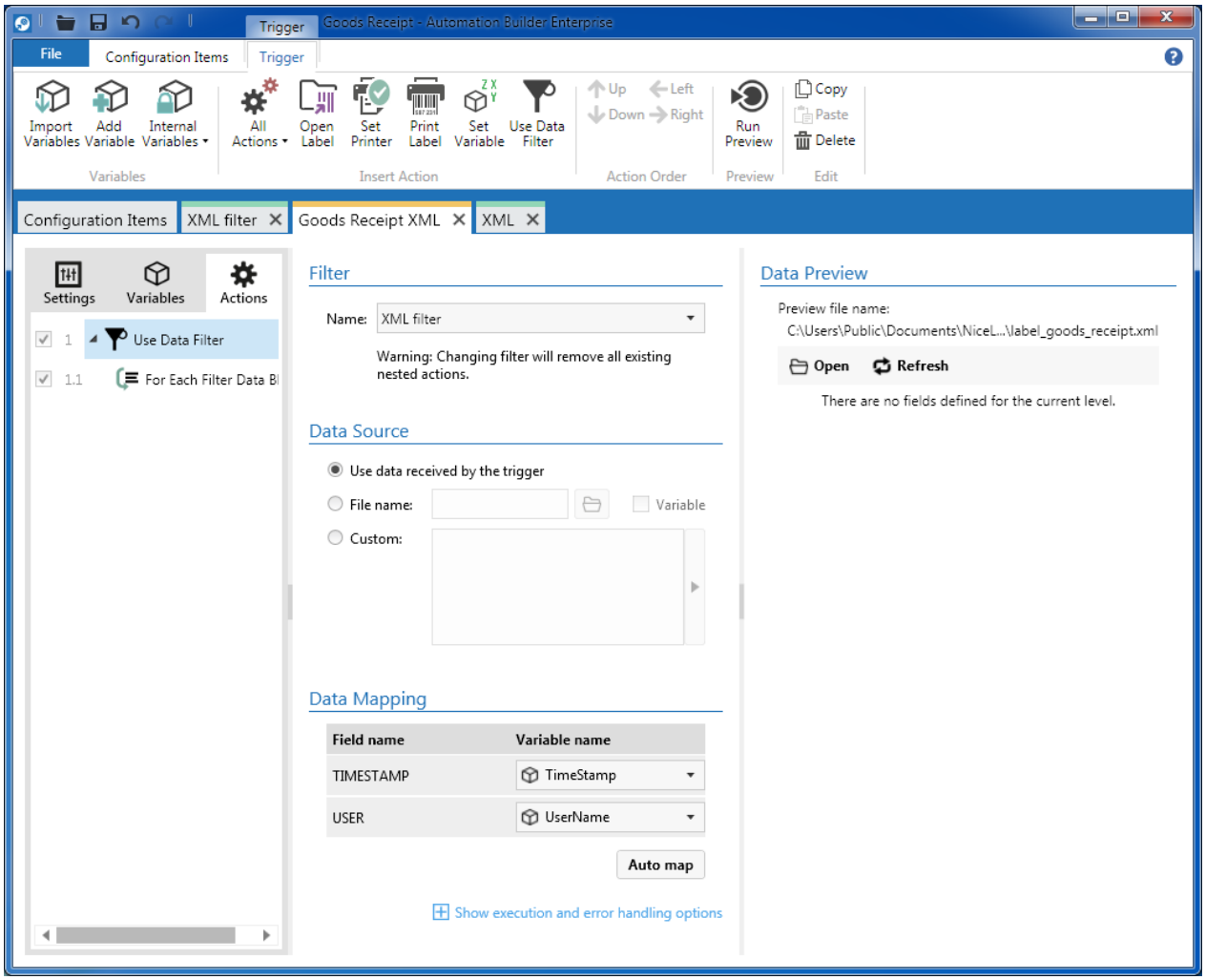


Figure 58 - Automation Builder - XML Data Filter in Use Data Filter Action - Data Mapping

Actions

Actions

There are dozens of actions available in NiceLabel Automation that are organized in several categories:

- **General**: All actions in this category have something to do with printing labels.
- **Printer** actions allow you to interact with printers, print jobs and printer statuses.
- Actions in **Variable** category set variable values, save and load their values from files and perform string manipulations.
- **Batch Operation** actions enable executing actions in batches or loops.
- **Connectivity** actions help you achieve interconnectivity with TCP/IP ports, HTTP servers, serial ports, etc.
- As the name suggests **Other** category contains actions that would not fit in any other category.

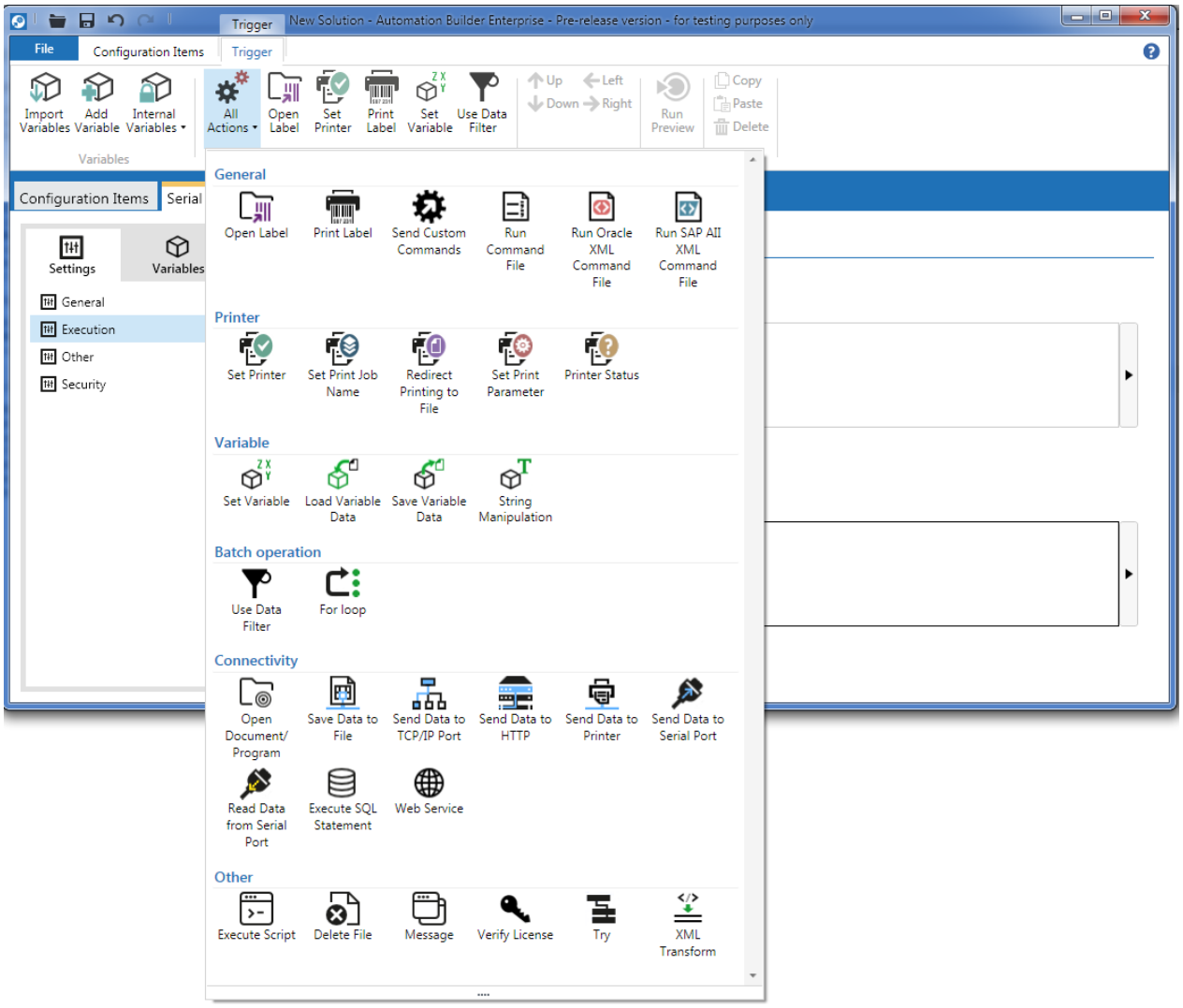


Figure 59 - Automation Builder – Available Actions

Not all actions are available in all NiceLabel Automation editions.

Action	Automation Easy	Automation Pro	Automation Enterprise
Open Label	X	X	X
Set Printer	X	X	X
Print Label	X	X	X
Use Data Filter	X	X	X
Set Print Job Name	X	X	X
Redirect Printing to File	X	X	X
Set Variable	X	X	X
String Manipulation	X	X	X
Send Data to TCP/IP Port	X	X	X
Send Data to Printer	X	X	X
Save Data to File			X
Delete File			X
Read Data from Serial Port	X	X	X
Send Data to Serial Port	X	X	X
For Loop			X
Open Document / Program	X	X	X
Execute Script	X	X	X
Send Custom Commands		X	X
Run Command File		X	X
Run Oracle XML Command File		X	X
Run SAP AII XML Command File		X	X
Set Print Parameter		X	X
Printer Status		X	X
Load Variable Data		X	X
Save Variable Data		X	X
XML Transform		X	X
Web Service			X
Send Data to HTTP			X
Execute SQL Statement			X
Verify License			X
Message Box			X
Try			X

Table 4 - Actions in different editions of NiceLabel Automation

Adding Actions

To add an action to the list you have to select it from the list in the **Trigger** ribbon. Actions that are used by most users are available with one click while for others you have to click the **All Actions** button to open the drop down gallery of actions separated in different categories.

If there are no actions in the list yet, the action will be added at the top of the list. If there are already some actions in the list, the new action will be added right after the one that is currently selected.

Some actions, like **Print Label** or **Set Printer**, cannot be used without a specific parent action. For these two actions, as well for **Send Custom Commands**, **Run Command File**, **Run Oracle XML Command File**, **Run SAP AII XML Command File**, **Set Print Job Name** and **Set Print Parameter** actions, **Open Label** is their parent action and they need to be nested inside of it. If you try to add one of previously mentioned actions, and **Open Label** action does not exist or is not selected, one will be added automatically and the requested action will be positioned inside of it.

Use Data Filter and **For Loop** actions also can have other actions nested below them.

Removing Actions

If you want to delete an action, just press the DELETE key or right click it to open popup menu, and select **Delete** option. If currently selected action has nested actions, they will also be removed.

The application does not ask for confirmation before it deletes the actions, because you can always undo the deletion.

Editing and Copying Actions

Once you select an action you can immediately start editing its properties, as they are displayed on the right side of the window.

If you want to copy an action, select it, press CTRL + C to copy it to clipboard, move focus to the position you would like to put the copy of the action, and press CTRL + V to paste it. Copy and paste commands are also available in the ribbon and in the popup menu.

Navigating and Ordering Actions

To navigate between actions you can use the UP and DOWN keys or respective command buttons in the ribbon. LEFT and RIGHT keys can be used to close or open a list of nested actions.

To change the order of actions and their nesting level you can either use the CTRL + UP, CTRL + DOWN, CTRL + LEFT, and CTRL + RIGHT keys, or command buttons in the ribbon and the contextual popup menu.

Common Action Properties

All actions have some common properties:

- **ID number**: Every action has an ID number automatically associated with it and it cannot be changed. This ID comes in very handy if you have lots of actions and you want to find the problematic one. You can see the ID in the actions list.
- **Enabled**: If you need to prevent the action from executing, you can disable it. The fastest way to toggle this flag is to check or uncheck it in the actions list.
- **Condition**: This property allows you to execute an action only if certain condition is met. Condition is a one line Python or VBScript script (depending on the selected scripting language in the trigger definition) that needs to return a True or False. An example of such condition can be

found in the screenshot below. Selected action will be executed only if the LabelFile variable will have “BigLabelOnly” as its value.

- **Ignore error**: If you want to continue with the execution of the action, even if an error occurs, check this property. Error will be saved in the log, and the actions following the erratic action will still be executed.
- Checking **Save error to variable**, and selecting a variable will cause NiceLabel Automation to save errors that might occur during the execution of selected action to the specified variable. Internal variables **ActionLastErrorId** and **ActionLastErrorDesc** will have error details anyway. Please note that, for **Print Label** action, all errors will be caught only if Supervised printing property of the selected trigger is enabled.

All these common properties (with the exception of the action ID) can be found if you click the **Show execution and error handling options** link.

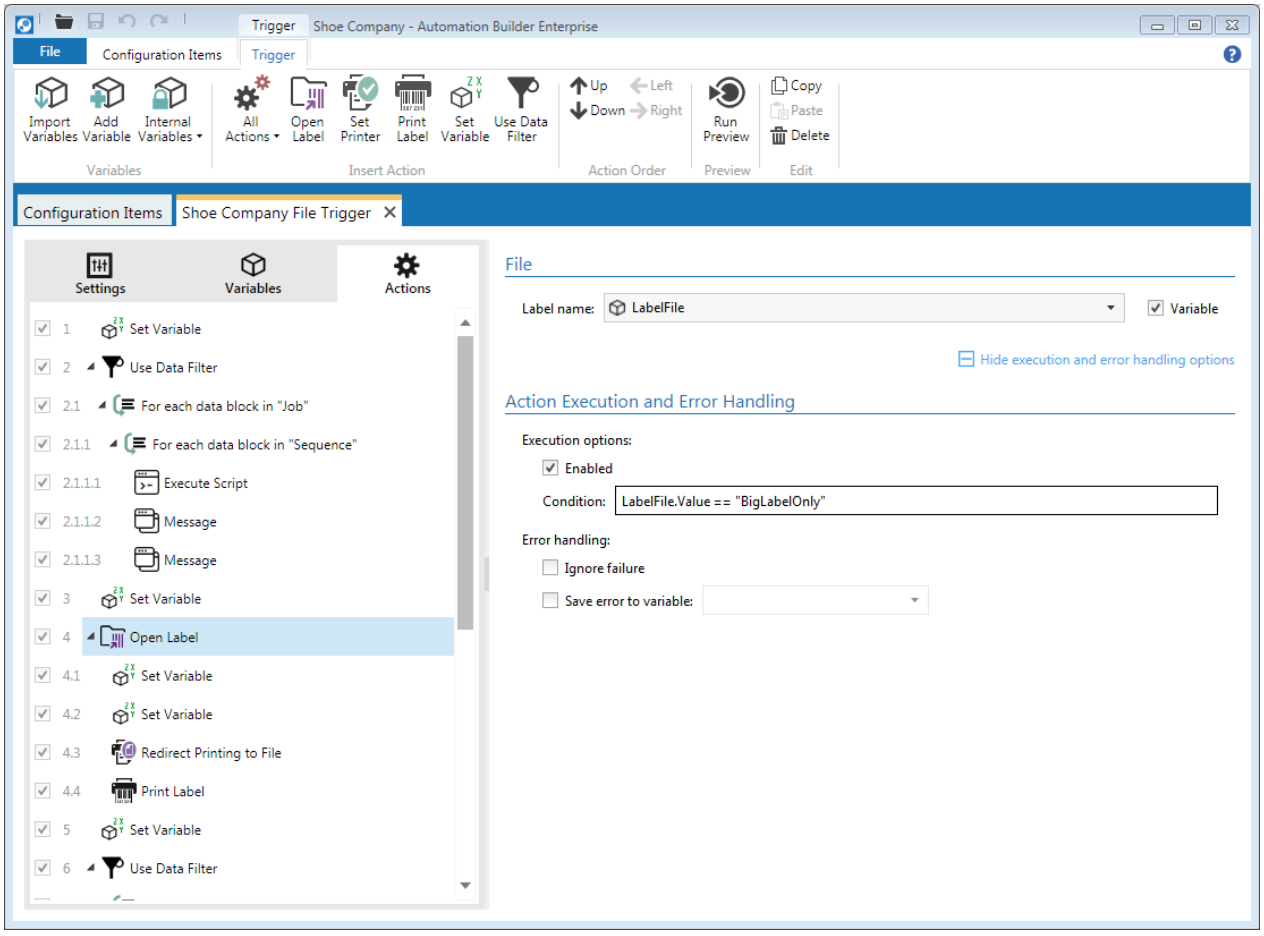


Figure 60 - Automation Builder - Common Action Properties

Open Label Action

Open Label action is the parent action for **Print Label**, **Set Printer**, **Send Custom Commands**, **Run Command File**, **Run Oracle XML Command File**, **Run SAP AII XML Command File**, **Set Print Job Name** and **Set Print Parameter** actions. It opens the label that previously mentioned actions then utilize.

Action has (besides common properties that all actions have) only one setting – **label name**. Label name can be either hardcoded or it comes from a variable.

Because NiceLabel Automation runs as a service it might not have the same permissions as the user that designed the configuration, so make sure that the permissions are configured correctly. Also if you intend to use a file on a network share it would be best, if you used an UNC (Uniform Naming Convention - (Wikipedia)) notation for the file path and not the mapped drive letter, as it might not be accessible to the NiceLabel Automation service. Use \\ComputerName\Shared Folder\Resource instead of "M:\Shared Folder\Resource", where M is mapped to \\ComputerName.

Set Printer Action

Set Printer action sets the printer for selected label used in the parent **Open Label** action. This action overrides the printer that is saved in the label, so when the **Print Label** action is executed, it prints to the printer set in this action.

Printer name can be selected from the list of printers installed on a computer where Automation Builder is running, entered manually (it does not have to exist on a computer where you are using Automation Builder), or connected to a variable.

Tip

If you are setting printer to a completely different type of printer (like from thermal printer to laser printer), or to a different type of thermal printer (e.g. from CAB printer to Toshiba printer), you should be careful and test label designs thoroughly, because printer driver settings, internal fonts or counters and other printer specific elements, might not be compatible between the two printers.

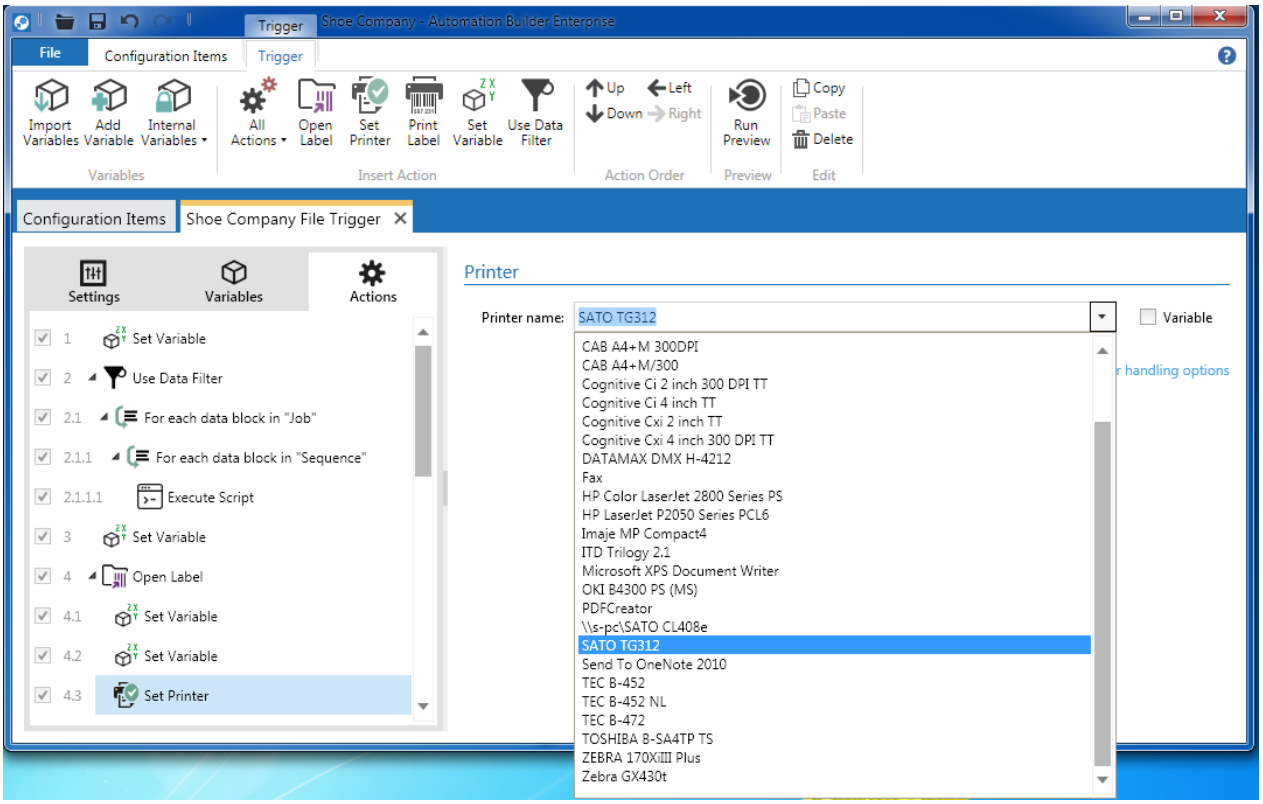


Figure 61 - Automation Builder - Set Printer Action

Print Label Action

To print labels use **Print Label** action. Action will print to the printer saved in the label, to the one set by the **Set Printer** action, or to the default printer.

There are three options to set a number of labels:

- **Fixed**: A constant number of labels is printed, or the number of labels can come from a variable value.
- **Unlimited**: This option is generally used with label files that use database access functions with **Database Record Retrieving** property set to **All records**.
- **Variable quantity**: When the label quantity is not known in advanced, but is determined at print time from a field in the database or when one of the variables on the label has **Use variable as label quantity** property selected, you can select this option.

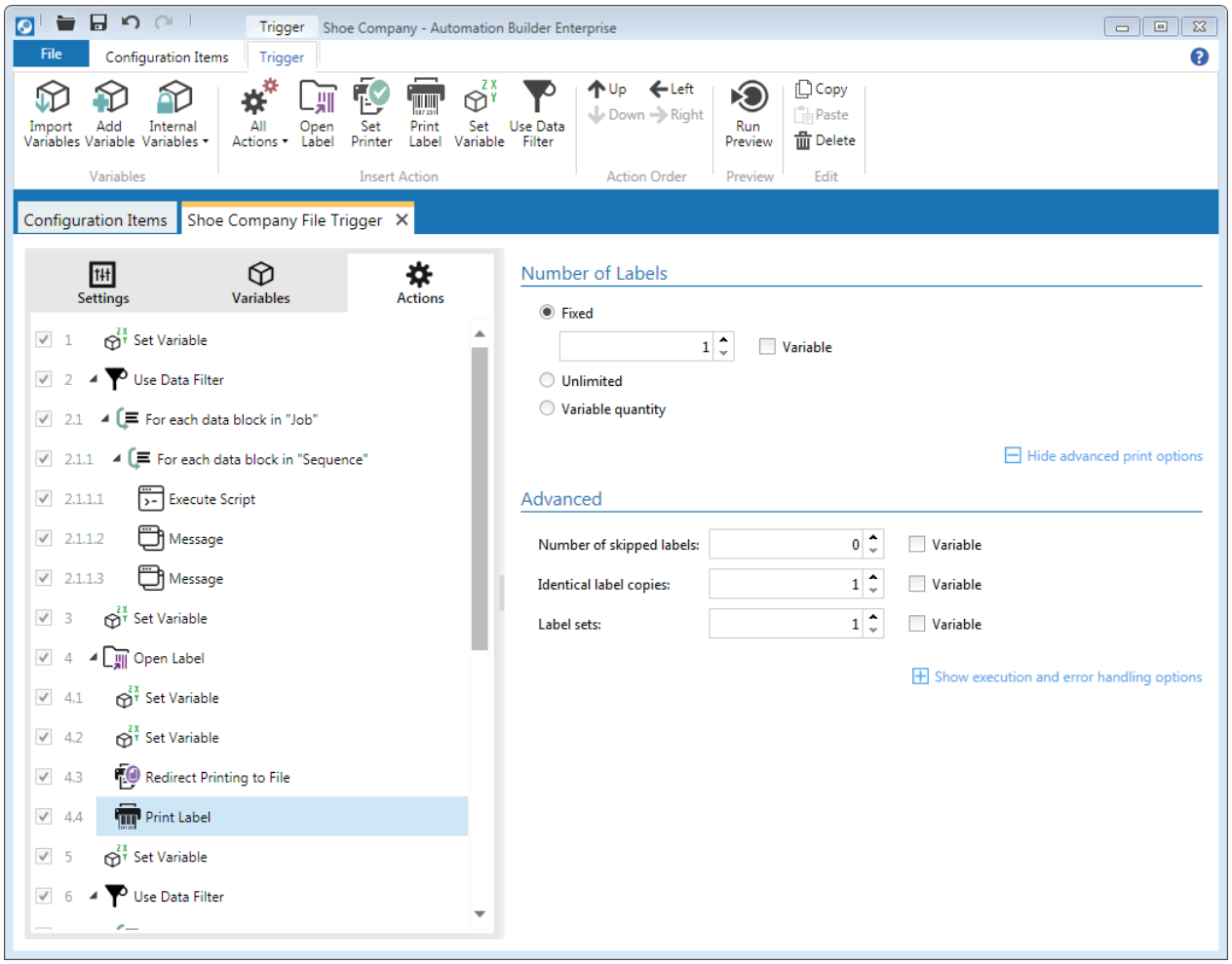


Figure 62 - Automation Builder - Print Label Action

Action also offers several advanced options:

- **Number of skipped labels** defines how many labels will be skipped on the first page. This feature is very useful if you use A4 or similar sheets of labels, and have already used some of them, but do not want to throw rest of them away.
- **Identical label copies** sets how many copies of every label will be printed.
- **Label sets** tell the printing process how many times it should be repeated.

All of these advanced settings can have predefined values or get them from variables at print time.

Use Data Filter Action

To get the data from the filter to actions, you need **Use Data Filter** action. **Use Data Filter** action maps fields from selected filter and variables.

When you add the action, you first have to select the filter you want to use from a drop down list. Additionally you can also create new filter directly from the drop down list.

Note

Be careful when changing selected filter in the list. If you already have some actions defined as sub actions to the Use Data Filter action, and you select a different filter, all existing nested actions will be removed. If you want to change the filter, first move sub actions outside of the Use Data Filter action.

Depending on the structure of the filter you use in action, one or more sub actions will be automatically added when you select the filter. This can be either **For each line** (if Structured Text filter is used) or **For each data block in …** (for XML and Unstructured Data filters) sub actions.

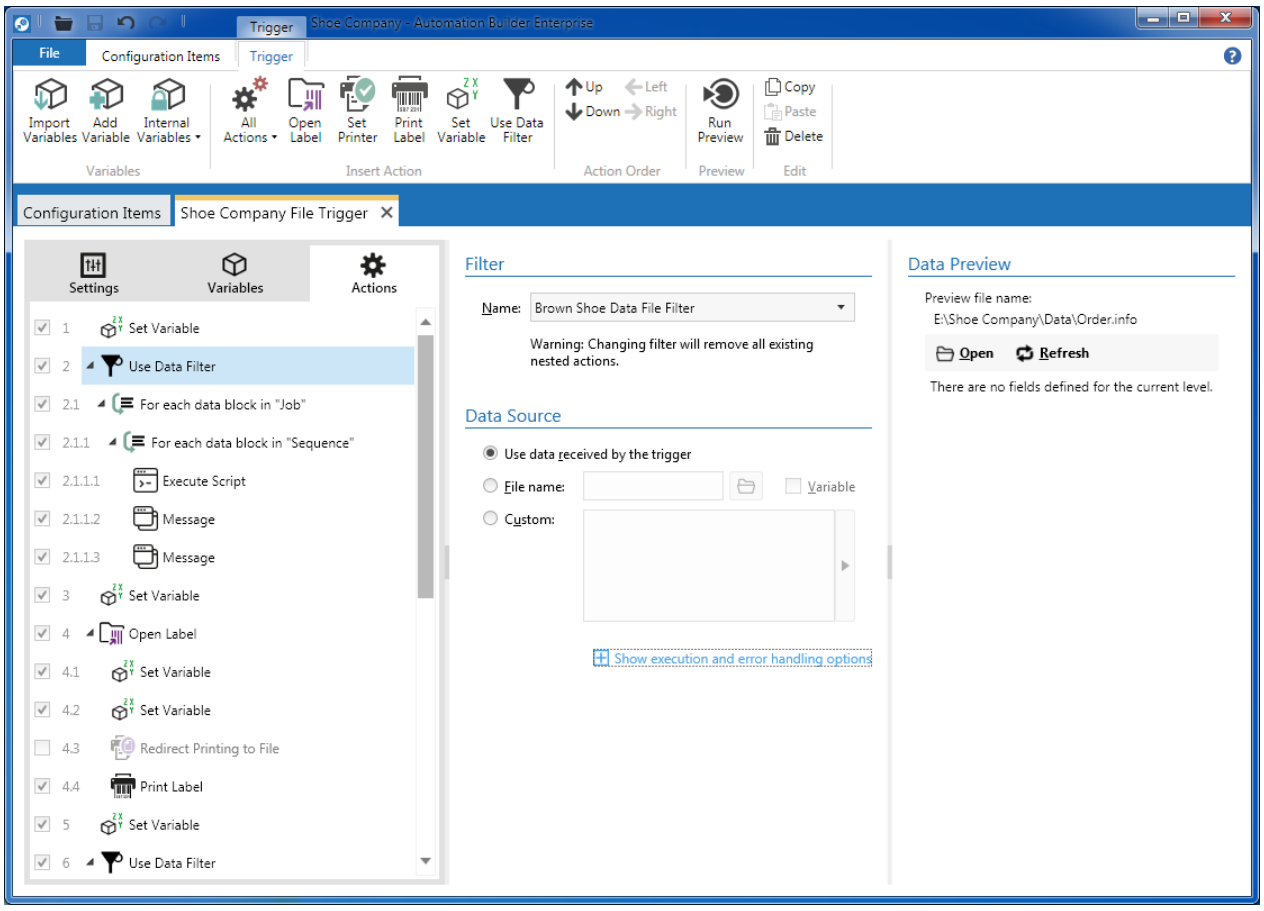


Figure 63 - Automation Builder - Use Data Filter Action

In the **For each data block in …** and **For each line** sub actions you define how field names are mapped to variables. If you gave variables the same names as fields in the filter, you can automatically map them by

clicking the **Auto map** button. If not, you can still map them manually by selecting variable name from a list next to the field name.

In the **Data Preview** pane you can select the data file and see how filter will parse specific values.

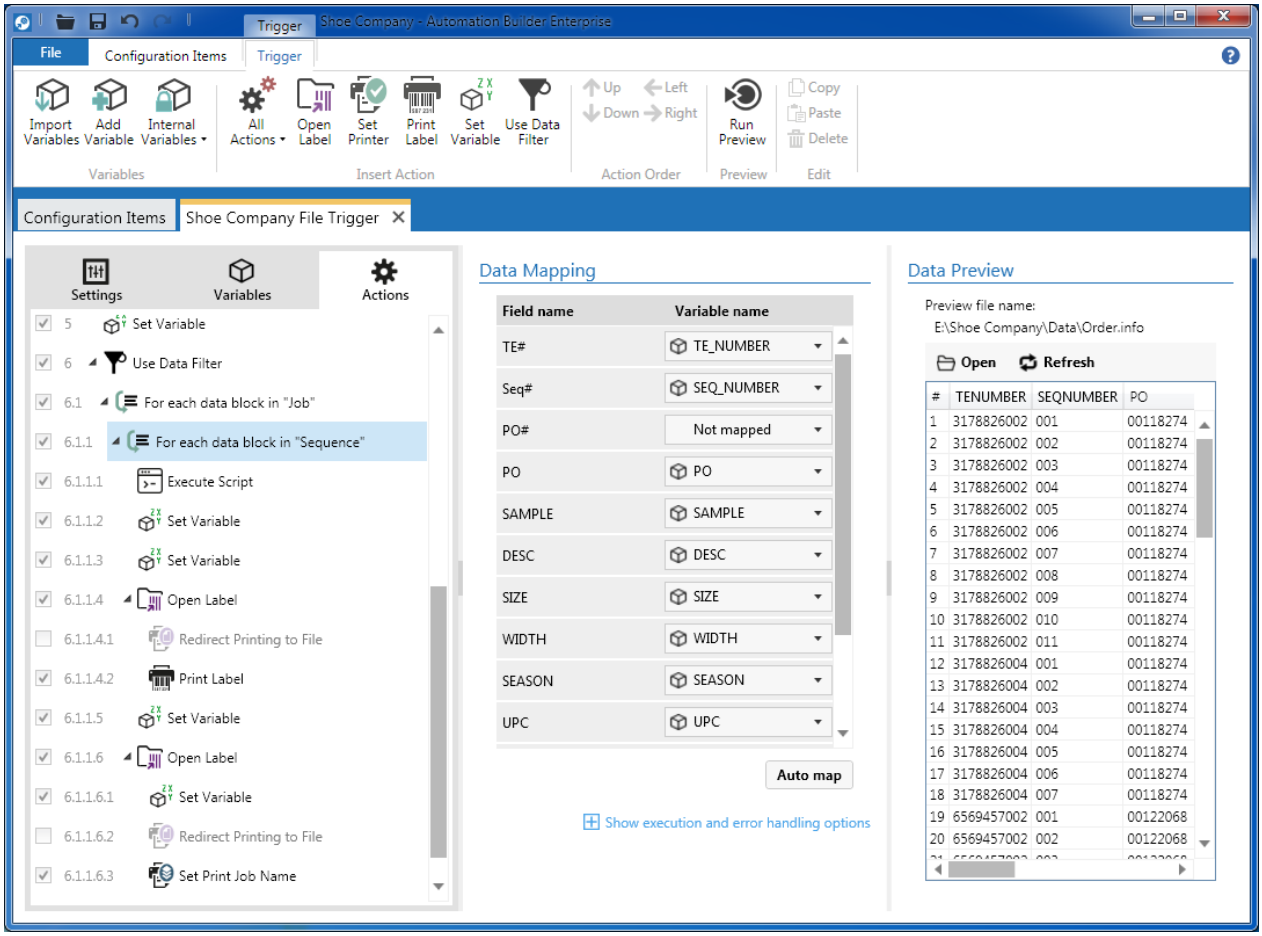


Figure 64 - Automation Builder - Use Data Filter Action - For each data block

Defining Data Source

To define what you would like to use as the data source for the filter used in the action, go to the **Data Source** section of the **Use Data Filter** action. There are three possibilities.

- Selecting **Use data received by the trigger** means that the filter will process data received by the trigger when it is fired.
- If you want to use data from specific file (which can be either fixed or variable), use **File name** option.
- Choosing **Custom** allows you to use fixed values, variable values or a combination of both. Special characters are also supported.

Send Custom Commands Action

Send Custom Commands action allows you to execute NiceCommands like PRINTER, SET, PRINT, etc. (for more information about NiceCommands see Appendix C). Almost all of the commands are already available as actions, so in most cases you will not have to use this action.

Send Custom Commands action needs to have a parent **Open Label** action, otherwise NiceLabel Automation cannot know which label you want to use. This means that you cannot use LABEL command because the label file is already defined by the **Open Label** action. This behavior is the same as in NiceWatch or in NiceForm.

Although you might not figure it out from the Automation Builder user interface, you can also use variable values in the custom commands. You can enter them manually by putting square brackets around variable name (e.g. [Quantity]), or you can use the script editor to help you do that.

In Automation Builder you can also load a list of custom commands from a file or save it to a file.

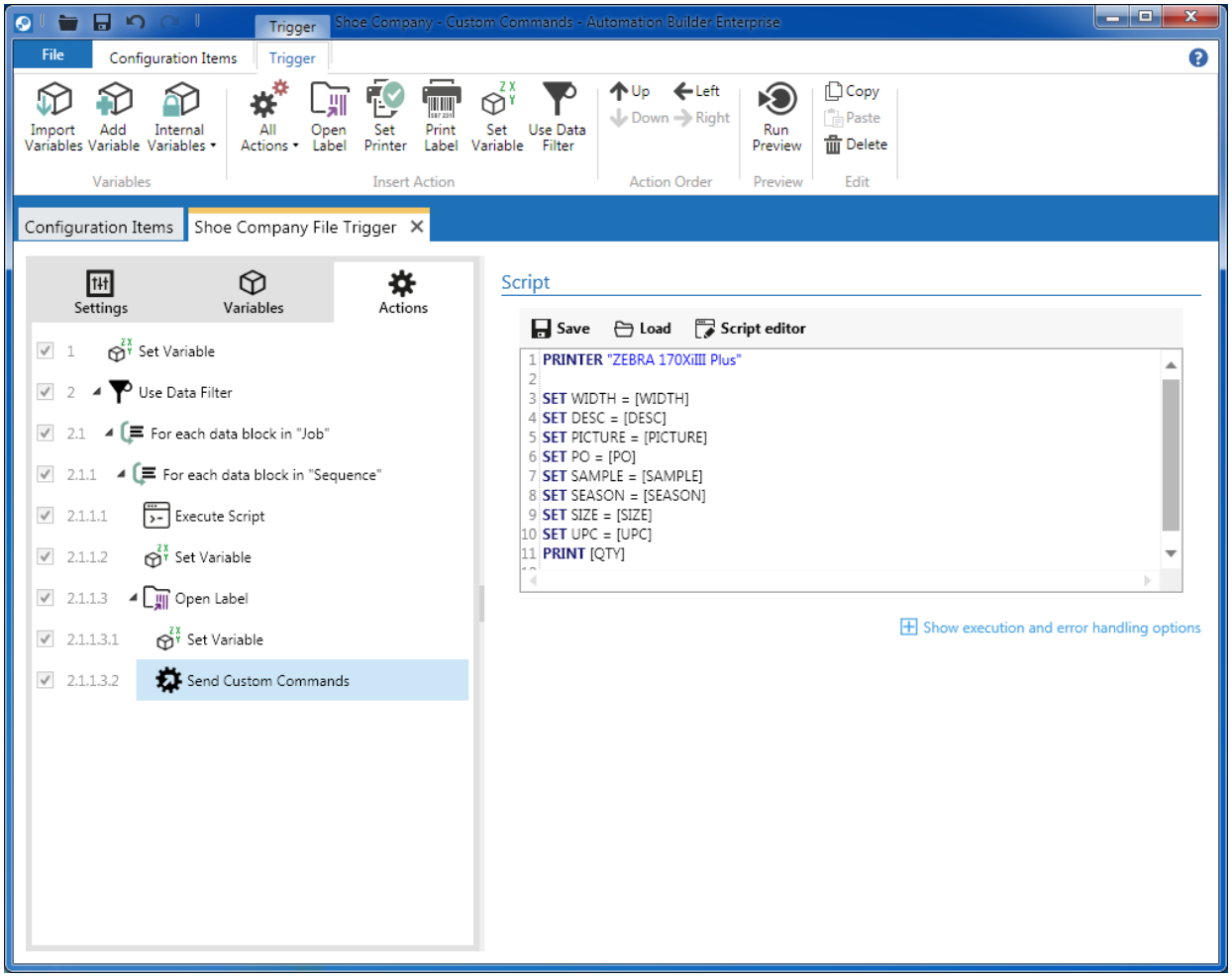


Figure 65 - Automation Builder - Send Custom Commands Action

Tip

If you need to manually create a list of commands or a command file, you can use Label Structure utility that is available for download from www.nicelabel.com. Label Structure can generate .JOB, .XML and .CSV command files from the selected label file.

All you have to do is to open label file, go to the Tools menu and select Create command file.

Run Command File Action

NiceLabel Automation supports five types of command files (for more information about command files, see appendices):

- JOB File
- XML File
- CSV File
- Oracle XML File
- SAP AII XML File

Run Command File action can handle first three types and will execute commands from a file that you defined in the **File name** property. Commands are executed in the same order as they are listed in the file.

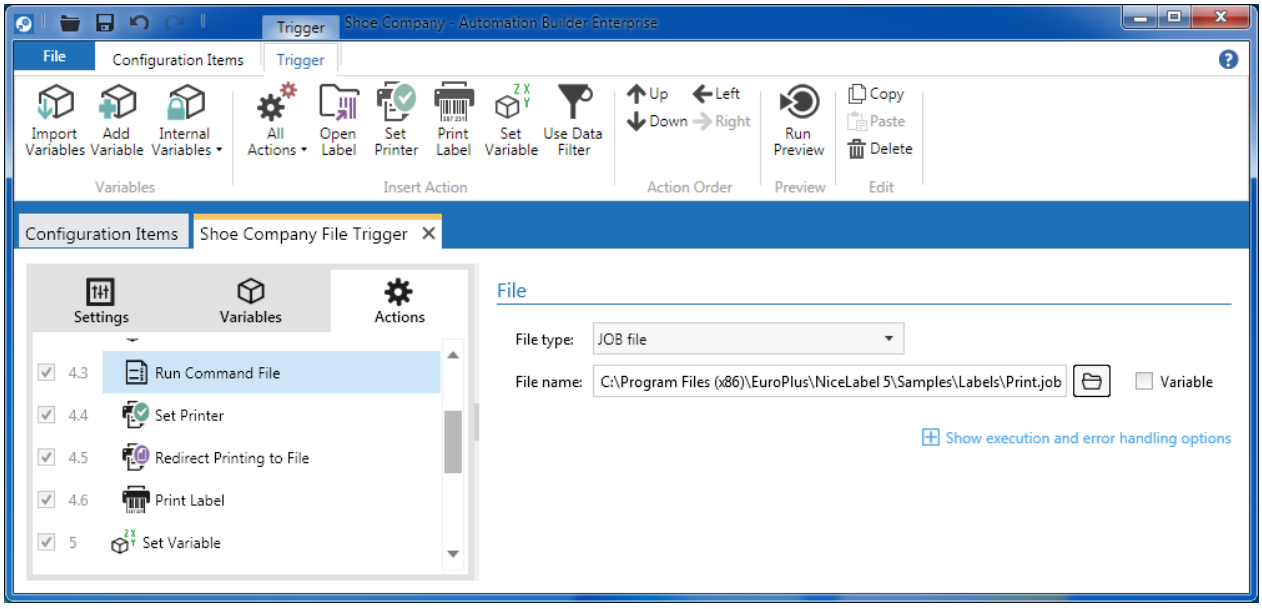


Figure 66 - Automation Builder - Run Command File Action

Run Oracle XML Command File Action

Run Oracle XML Command File action allows NiceLabel Automation to parse XML command files provided by Oracle Warehouse Management (WMS) and Oracle Mobile Supply Chain Architecture (MSCA) products, and execute commands from those files.

The following XML DTD (Document Type Definition - (Wikipedia)) defines elements that can be used in the XML file:

```
<!ELEMENT labels (label)*>
<!ATTLIST labels _FORMAT CDATA #IMPLIED>
<!ATTLIST labels _JOBNAME CDATA #IMPLIED>
<!ATTLIST labels _QUANTITY CDATA #IMPLIED>
<!ATTLIST labels _DUPLICATES CDATA #IMPLIED>
<!ATTLIST labels _PAGES CDATA #IMPLIED>
<!ATTLIST labels _PRINTERNUMBER CDATA #IMPLIED>
<!ATTLIST labels _PRINTERNAME CDATA #IMPLIED>
<!ELEMENT label (variable)*>
<!ATTLIST label _FORMAT CDATA #IMPLIED>
<!ATTLIST label _JOBNAME CDATA #IMPLIED>
<!ATTLIST label _QUANTITY CDATA #IMPLIED>
<!ATTLIST label _DUPLICATES CDATA #IMPLIED>
<!ATTLIST label _PAGES CDATA #IMPLIED>
<!ATTLIST label _PRINTERNUMBER CDATA #IMPLIED>
<!ATTLIST label _PRINTERNAME CDATA #IMPLIED>
<!ELEMENT variable (#PCDATA)>
<!ATTLIST variable name CDATA #IMPLIED>
```

From the previous DTD, the following sample XML command file is generated:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE labels SYSTEM "label.dtd">
<labels>
<label _FORMAT="ProductA.lbl" _QUANTITY="2" _PRINTERNAME="CAB A4+" _JOBNAME="Items">
<variable name="ITEM">AS54888</variable>
<variable name="ITEMDESC">Desktop</variable>
<variable name="LOT">10292</variable>
<variable name="QTY">2</variable>
<variable name="UOM">Ea</variable>
</label>
<label _FORMAT ="ProductB.lbl" _QUANTITY="100">
<variable name= "ITEM">AS12345</variable>
<variable name= "ITEMDESC">Mouse</variable>
<variable name= "LOT">10200</variable>
<variable name= "QTY">100</variable>
<variable name= "UOM">Ea</variable>
</label>
</labels>
```

As a result of the XML file above, NiceLabel Automation would print 2 ProductA labels to CAB A4+ printer, and 100 ProductB labels to the printer defined in the label design. In order for NiceLabel Automation to

print correct values on a label, ITEM, ITEMDESC, LOT, QTY and UOM variables need to be defined in NiceLabel Automation and on the label.

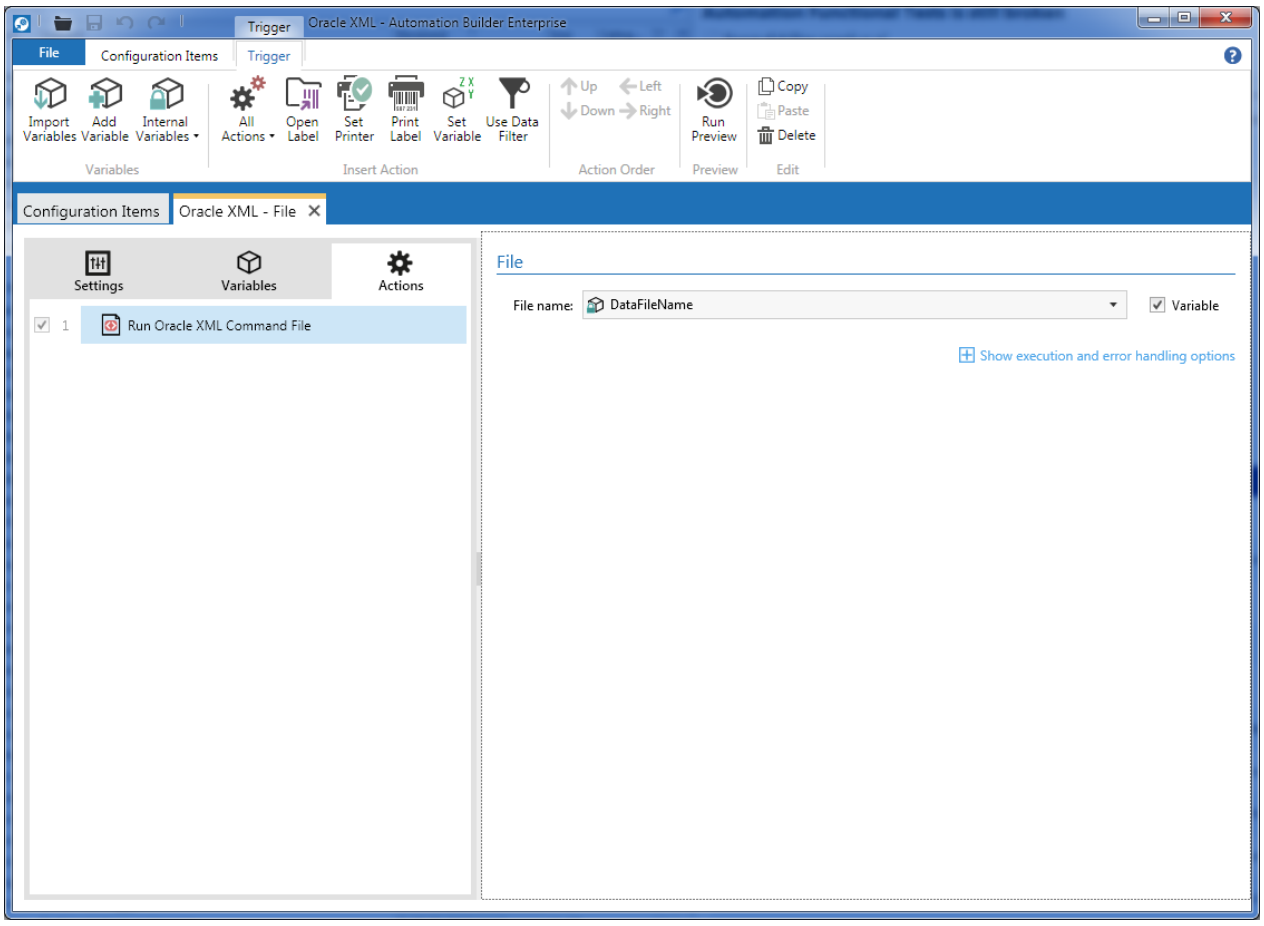


Figure 67 - Automation Builder - Run Oracle XML Command File Action

Run SAP AII XML Command File Action

Run SAP AII XML Command File action transforms commands that are listed in the file provided by SAP ERP software into actions and executes them.

SAP AII XML file might look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Command xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="Command.xsd">
  <WriteTagData readerID="Writer_Device">
    <item>
      <FieldList format="E:\ProductA.lbl" jobName="ProductA" quantity="2">
        <Field name="EPC">30740242220403B8000000008</Field>
        <Field name="EPC_TYPE">SGTIN-96</Field>
        <Field name="EPC_URN">urn:autoid:tag:sgtin-96:3.5.0037000.065774.8</Field>
        <Field name="PRODUCT">Schrauben</Field>
        <Field name="PRODUCT_DESCRIPTION">Panhead-AW 20 Vollgewinde</Field>
      </FieldList>
```

```
    </item>
  </WriteTagData>
</Command>
```

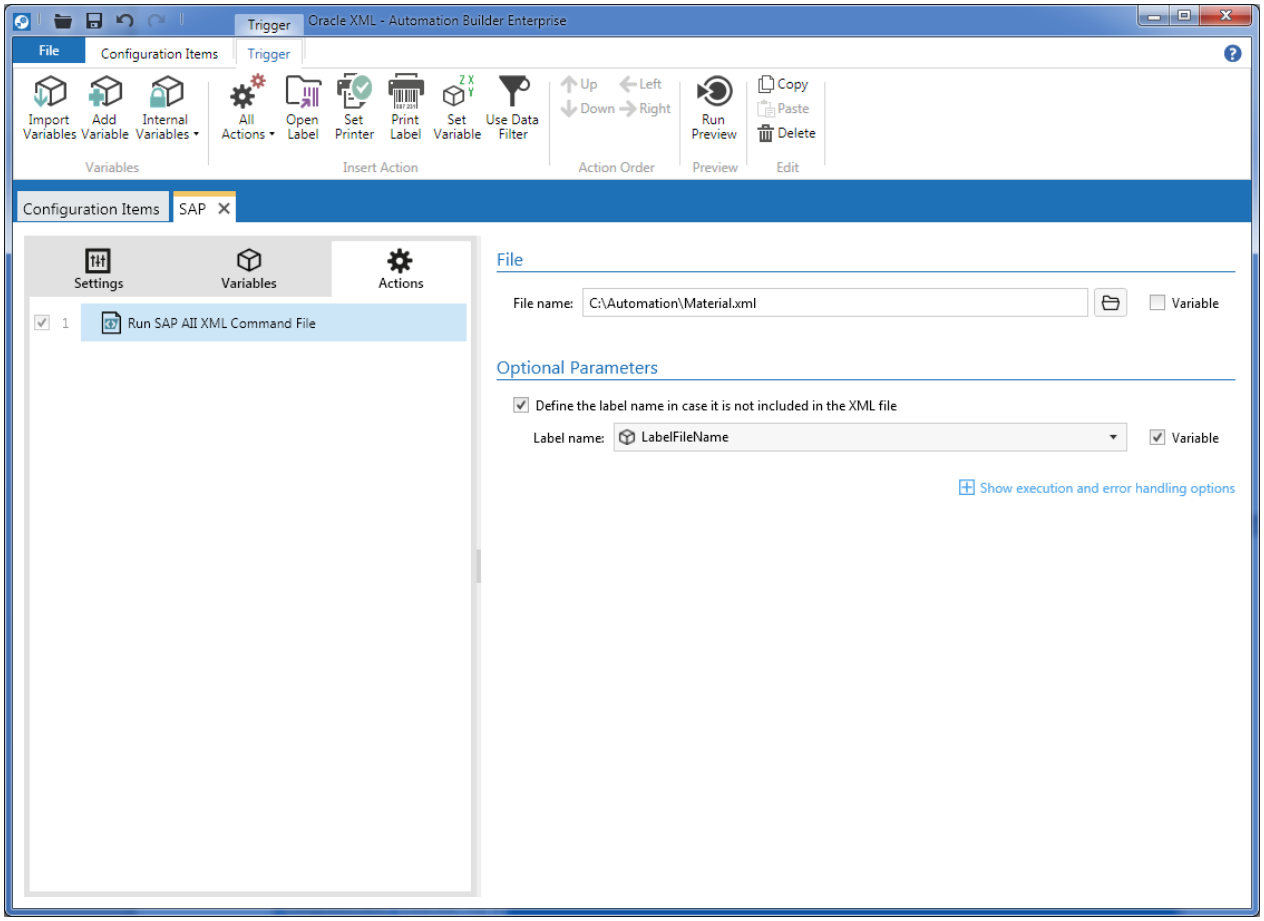


Figure 68 - Automation Builder - Run SAP AII XML Command File Action

In case the SAP AII XML file does not contain information about the label name, you can define it in the action properties. Just like with **File name**, you can use predefined file name, or connect **Label name** with a variable that will provide the value when action is executed.

Set Print Job Name Action

To set the document name that will appear in the Windows Spooler, use the **Set Print Job Name** action. Document name can be either a fixed value or a variable one. You can construct variable value with **Set Variable** action or **Execute Script Action** and include other variable values in the name like, current date or time, label name, etc.

In a screenshot below you can see 6 documents. First document has the Document Name set to the name of the label that was printed (this is the default behavior), while all the rest have a custom value that is defined by the **Set Print Job Name** action.

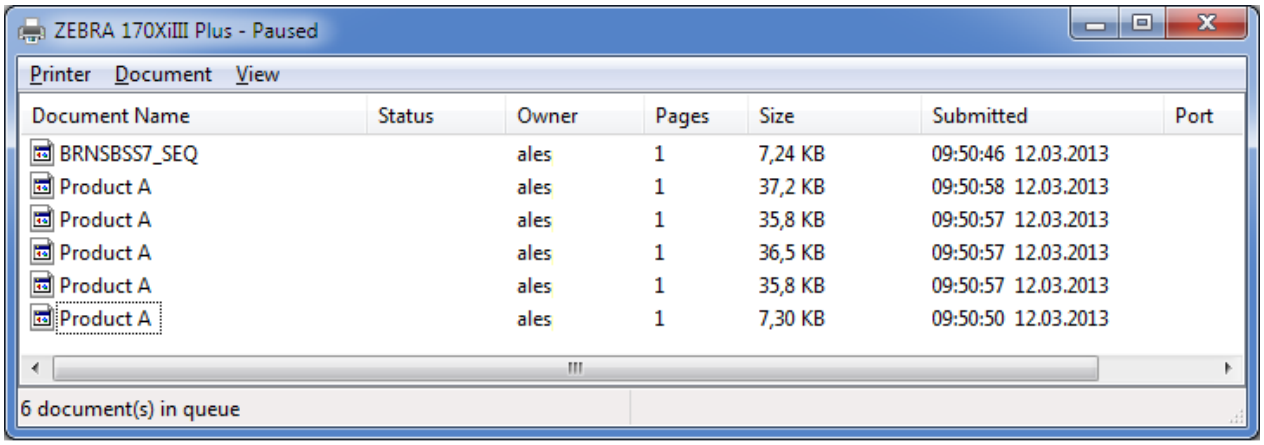


Figure 69 - Windows Spooler - Documents

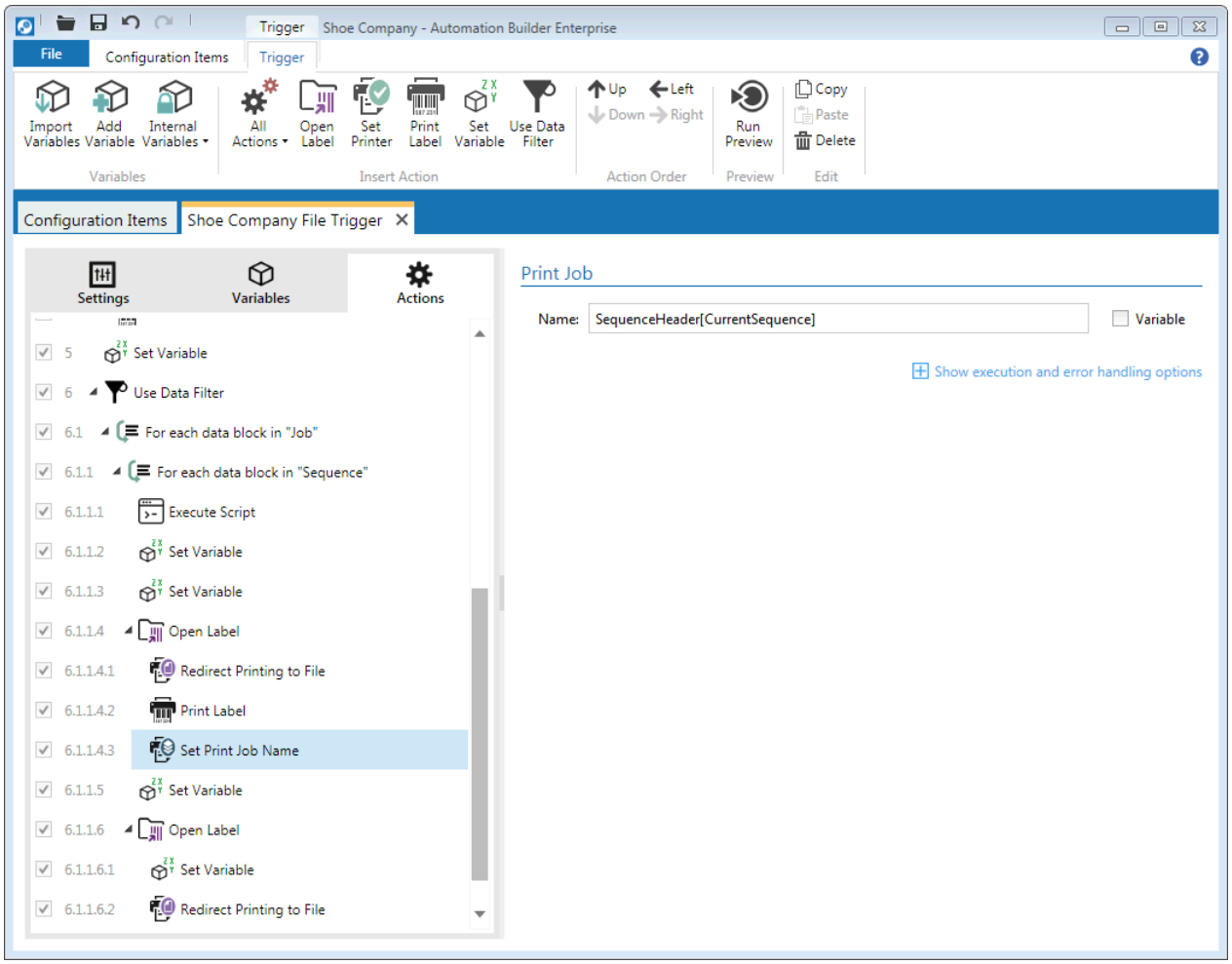


Figure 70 - Automation Builder - Set Print Job Name Action

Redirect Printing to File Action

Redirect Printing to File action saves the output of the **Print Label** action to a file instead of sending it to the printer. The selected file can be overwritten every time the action is executed, or the print stream is appended at the end of an existing file.

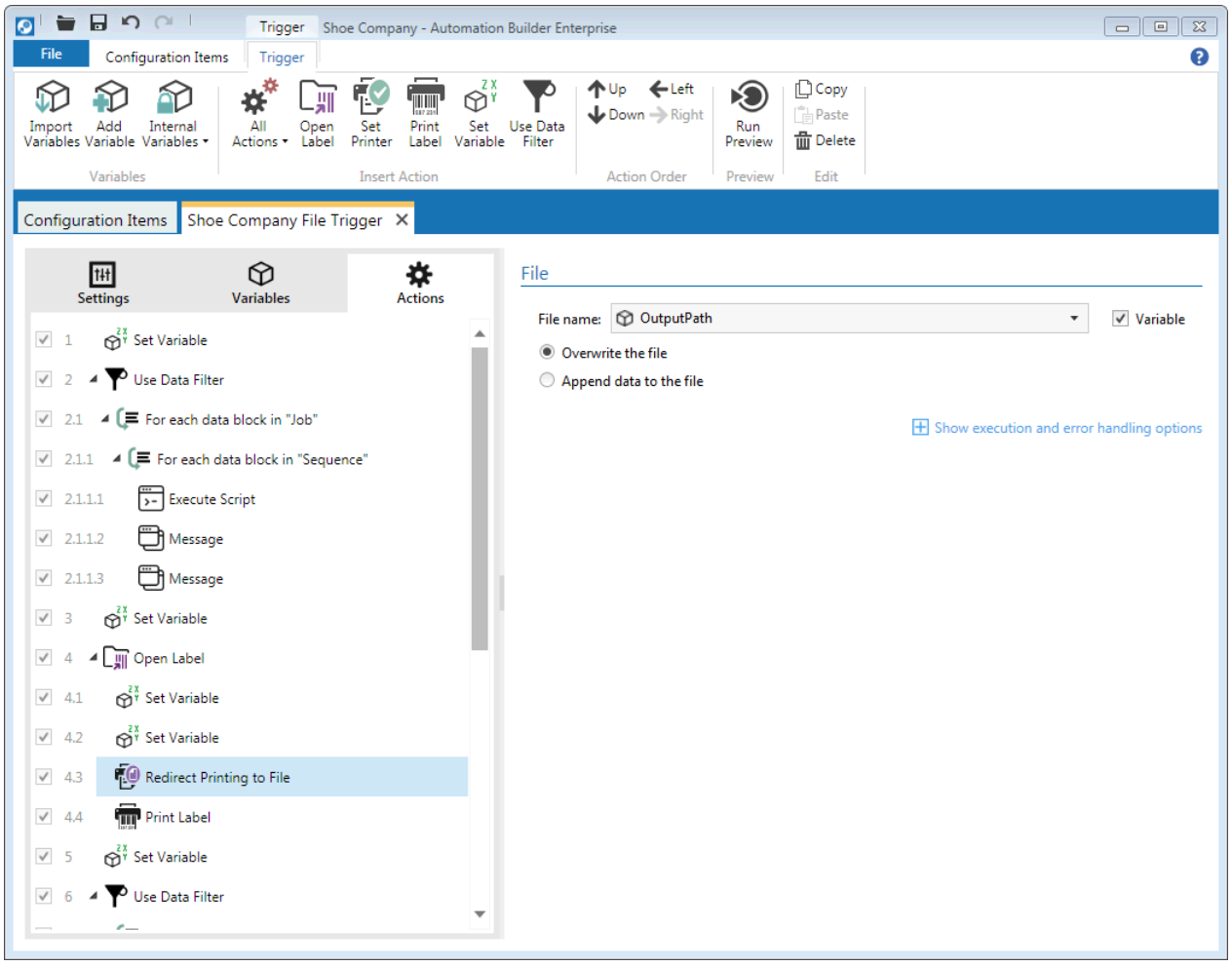


Figure 71 - Automation Builder - Redirect Printing To File Action

Set Print Parameter Action

Set Print Parameter action allows you to set the following print and printer parameters:

- **Paper bin** defines which printer bin with labeling material will be used for printing. This parameter usually applies to laser or ink jet printers. You need to be careful when setting it, because the parameter value has to match to the one defined in the printer driver settings.
- **Print speed** parameter defines with what speed printer will print labels. You should check the driver setting to see which values are valid for the printer you intend to use.
- **Darkness** parameter affects the quality of the printout. Just like with the print speed you need to use a value that printer driver supports.

- **Print offset X**: X means horizontal in this case, so the action will set the horizontal offset (in dots) of the label printout. Setting can also handle negative values.
- **Print offset Y**: Label will be repositioned vertically for the specified number of dots. Again negative value can be used.

All parameters can either be predefined or set from a variable when action is executed.

Note

Action should be used after the Set Printer action and before Print Label action in order to work correctly. Also make sure that you set the correct values for the selected printer, because not all printers have the same settings.

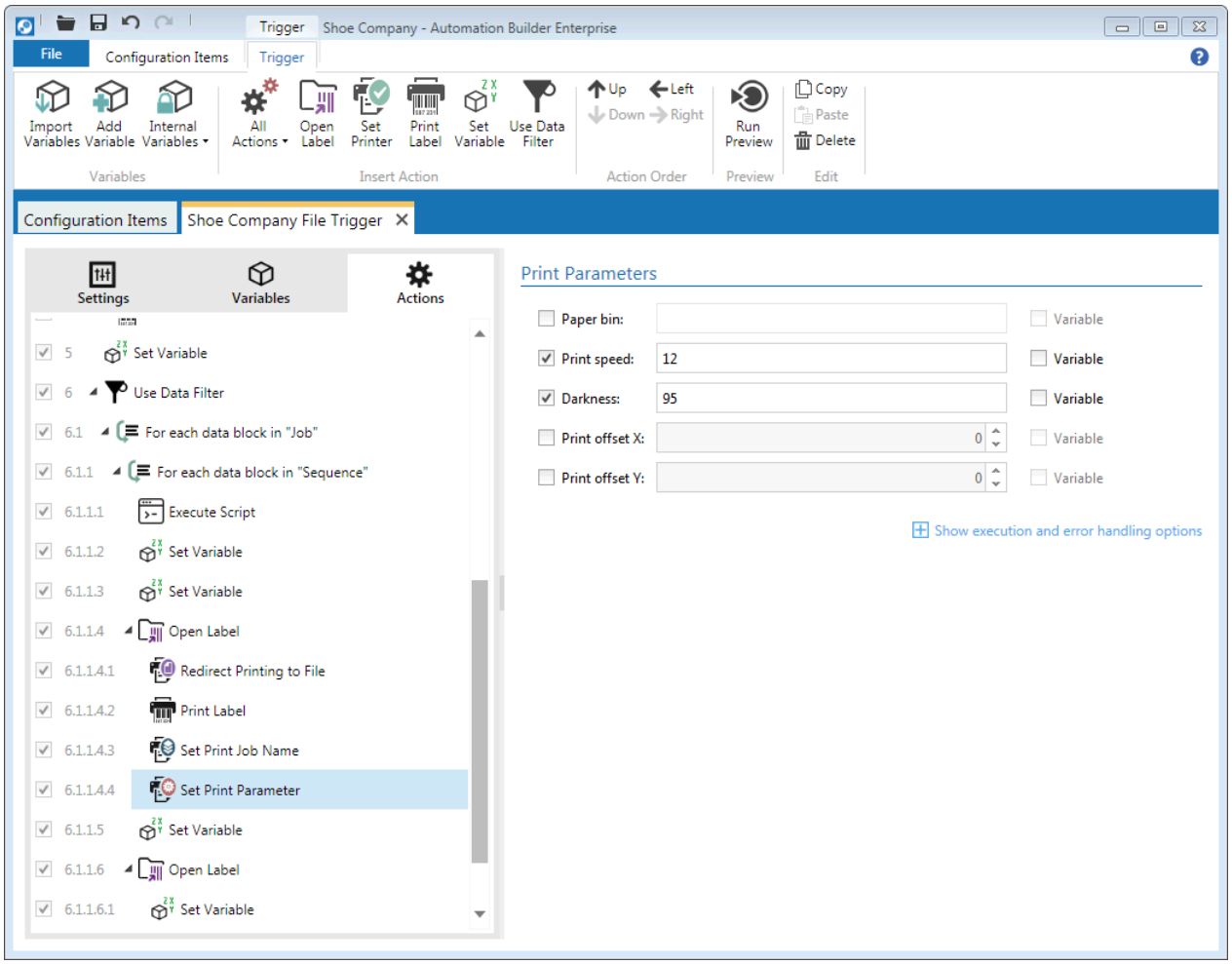


Figure 72 - Automation Builder - Set Print Parameter Action

Printer Status Action

Printer Status action can be used to obtain the information about the printer state. In order for the action to work correctly and return all requested information:

- Selected printer must be capable of reporting the state of the printer.
- Interface to the printer needs to support bi-directional communication.
- NiceLabel printer driver need to be installed for the selected printer.

Some of the information, like **Spooler status** or **Number of jobs in spooler**, can also be returned even if NiceLabel printer driver is not used.

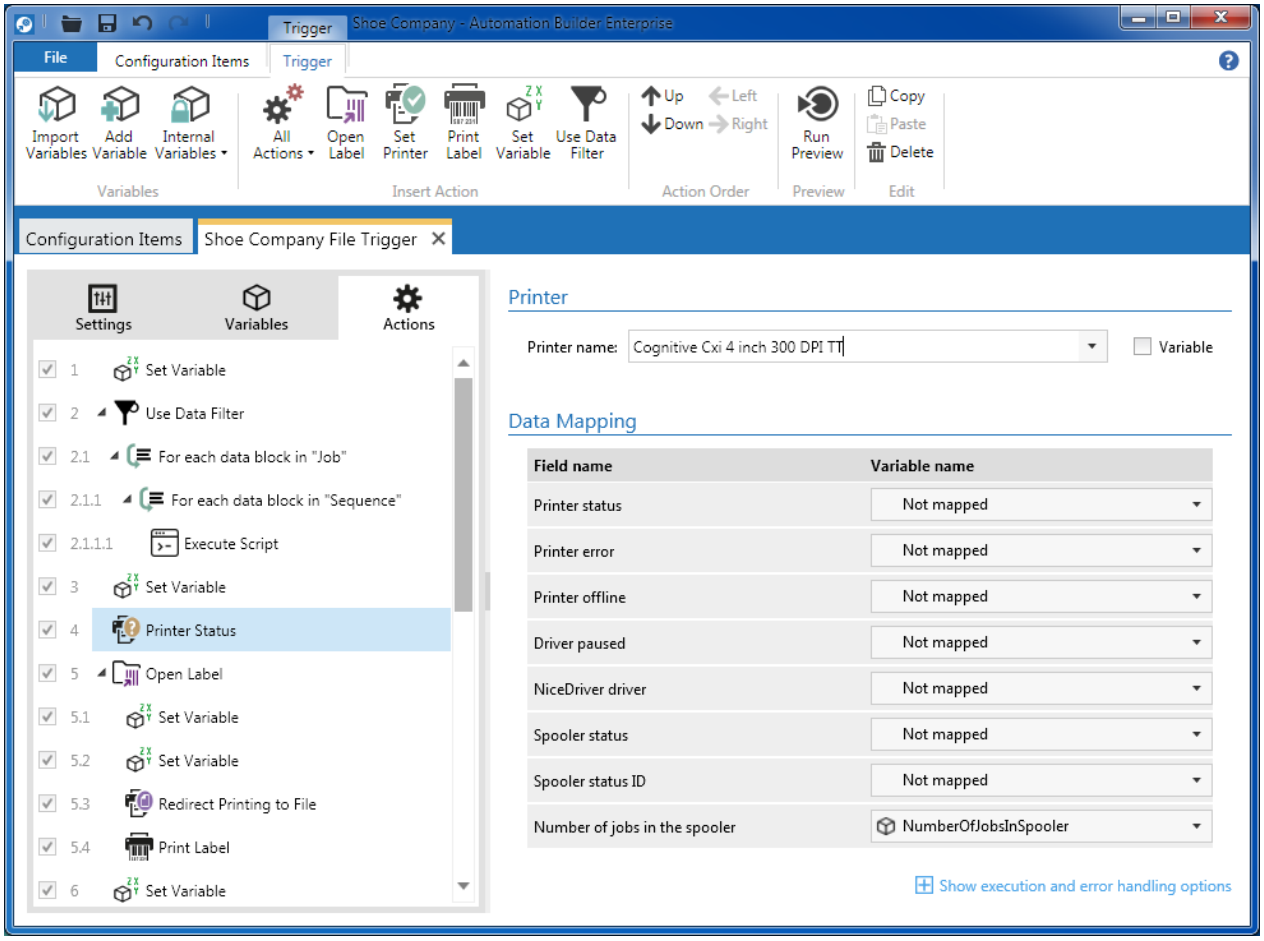


Figure 73 - Automation Builder - Printer Status Action

Printer Status action can query Windows spooler and printer to obtain the following information:

- **Printer status** returns the current status of the printer. Because the printer can be in several states at once, like “Offline”, “Out of labels” or “Ribbon near end”, all these states will be concatenated together, with comma as the delimiter. Because there is no standard that would describe what printer statuses should contain, printers from different manufacturers can report different statuses.

- If selected printer is in the error state, **Printer error** field returns 1. Otherwise 0 is returned.
- **Printer offline** is also a Boolean field that can contain 1 or 0. If printer is offline, then 1 is returned, otherwise it contains 0.
- **Driver paused** tells you if the printer driver is in the paused state. 1 is returned if printer driver is paused, and 0 if not.
- To check if driver for the selected printer is a NiceLabel printer driver, use **NiceDriver driver** field.
- **Spooler status** returns current status of the printer as reported by Windows spooler. Windows spooler can return several statuses at once. In such cases, all statuses will be concatenated , with comma as the delimiter. For example, “Printer is paused., Printer is in error.”.
- Windows spooler can also report a printer status ID. There are four basic values of the spooler status ID, and if the printer is in more than one state, the values are added together. For example, if printer is paused and in error state, **Spooler status ID** field contains would contain value 5.
    - 0 - No status
    - 1 - Printer is paused.
    - 2 - Printer is printing.
    - 4 - Printer is in error.
- **Number of jobs in the spooler** returns the number of print jobs in the Windows spooler for the selected printer.

Every available piece of information about printer status needs to be stored to a variable, if you want to use it.

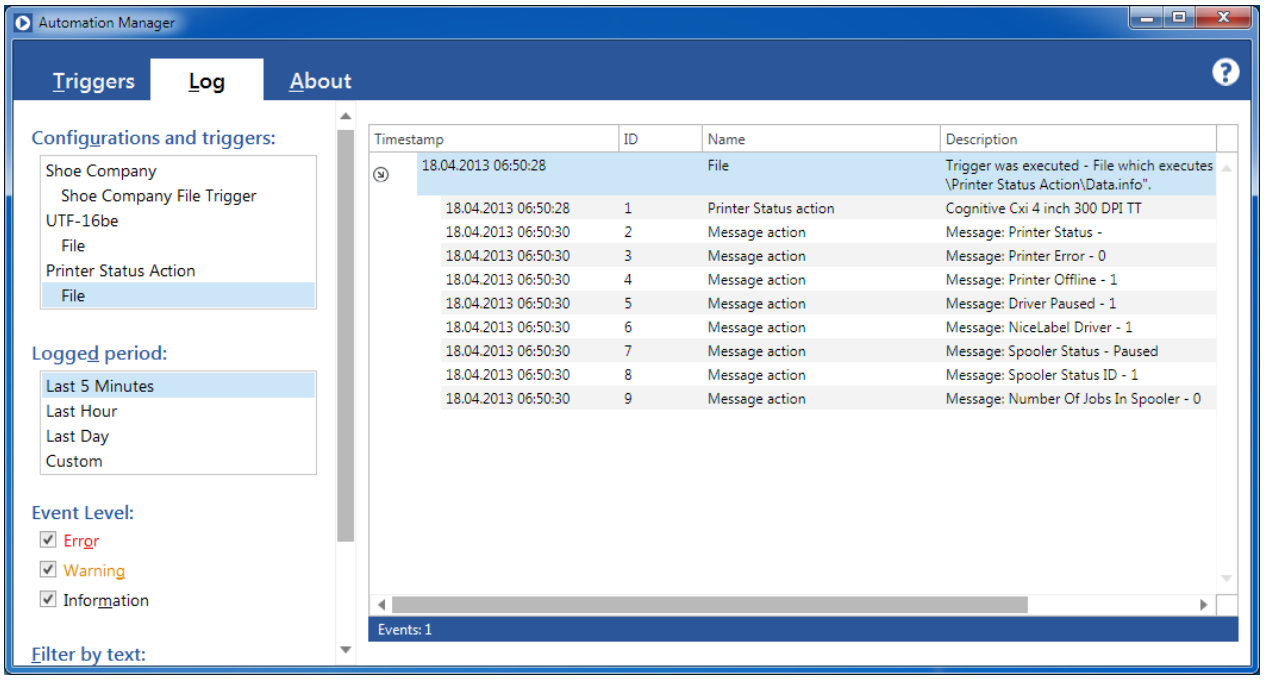


Figure 74 - Automation Manager - Results of Printer Status Action

Set Variable Action

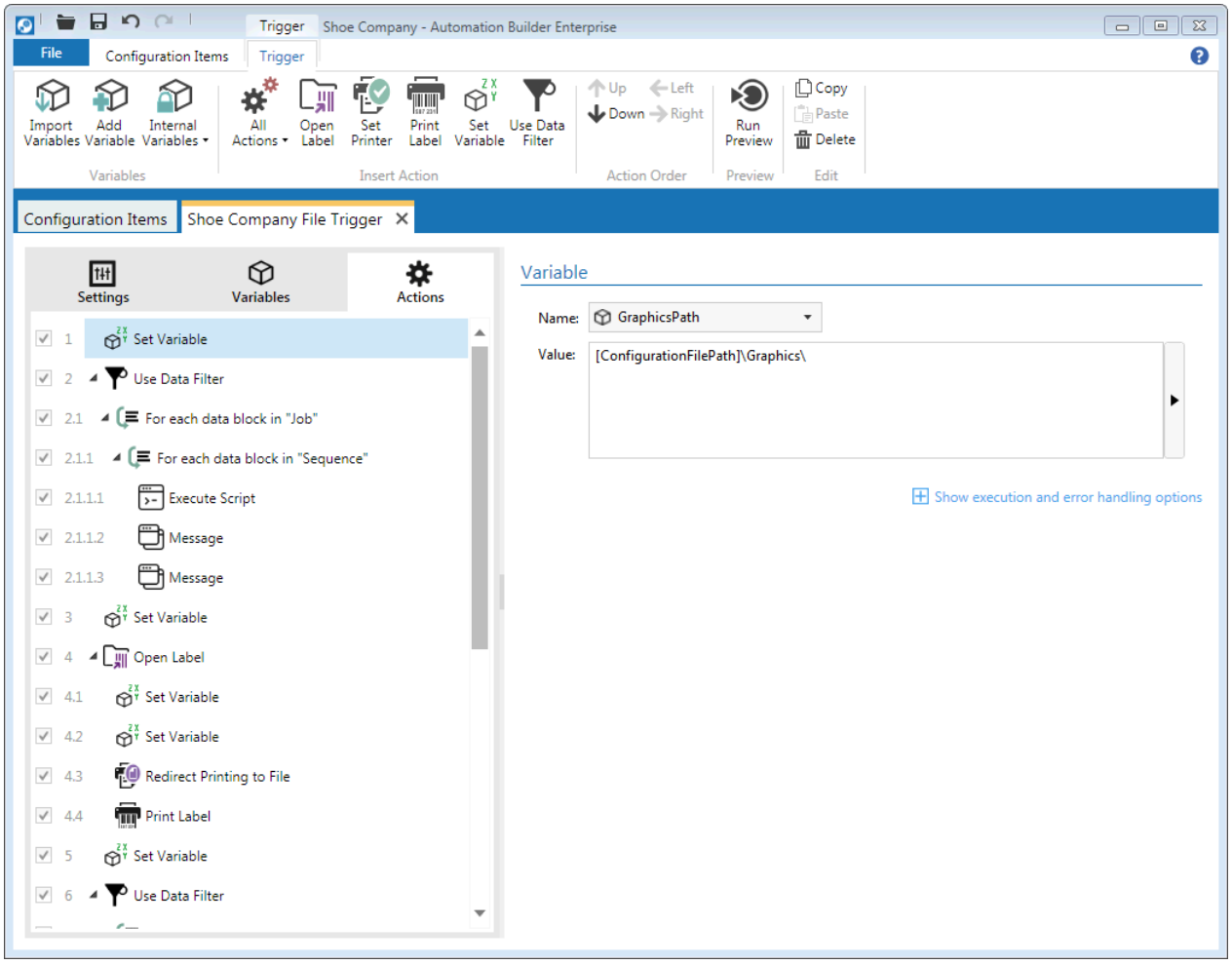


Figure 75 - Automation Builder - Set Variable Action

Of the actions that allow setting variables, **Set Variable** action is probably the simplest, as it has only two properties: name of the variable you want to set, and a value.

The value field can contain fixed contents, variable contents, or the combination of both. See the figure above for a sample how **ConfigurationFilePath** internal variable is used with the combination of predefined folder name to set the value of the GraphicsPath variable.

Special characters are also supported as part of the value.

Load Variable Data Action

Load Variable Data action reads variable values from a text file. Values in the file need to be saved as comma-separated values, but you can define the delimiter, text qualifier and encoding. Delimiter and text qualifier can be selected from a predefined list of possible values, or you can enter your own value. The action can read UTF-8, UTF-16 or ASCII encoded files.

File with variable values can be generated by a third-party application, but is usually created by the **Save Variable Data** action. For example, if you have two file triggers and you might want to transfer some data between them, you could use a combination of **Save Variable Data** action in one trigger, and **Load Variable Data** in the second.

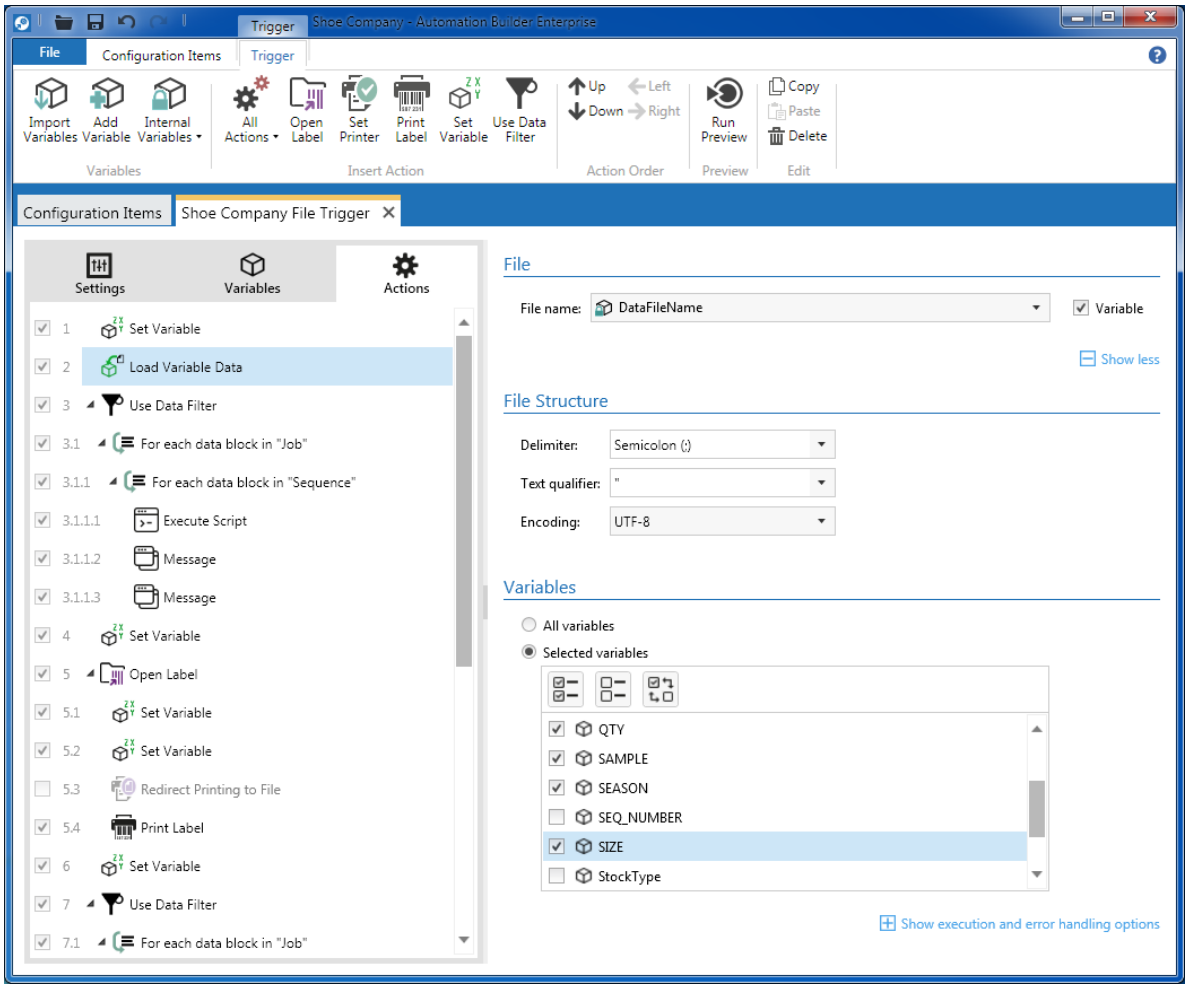


Figure 76 - Automation Builder - Load Variable Data Action

Automation Builder also allows you to define whether you want to load values for all variables, or just for the selected ones. If you want for action to work correctly, you need to ensure that the variable value file has the correct structure (correct header line, number of fields matches to number of selected variables), the correct text encoding is set in the action definition, etc.

Save Variable Data Action

Save Variable Data is the counterpart of the **Load Variable Data** action and they are usually used in combination. First you save variable value, and then you load them. Both actions also share the same set of settings.

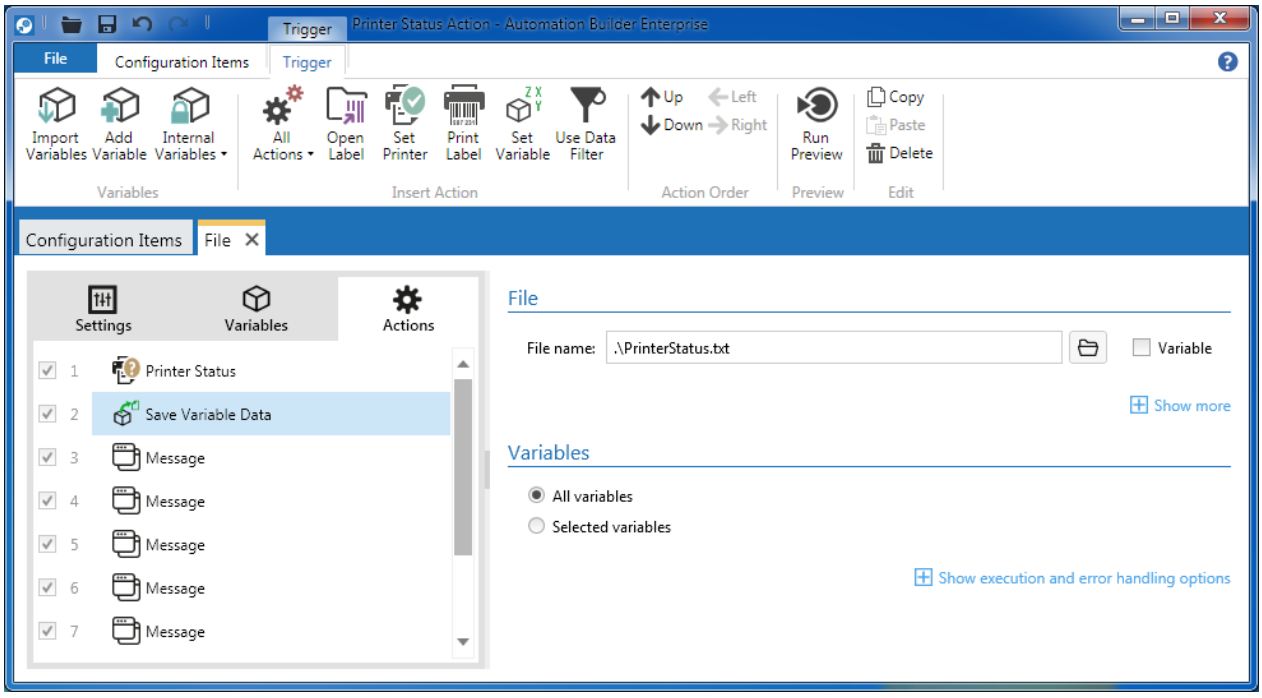


Figure 77 - Automation Builder - Save Variable Data Action

String Manipulation Action

If you need to perform simple text transformations on the values of selected variables, then **String Manipulation** action is the right choice. Action allows you to:

- **Delete spaces at the beginning**.
- **Delete spaces at the end**.
- **Delete opening** and **closing character**. If you have specified "(" as the opening character and ")" as the closing character, and variable contains "*(_some data_)*", you will get "*_some data_*" as the value of the variable, after the action is executed.
- **Search and replace** strings. Just like in word processing applications like Microsoft Word, you define what you want to search for and what it should be replaced with. Additional options are available that narrow the search results:
  - **Match case**: Letter case is taken into consideration while searching. For example "AAAA" is not the same as "aaaa".
  - **Find whole words only**: Searching for "end" will also find "pretend", "intended", "attend", "endless", if this option is not selected.
  - **First occurrence only**: Only the first occurrence of the text you are looking for will be replaced with the new value.
  - **Regular expressions**: Regular expressions provide a flexible and powerful method for processing text. There are lots of different implementations of regular expressions that usually differ in little nuances. Since NiceLabel Automation uses Microsoft .NET framework, it supports the same regular expressions as Microsoft .NET does. Be careful though, regular expressions are not for the faint of heart.

If you click the **What are regular expressions?**, you will be taken to a Microsoft web site where you can read all about regular expressions in .NET Framework.

- **Delete non printable characters**. Non printable or control characters are used to control the interpretation or display of text but do not have a visual or spatial representation. All the data in NiceLabel Automation is Unicode encoded and the Unicode standard assigns the following code points to control characters: from \U0000 to \U001F, \U007F, and from \U0080 to \U009F.
- **Search and delete everything before** a specified value. You can also remove the search string.
- **Search and delete everything after** the specified value. Search string can also be deleted.

If you select more than one option, then string manipulations will be performed in the same order as they are listed in the Automation Builder.

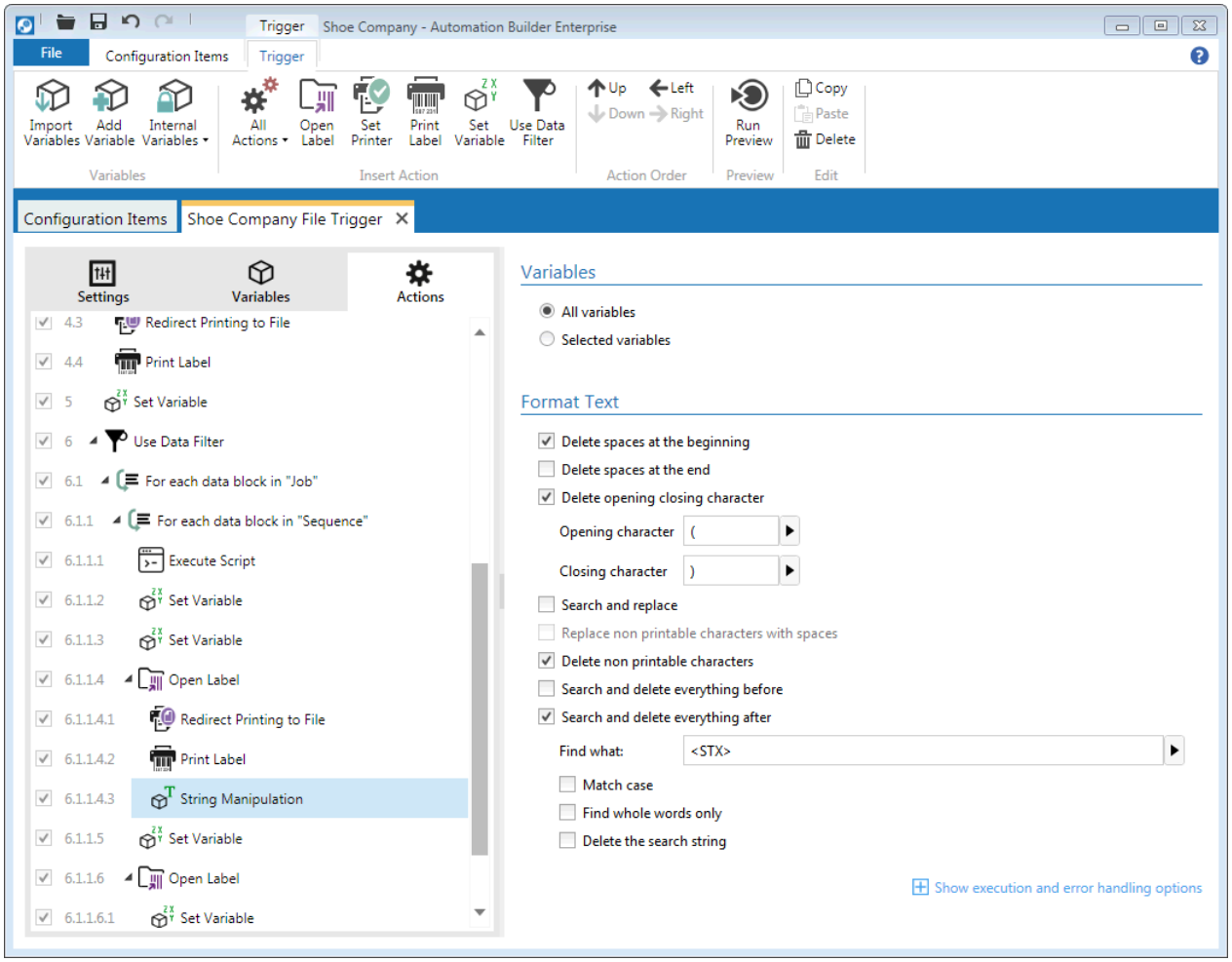


Figure 78 - Automation Builder - String Manipulation Action

Special characters are supported in **Manipulate String** action, so you can for example search for them and replace them with some other values, etc.

Tip

If you need even more complex text processing capabilities, you can use Execute Script action where you can program them yourself in Python or VBScript.

Web Service Action

Web Service action allows you to call web service methods, provide inbound parameter values, and save the values of outbound values into variables. NiceLabel Automation supports XML web services that follow the SOAP standard and use simple data types like Boolean, string, double, integer.

First you need a machine-readable description of the methods offered by the web service. This description needs to be written in the Web Services Description Language (WSDL).

Once Automation Builder fetches the list of available methods, and you select the one you want to use from a drop down list, a list of parameters is displayed. For every parameter you can see if it is an inbound (value is sent to the web service), or an outbound (value is set by the web service) and you can connect them to variables.

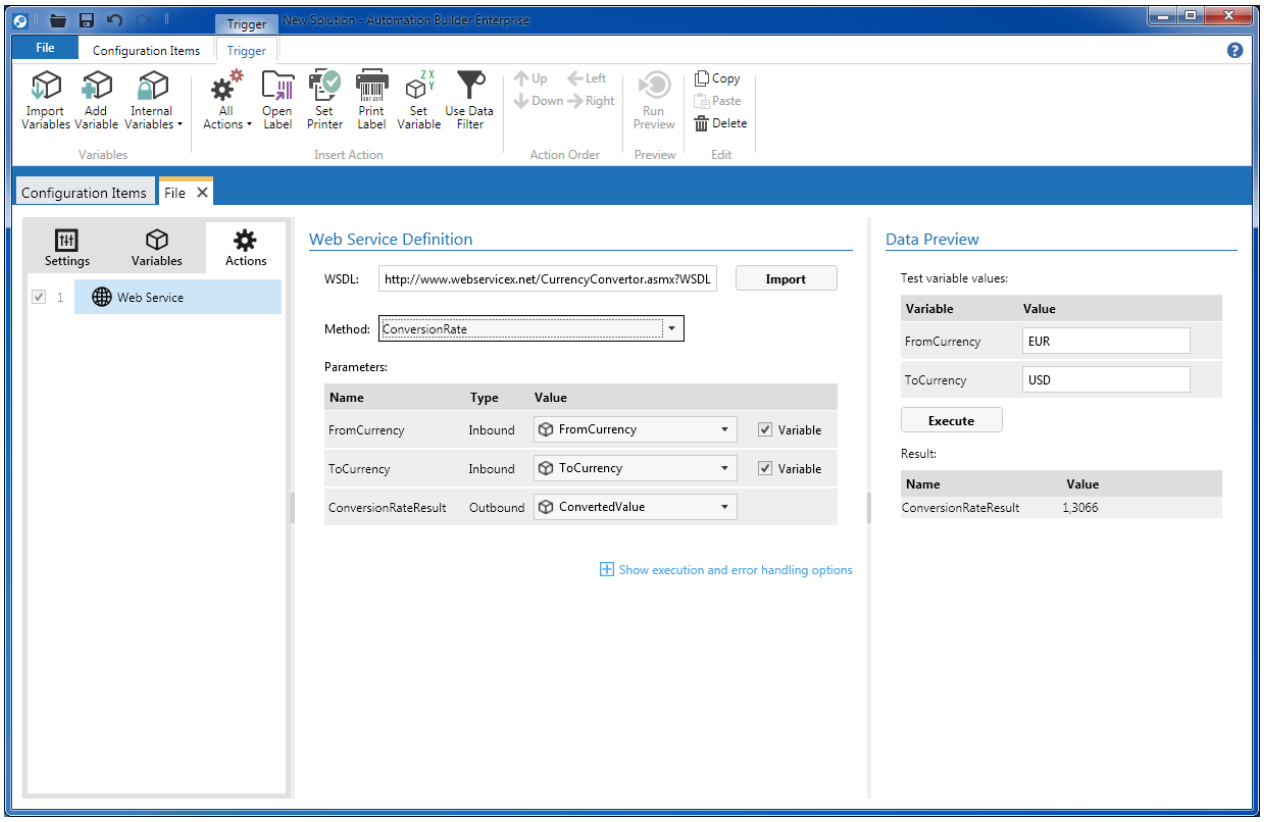


Figure 79 - Automation Builder - Web Service Action

Finally you can test the web service in the **Data Preview** pane by entering values for inbound parameters and clicking **Execute** button. Result of the web service method call (successful or not) is then displayed.

Execute SQL Statement Action

Sometimes you need to get some data into variable directly from a database, or make a change to the database, maybe delete records that are no longer used. **Execute SQL Statement** action can help you with such tasks.

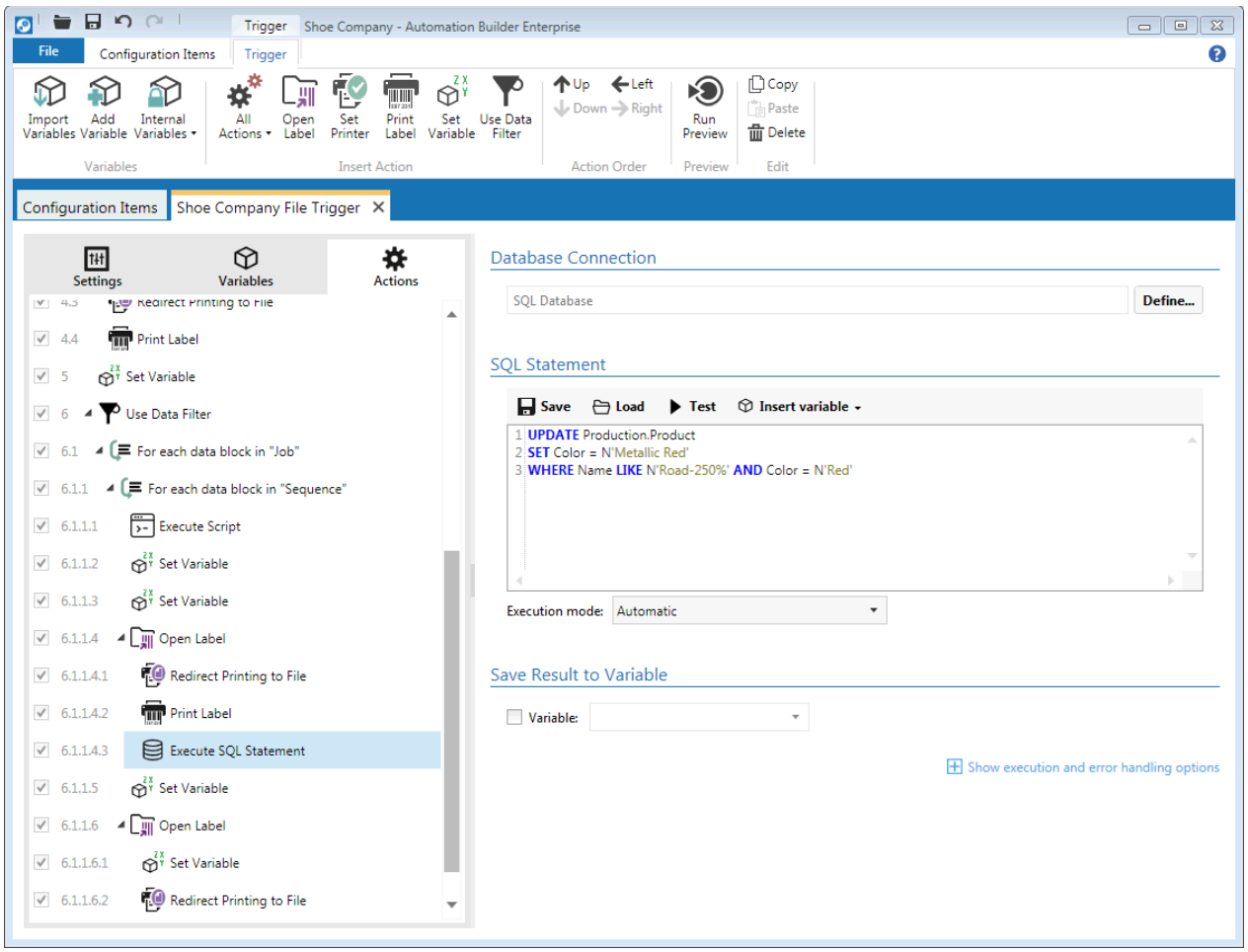


Figure 80 - Automation Builder - Execute SQL Statement Action

After you define the connection to the database (file based databases like Excel spreadsheet or CSV files cannot be used), you can enter the SQL statement. If you have SQL statement already saved somewhere you can either copy and paste it to the editor, or you can simply load it from the file. Saving SQL statement to the file of your choice is also possible by clicking **Save** button.

SQL statement can be completely fixed, or it can contain variable elements. **Insert variable** button can help you quickly add variable to the query. Variable values can only be used as a part of the WHERE clause of the SQL statement.

To preview the results of the SQL statement click the **Test** button. A **Data Preview** pane will appear where you can enter test variable values (if you are using them in the query), and simulate execution of the SQL statement. If you uncheck **Simulate execution,** and use UPDATE, INSERT or DELETE commands, changes will be made to the database. Otherwise only simulation will be made and no permanent changes will be committed to the database.

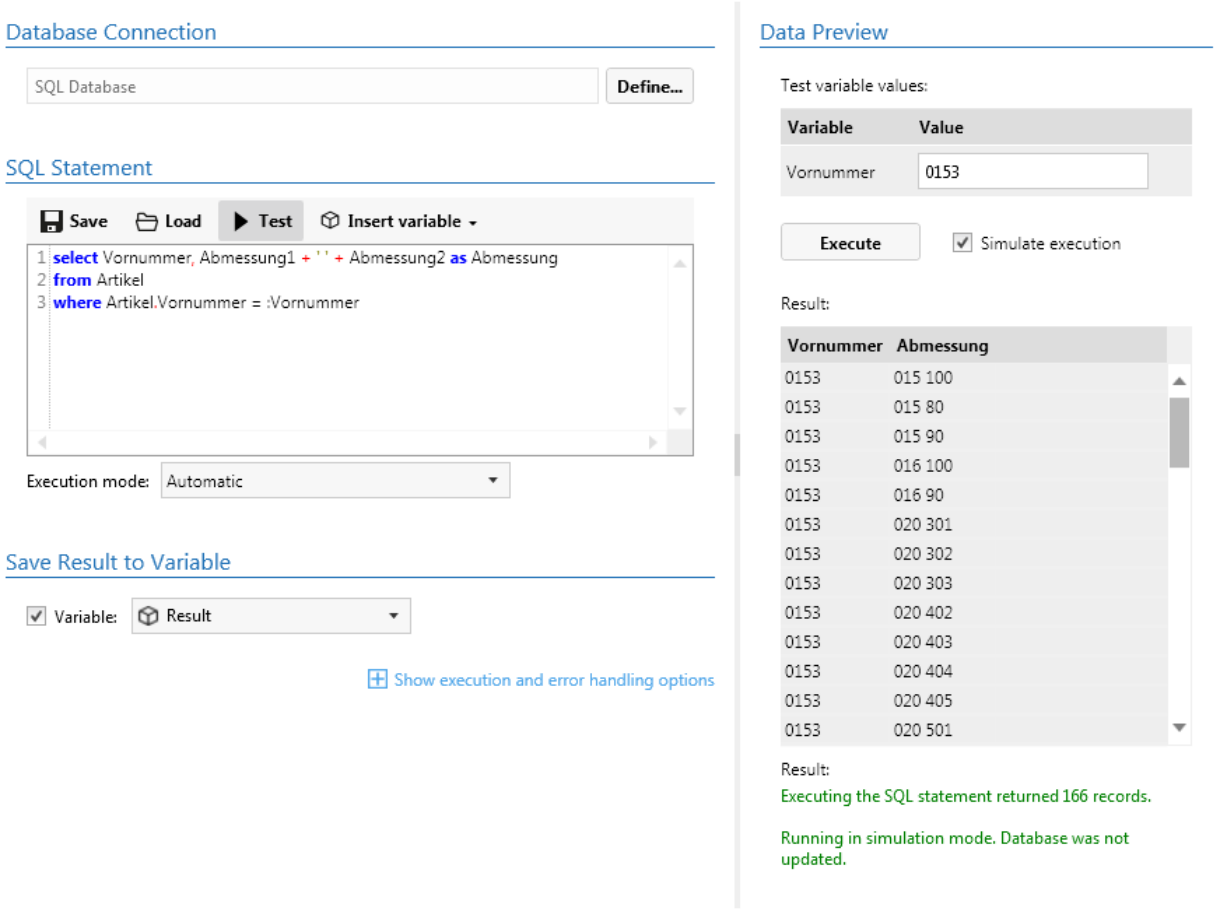


Figure 81 - Automation Builder - Execute SQL Statement Action - Preview

The **Execute SQL Statement** action has three execution modes:

- **Automatic**: NiceLabel Automation tries to automatically figure out what you are trying to do with the SQL command. If this mode is not working for you, because you have a complicated SQL statement, you can select one of the next two options.
- If your SQL query returns a set of records, use **Returns set of records (SELECT)**.
- If you are trying to add records to the database table, delete them, or just update them, then you should select **Does not return set of records (INSERT, DELETE, UPDATE)**.

You can save the result of the SQL statement to a variable. If you use statement that does not return a dataset (INSERT, UPDATE, DELETE), number of affected records will be stored into the selected variable.

If the query returns a set of records, then the contents of the variable depends on the number of records returned. If only one row with one column is returned, then the variable will contain just that value. If there are more rows and/or columns returned, then the variable will contain all those values in a CSV format. For example after SQL statement used in the screenshot above, would be executed, variable Result would contain something like this:

```
"Vornummer","Abmessung"
"0153","015 100"
"0153","015 80 "
"0153","015 90 "
"0153","016 100"
"0153","016 90 "
```

If you need to parse such values you can use **Structured Text** data filter and **Use Data Filter** action.

Note

SQL commands like CREATE DATABASE, CREATE TABLE, or DROP TABLE cannot be used in the Execute SQL Statement action.

Send Data to TCP/IP Port Action

To send data to some other application or service, using TCP/IP protocol, use **Send Data to TCP/IP Port** action.

Destination that will receive the data by the action can be fixed or variable and it should contain both IP address and port. For example if you want to send some data to the service listening on port 67544 at IP address 192.168.1.1, you would enter the following value in the **Destination** field: 192.168.1.1:67544. Firewall must allow traffic on selected port, otherwise action will not work.

Data that you can send with this action can be hardcoded, variable or combination of both. Special characters are also allowed.

As with many other actions you can also select what kind of encoding you would like to use when sending the data: Auto, UTF-8, UTF-16, or ASCII.

Did You Know?

“TCP and IP were developed by a Department of Defense (DOD) research project to connect a number different networks designed by different vendors into a network of networks (the "Internet"). It was initially successful because it delivered a few basic services that everyone needs (file transfer,

electronic mail, remote logon) across a very large number of client and server systems. Several computers in a small department can use TCP/IP (along with other protocols) on a single LAN. The IP component provides routing from the department to the enterprise network, then to regional networks, and finally to the global Internet. On the battlefield a communications network will sustain damage, so the DOD designed TCP/IP to be robust and automatically recover from any node or phone line failure. This design allows the construction of very large networks with less central management."

Source: http://www.yale.edu/pclt/COMM/TCPIP.HTM (Gilbert)

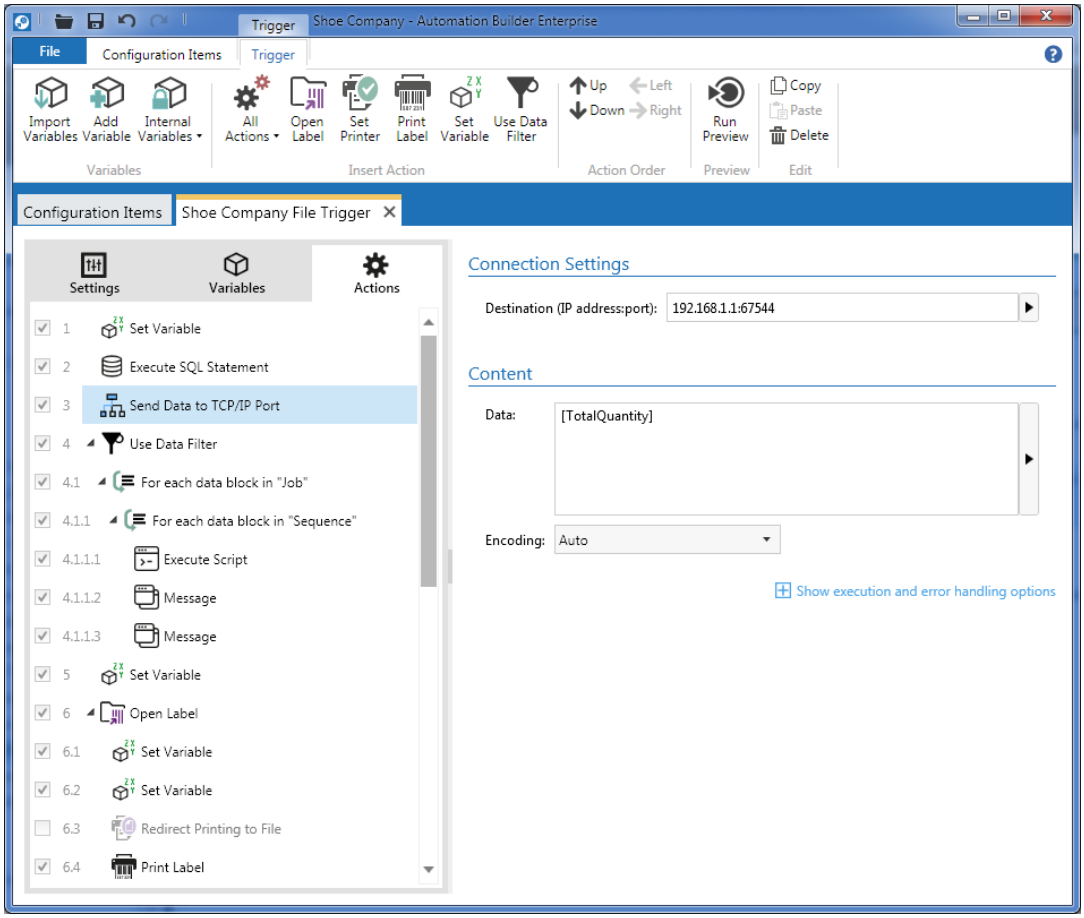


Figure 82 - Automation Builder - Send Data to TCP/IP Port Action

Send Data to HTTP Action

Send Data to HTTP action allows you to send fixed, variable, or combination of fixed and variable data to a service (usually to the Web server) using the HTTP protocol and POST request method. Both http:// and https:// web addresses (URL) are supported.

When defining this action you need to specify several parameters:

- **Destination** defines the address of the service receiving the data. As with **Send data to TCP/IP port** action, you need to enter both the address and port. If the (default) port 80 is used, you can omit it. Again you must open port in the firewall for action to work.
- Since server might not answer in a timely manner, you can define the **Timeout** period. NiceLabel Automation will wait the specified amount of milliseconds and then terminate the connection if the server does not respond.
- Web servers reply to every request with a response. If you want to use that feedback, select **Wait for status reply**.
- **Save status reply in a variable settings**: If **Wait for status reply** is selected you can then select which variable should be used to get the HTTP status code.

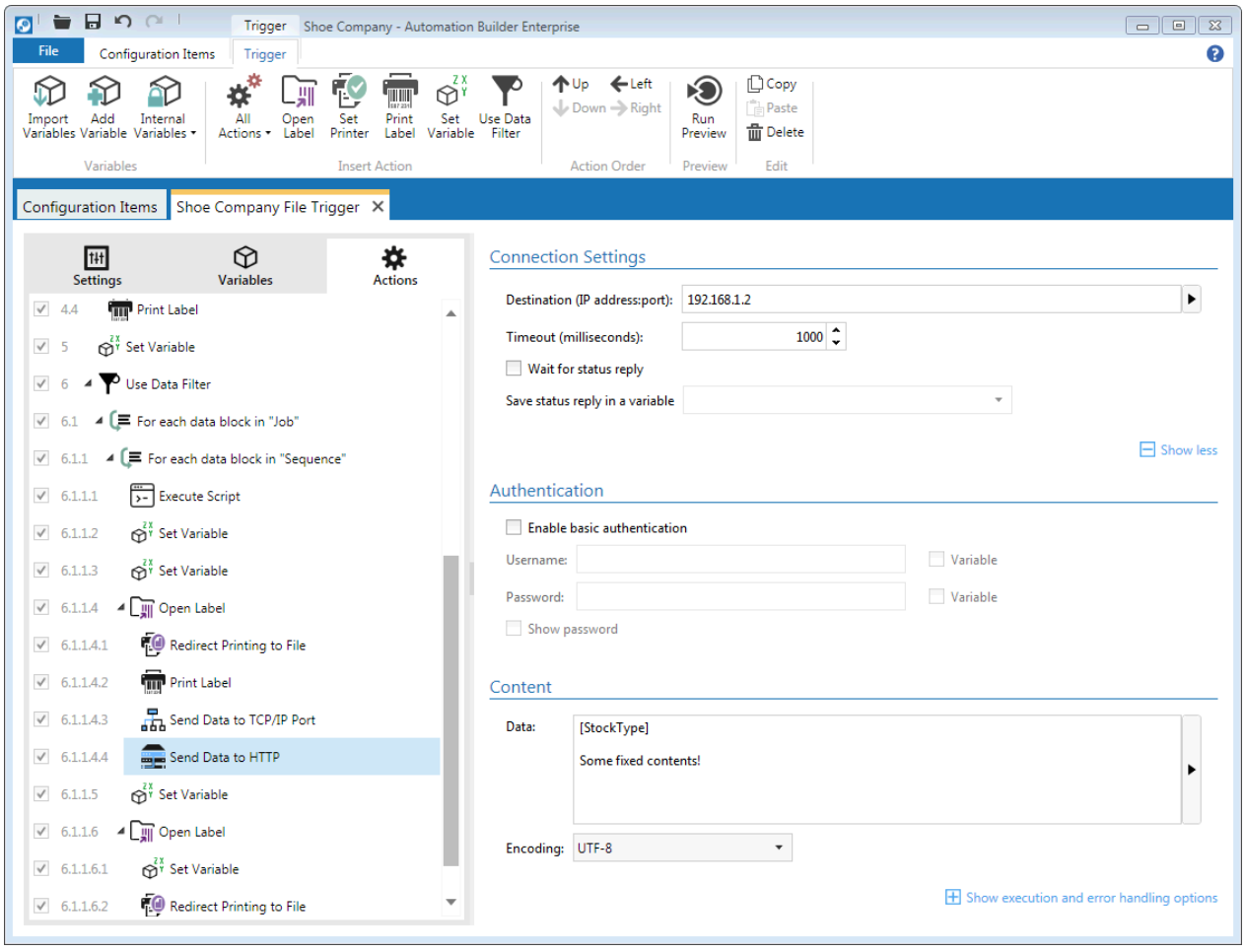


Figure 83 - Automation Builder - Send Data to HTTP Action

If web server (or some other service on the other side of the communication) requires user name and password, you can define them in the **Authentication** section of the action properties.

Send Data to Printer Action

To send pre-generated print streams or any other data to arbitrary printer, use **Send Data to Printer** action.

You can select a printer from a list of locally installed printers, or enter any printer name (this is handy if you plan to deploy configuration to some other computer with a printer that you do not have installed on your computer), or configure action to get the printer name from a variable.

You can send either data received by the trigger, contents of a file (path to the file can be variable or fixed), value of a variable, or custom contents. The custom contents can contain any combination of hardcoded data, including special characters and variable values.

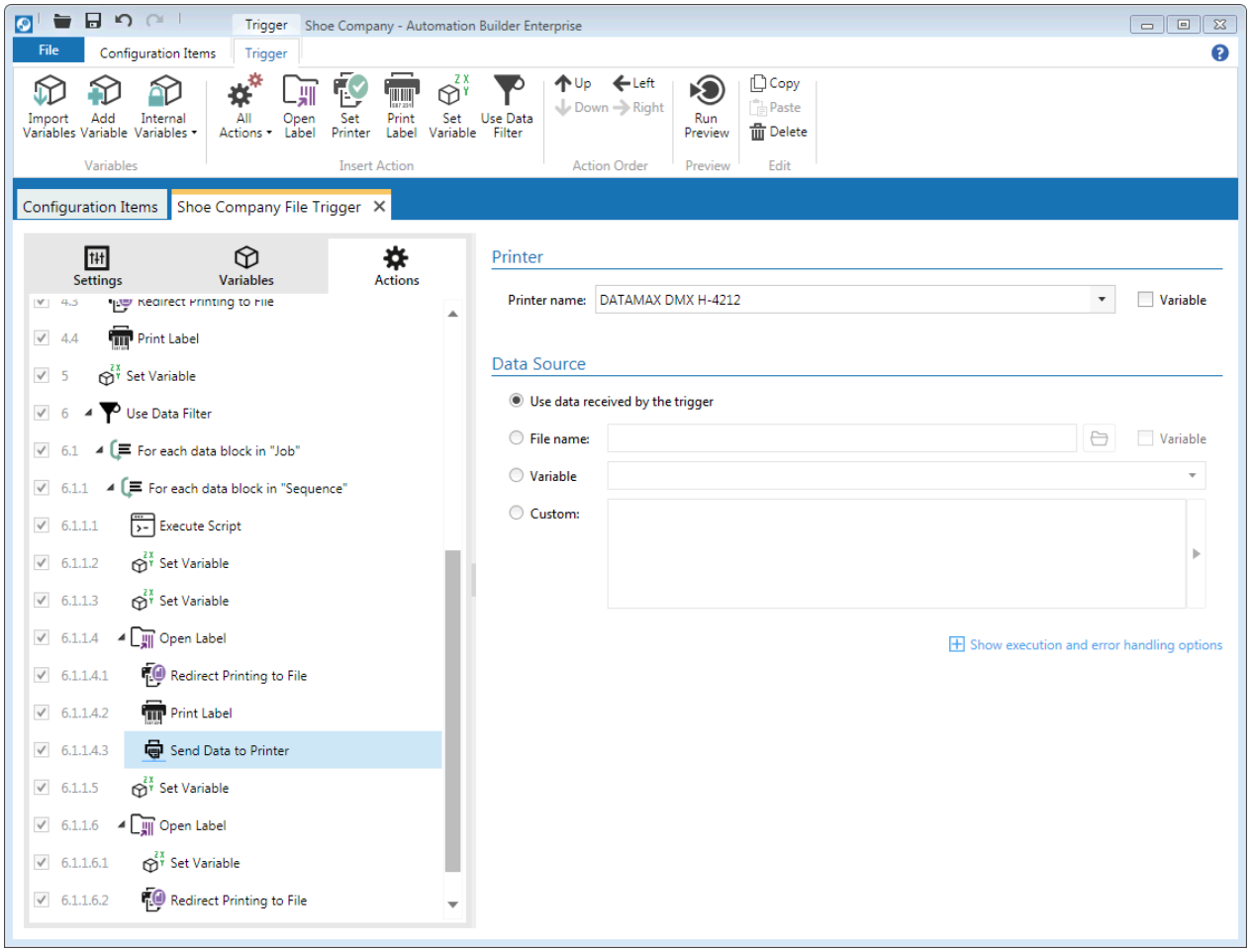


Figure 84 - Automation Builder - Send Data to Printer Action

Save Data to File Action

Save Data to File action saves arbitrary data to the predefined file or variable one. When the data is saved to the file, the file can be overwritten every time (previous contents will be lost), or the data is appended at the end of existing contents.

You can save either data received by the trigger, or some custom contents. The custom contents can contain any combination of hardcoded data including special characters and variable values.

To add variable or special character to the contents click a small button with black arrow to the right of the **Contents** field, and select it from the drop down menu.

Data saved to the file can be encoded as UTF-8, UTF-16 or ASCII.

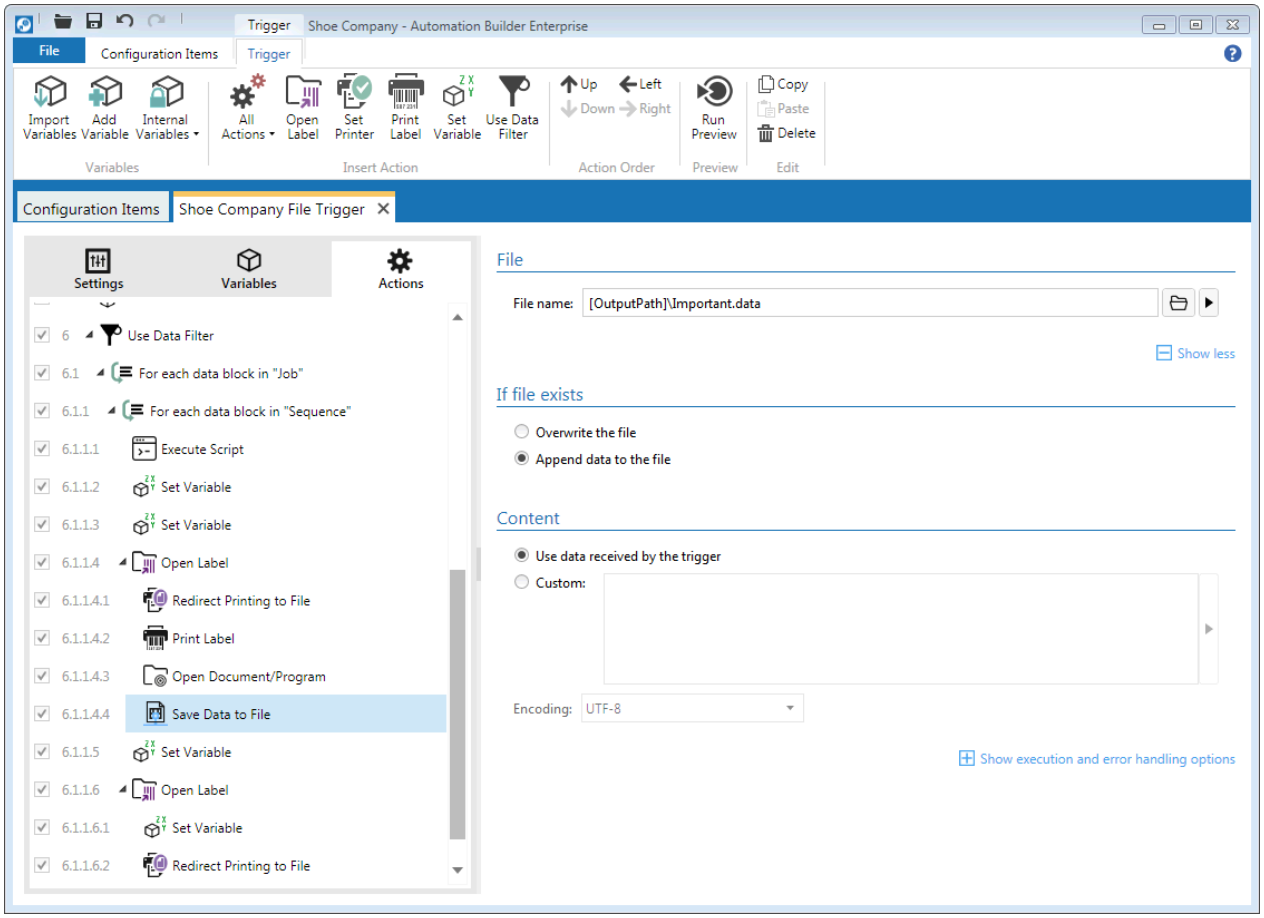


Figure 85 - Automation Builder - Save Data to File Action

Delete File Action

Delete File action deletes a file on the disk. The file name that tells the action which file to delete can be fixed like “C:\Trigger\Data\Info.data”, or a variable.

Once again a word of warning. Because NiceLabel Automation runs as a service it might not have the same permissions as the user that built the configuration, so make sure that the permissions are configured correctly. If the file cannot be deleted there will be no errors reported. Also if you intend to delete file on a network share, it would be best if you used an UNC notation for the file path and not the mapped drive letter, as it might not be accessible to the NiceLabel Automation service. So use \\ComputerName\SharedFolder\Resource instead of “M:\SharedFolder\Resource”, where M is mapped to \\ComputerName.

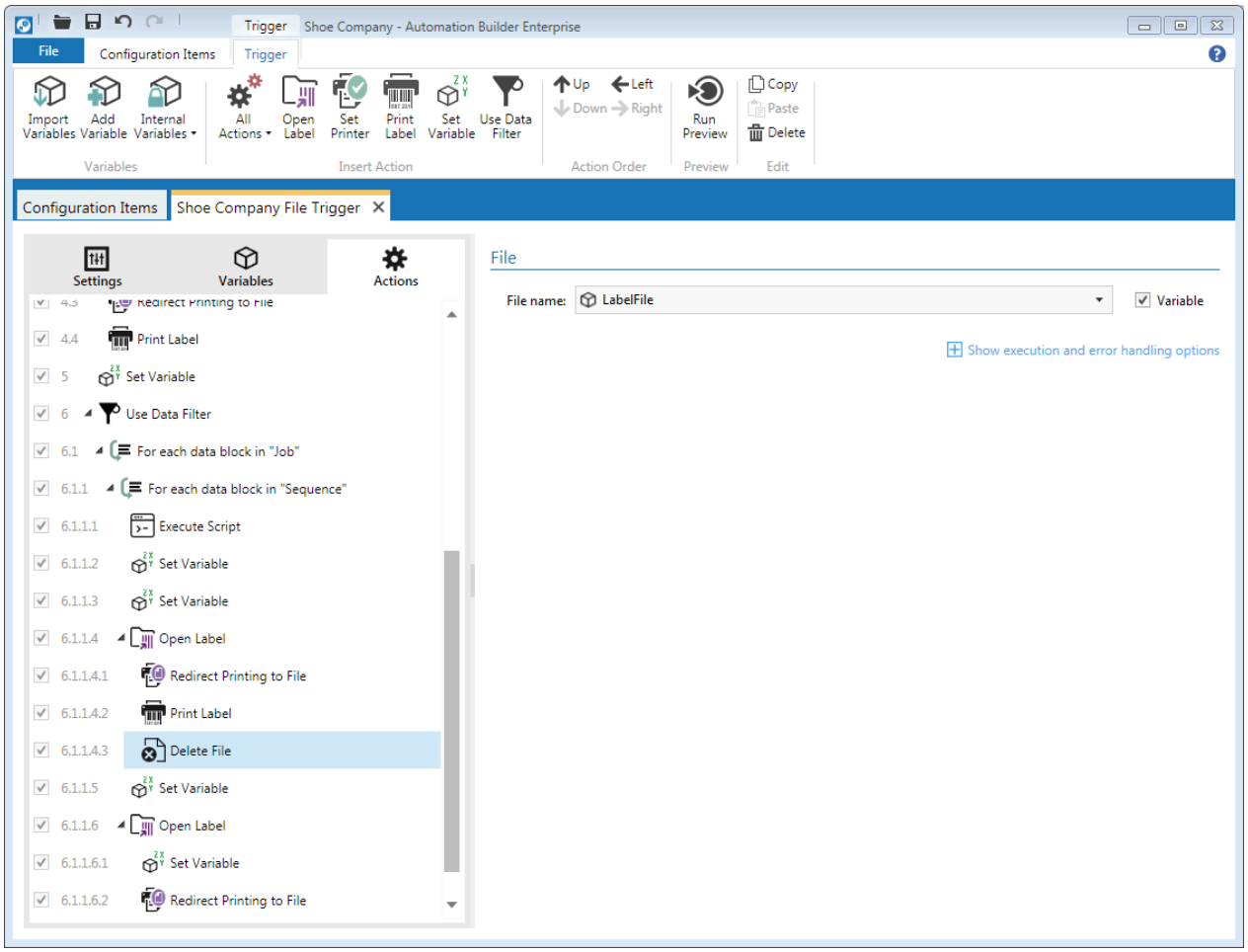


Figure 86 - Automation Builder - Delete File Action

Read Data from Serial Port Action

NiceLabel Automation and **Read Data from Serial Port** action allow you to connect to serial port devices like barcode scanners and read the input they provide. When you select the serial port, you also have to define the port settings:

- Speed (Bits per second)
- Data bits
- Parity
- Stop bits
- Flow control

For more detailed description of these settings see the chapter on **Serial Port Trigger**.

Sometimes you need to wait some time before you can read from the serial port. For such cases you can define the **Read delay** property.

To send some data to the serial device before the action starts reading from it, enable **Send initialization data** and define which characters you would like to send.

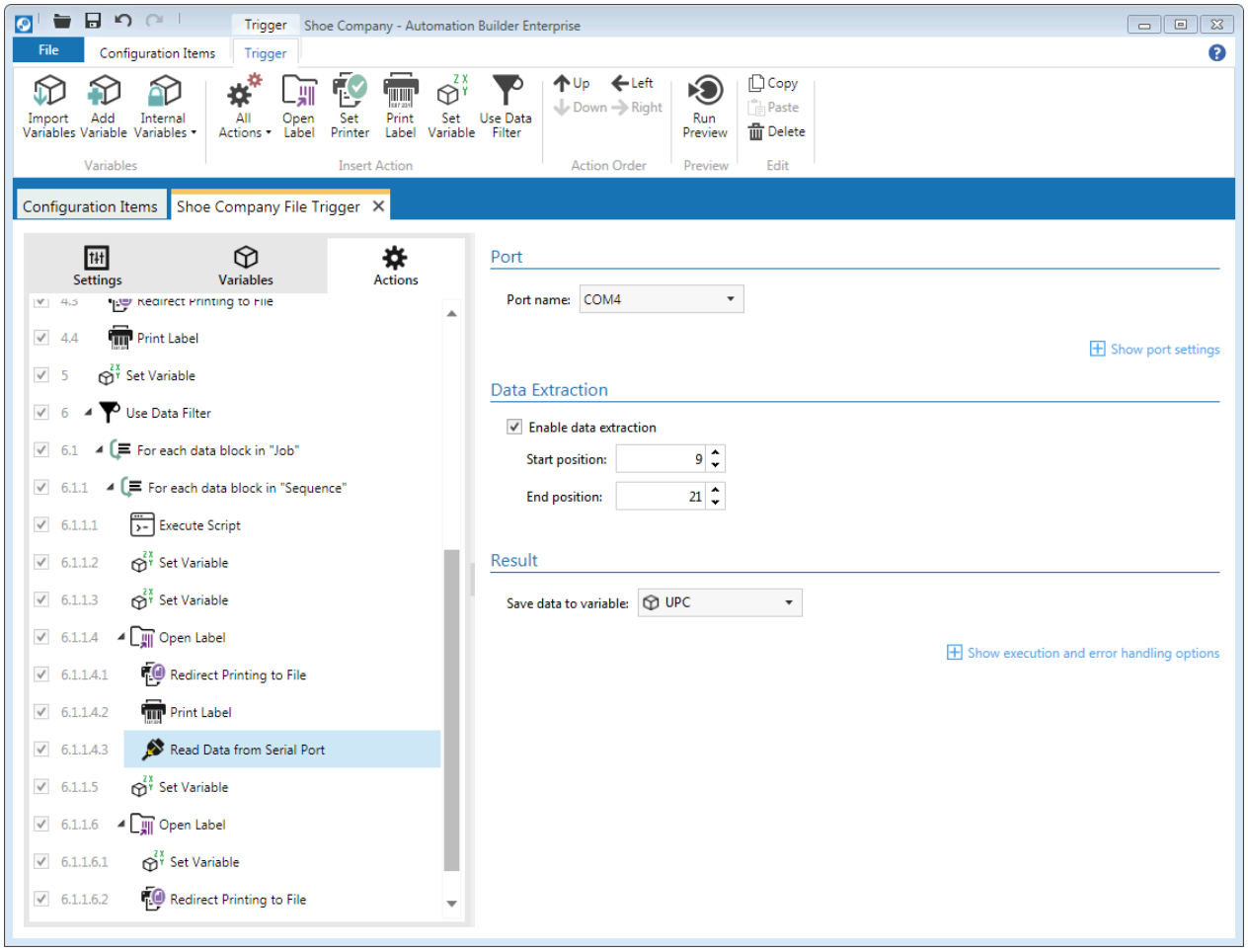


Figure 87 - Automation Builder - Read Data from Serial Port Action

When the data stream from the serial device contains more characters than you need, you can extract the useful ones by checking the **Enable data extraction** option and setting the start and end position.

To use the data that comes from the serial port in data filters or other actions, you need to save it to a variable.

Send Data to Serial Port Action

Send Data to Serial Port action sends fixed data, variable contents or combination of both, to a serial device.

For more information about serial port settings see the sections about **Serial Port Trigger** and **Read Data from Serial Port** action.

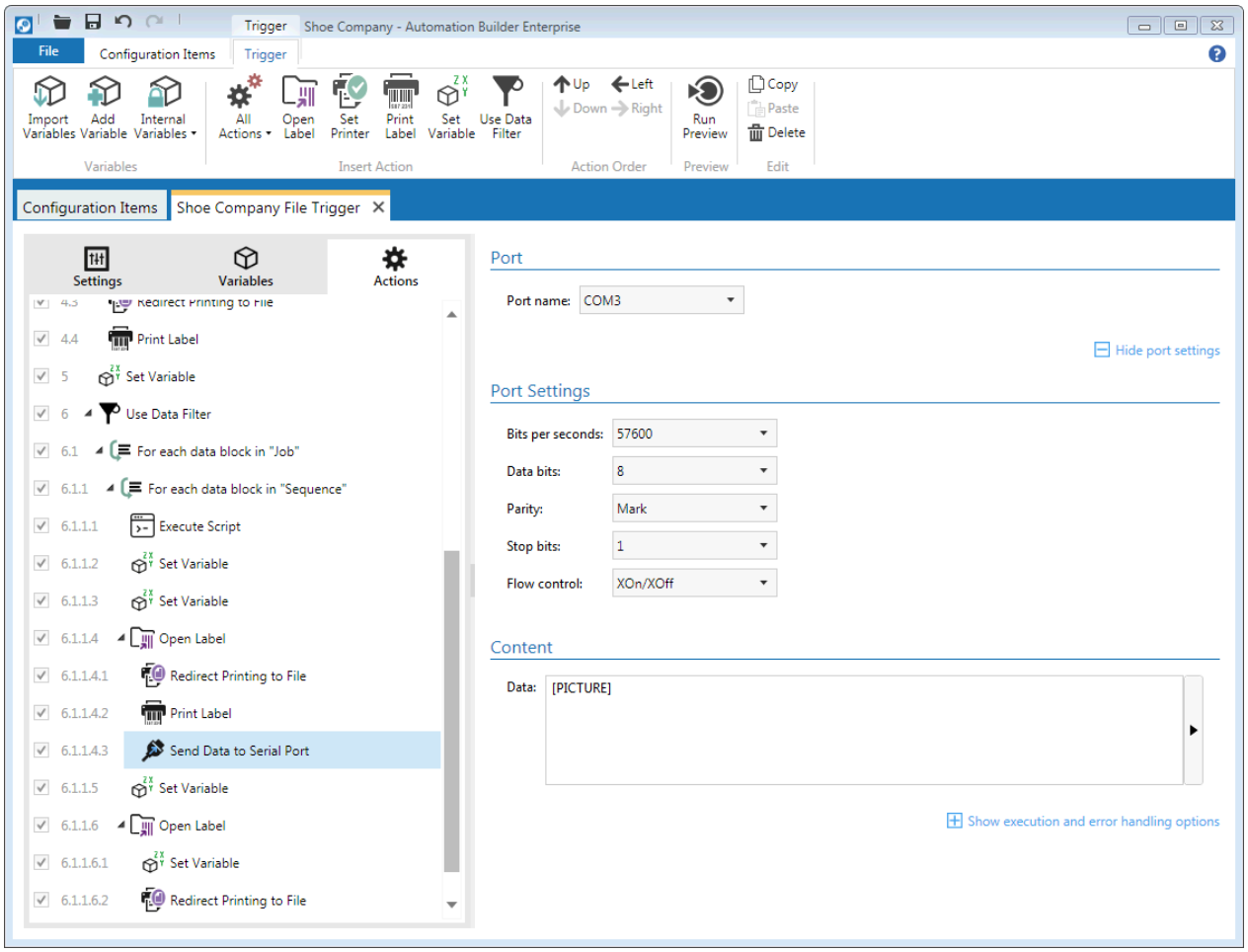


Figure 88 - Automation Builder - Send Data to Serial Port Action

For Loop Action

For Loop action repeatedly executes all child actions defined beneath it. For loops are usually used when the number of iterations is known before the action is executed. Both the start and end value can be hardcoded values or come from variables. Variable used in **For Loop** action needs to contain integer numeric values.

Did you know?

The name *for loop* comes from the English word *for*, which is used as the keyword in most programming languages to introduce a for loop. The loop body is executed "for" the given values of the loop variable.

Source: http://en.wikipedia.org/wiki/For_loop (Wikipedia)

If you need to know or use the value of the current step of the loop, you can save it to a variable. This variable value can then be used in other actions.

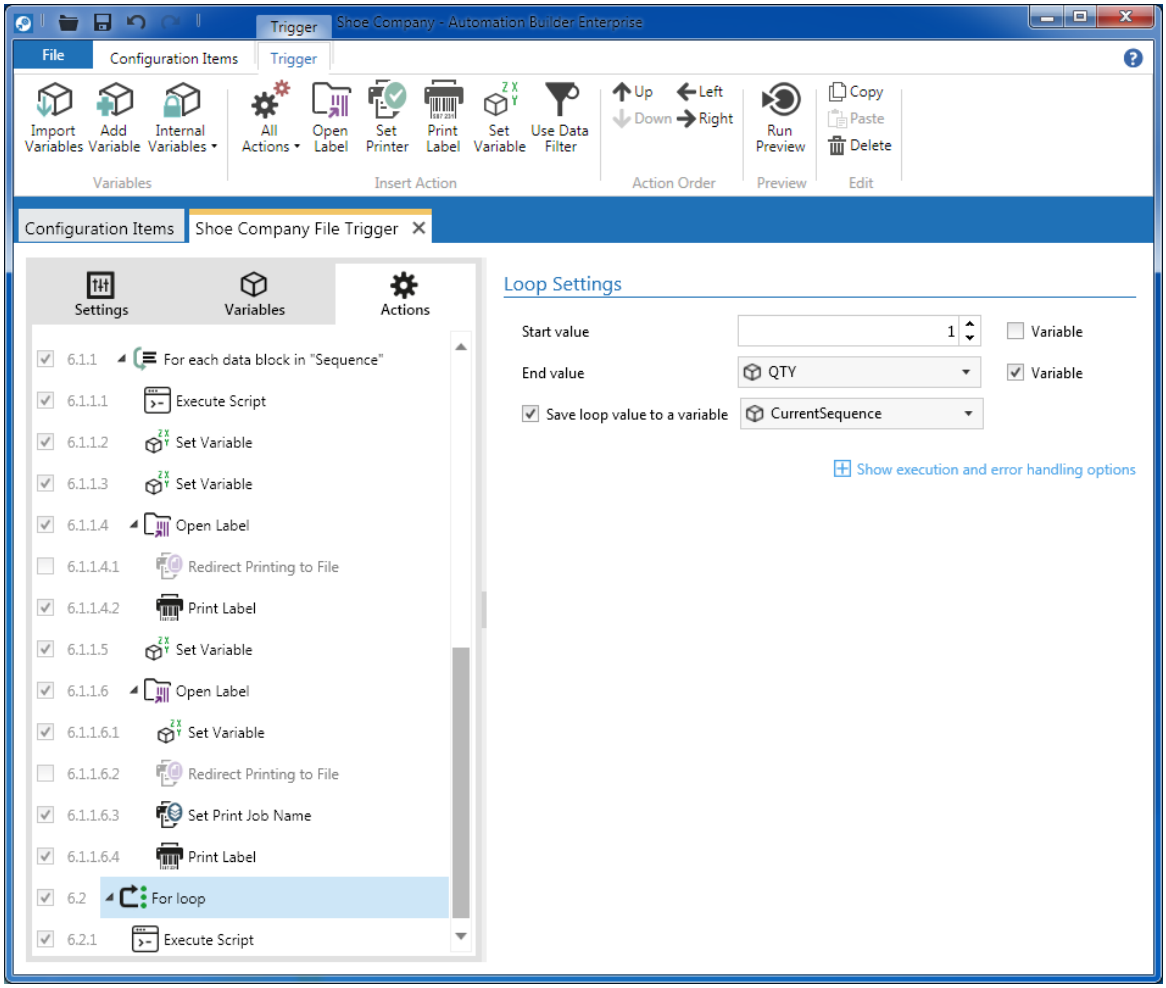


Figure 89 - Automation Builder - For Loop Action

Open Document / Program Action

In a situation where NiceLabel Automation does not have functionality that you need, or you have a third-party application you need to interact with, you can use **Open Document / Program** action. This way you can run batch files (.bat), Windows executable files (.exe), VBScript scripts (.vbs), etc.

Usually such programs or scripts need some parameters passed to them. You can define them in the **File name** property. For example, using

```
"C:\Program Files\MyApplication\myapplication.exe" /s [Param2]
```

in the **File name** setting would open myapplication.exe that is located in the "C:\Program Files\MyApplication\" folder and pass two parameters to it. First parameter ("/s") is a fixed value, while the second one gets its value from variable Param2.

If you have configured NiceLabel Automation service to run under Local System account and is allowed to interact with desktop, application window can be shown to the user. If you do not want that you can set **Hide Window** option. By default NiceLabel Automation service cannot interact with the desktop so the application run by the **Open Document / Program** action will not be visible.

Action will wait for the program to finish running, before moving to another action, if you check the **Wait for completion** setting.

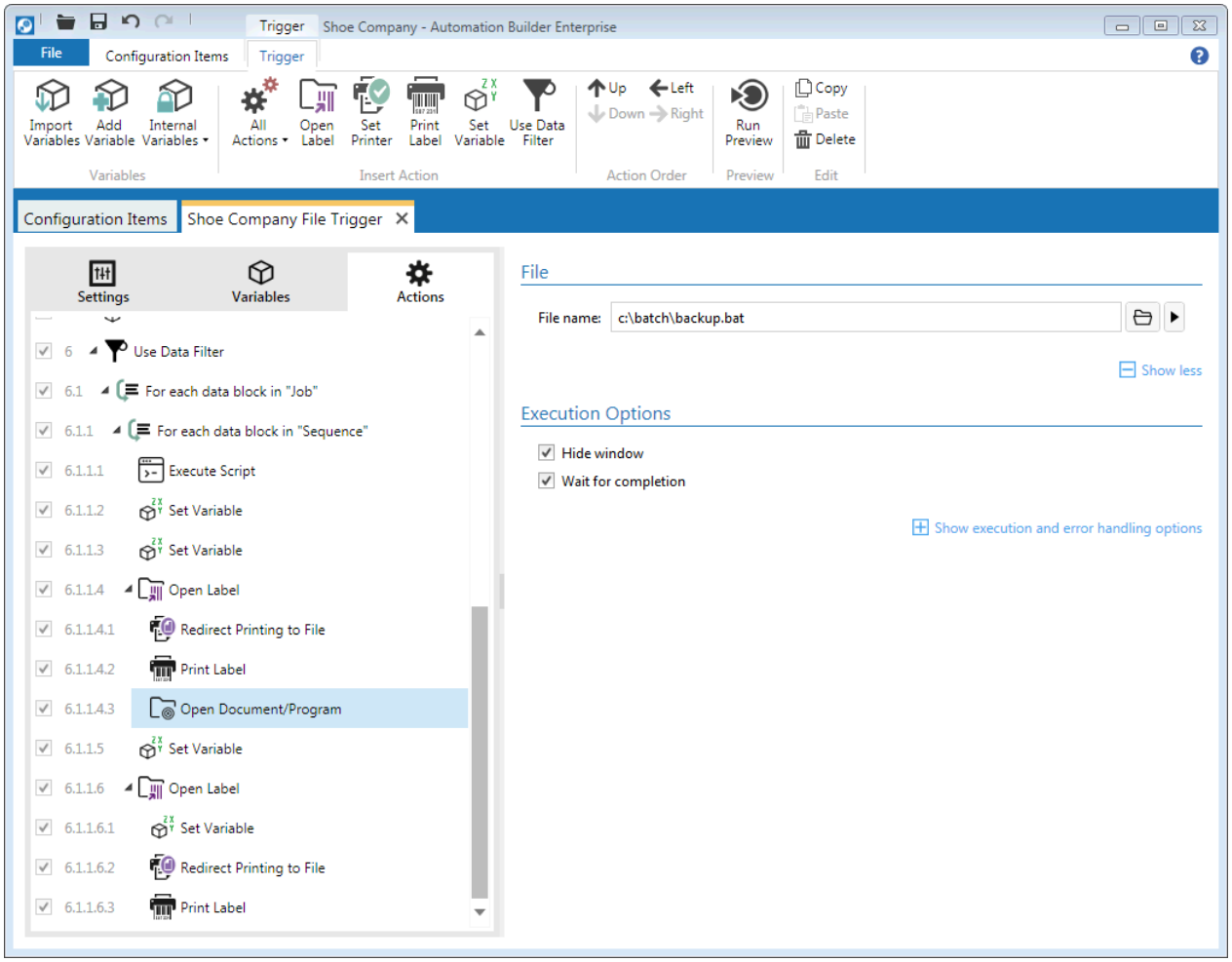


Figure 90 - Automation Builder - Open Document / Program Action

Tip

If the path to the program contains spaces you need to put the path in quotation marks, like this "C:\Program Files\MyApplication\myapplication.exe"

Execute Script Action

Execute Script action allow you to run custom scripts written in VBScript or Python. The language used in the actions depends on what you have selected in the **Scripting** section of the trigger properties (you can find the section in the **Other** category).

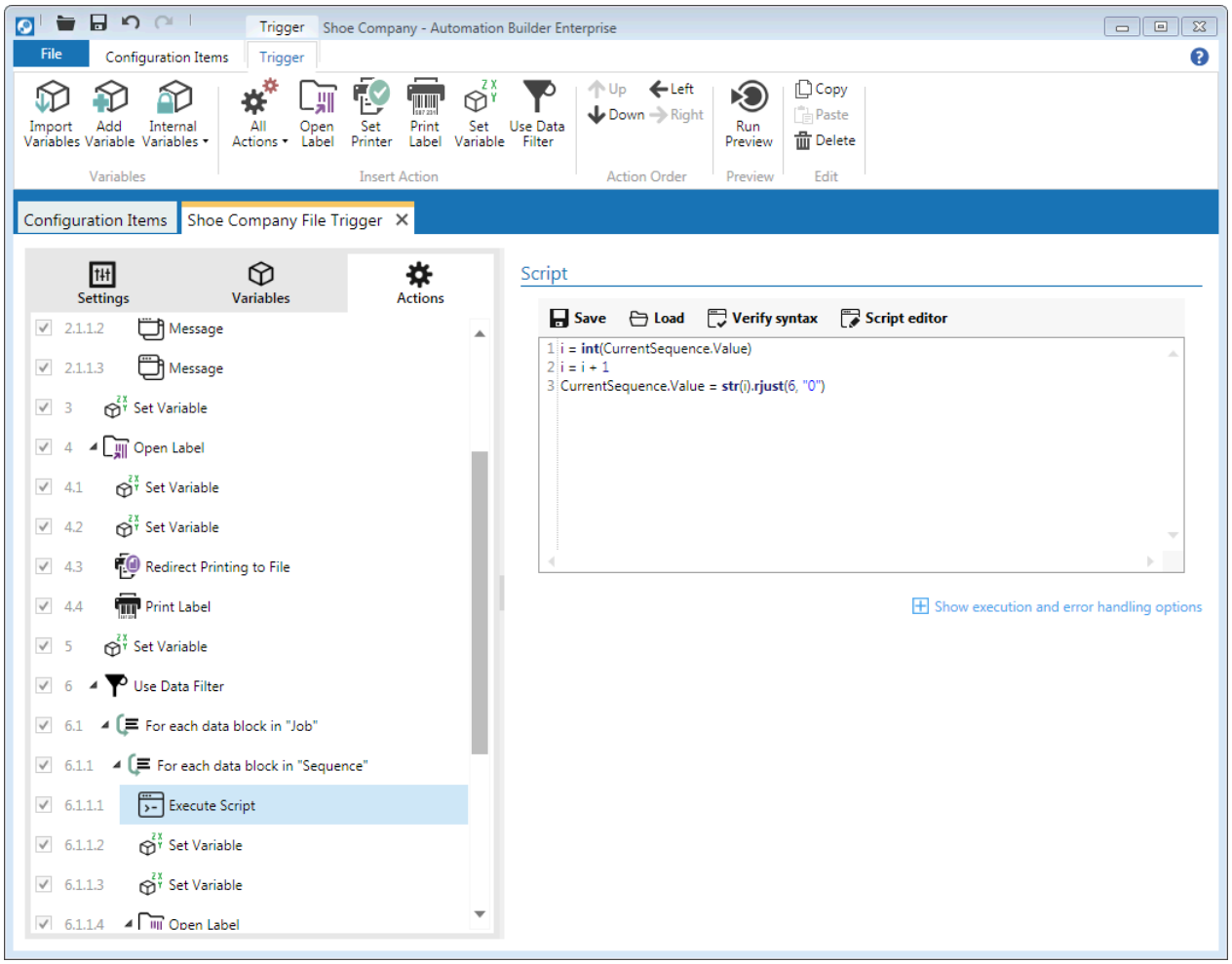


Figure 91 - Automation Builder - Execute Script Action

VBScript is a language developed by Microsoft that is modeled on Visual Basic. It has been installed by default in every release of Microsoft Windows since Windows 98. It was designed as a "lightweight" language with a fast interpreter for use in a wide variety of Microsoft environments. VBScript uses the Component Object Model to access elements of the environment within which it is running; for example, the FileSystemObject (FSO) is used to create, read, update and delete files. (Wikipedia)

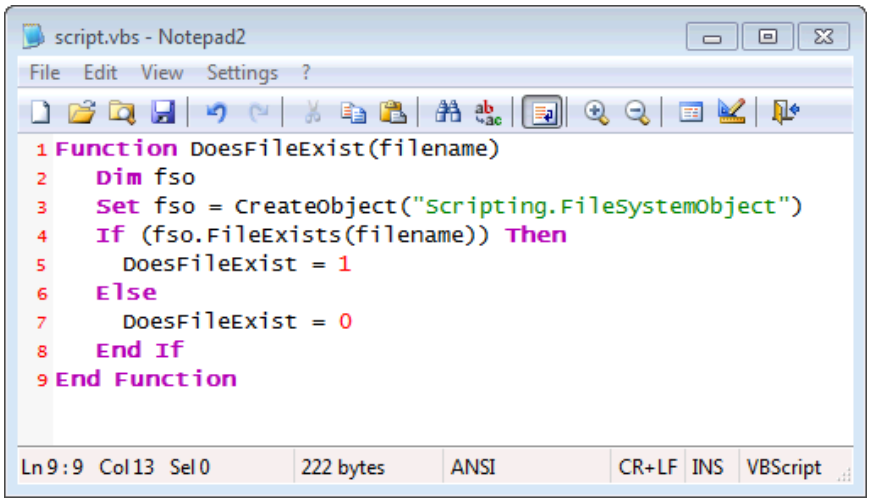


Figure 92 - Automation Builder - Execute Script Action – VBScript script

Python is a general-purpose, high-level programming language. It was conceived in the late 1980s by Guido van Rossum. Today it is available on many operating systems. Large organizations that make use of Python include Google, Yahoo!, CERN, and NASA. There are several implementations of the Python language. NiceLabel Automation uses IronPython, which is an implementation of the Python programming language targeting the .NET Framework. This automatically gives you access to both standard Python libraries and .NET Framework. (Python (programming language))

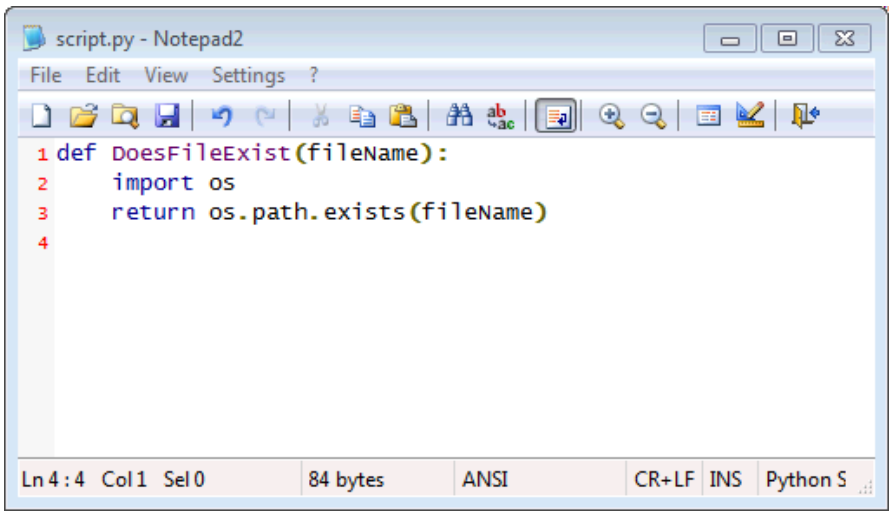


Figure 93 - Automation Builder - Execute Script – Python script

Tip

Python scripts offer better performance in NiceLabel Automation than VBScript. If you have VBScript scripts that need to be executed lots of times, and performance is of significance to you, you might consider writing them in Python instead.

Editing Scripts

Automation Builder offers a simple script editor with syntax highlighting and line numbering. You can load existing scripts from files or save them. To verify that your script does not have any syntax errors, click **Verify syntax** button. Result of verification will be displayed below the script editor.

If you need additional help with your script, you can open more advanced editor by clicking the **Script editor** button. Editor opens in a separate window and it is the same editor that NiceLabel and NiceForm use. It offers an easy way to add basic functions and constants for both scripting languages. Basic instructions for those language elements are available. Additionally you can access more detailed help for the selected language.

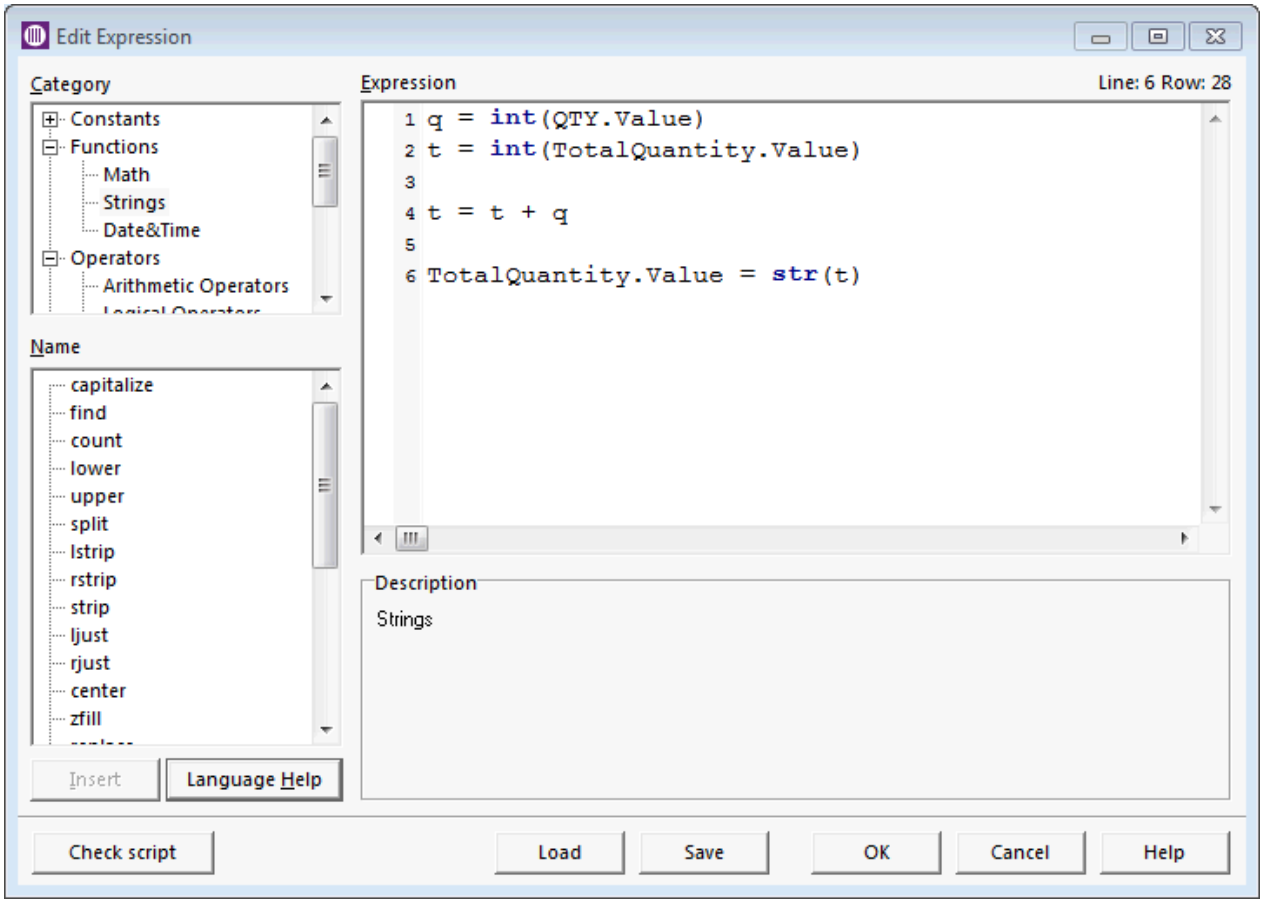


Figure 94 - Script Editor

Predefined Library of Functions

NiceLabel Automation comes with a library of predefined functions for both Python and VBScript. They are stored in two files: vbcustfunctions.vbs and pycustfunctions.py. On Windows 7 these two files are located in the "c:\Users\All Users\EuroPlus\NiceLabel Automation\system" folder.

They contain functions for working with files, Microsoft Access database and quite a few methods that implement different check digit algorithms, like Modulo11 or Modulo23.

Tip

You can add your own functions to the library and they will be available for you to use in NiceLabel Automation. Just do not forget to update the vbcustfunctions.ini and pycustfunctions.ini that contain descriptions of the functions that will appear in the script editor.

A word of warning: before upgrading NiceLabel Automation, make a copy of the files, as they will be overwritten during the upgrade.

Verify License Action

Verify License action allows you to prevent other people from using your configuration files on computers that are not authorized for that. The license key that comes with NiceLabel Automation can contain a unique number (**Solution ID**) that identifies the solution provider that sold the NiceLabel Automation license. If the configured ID matches the solution ID encoded in the license, the target machine is allowed to run actions. If not, then all actions that come after the Verify License action will not be executed.

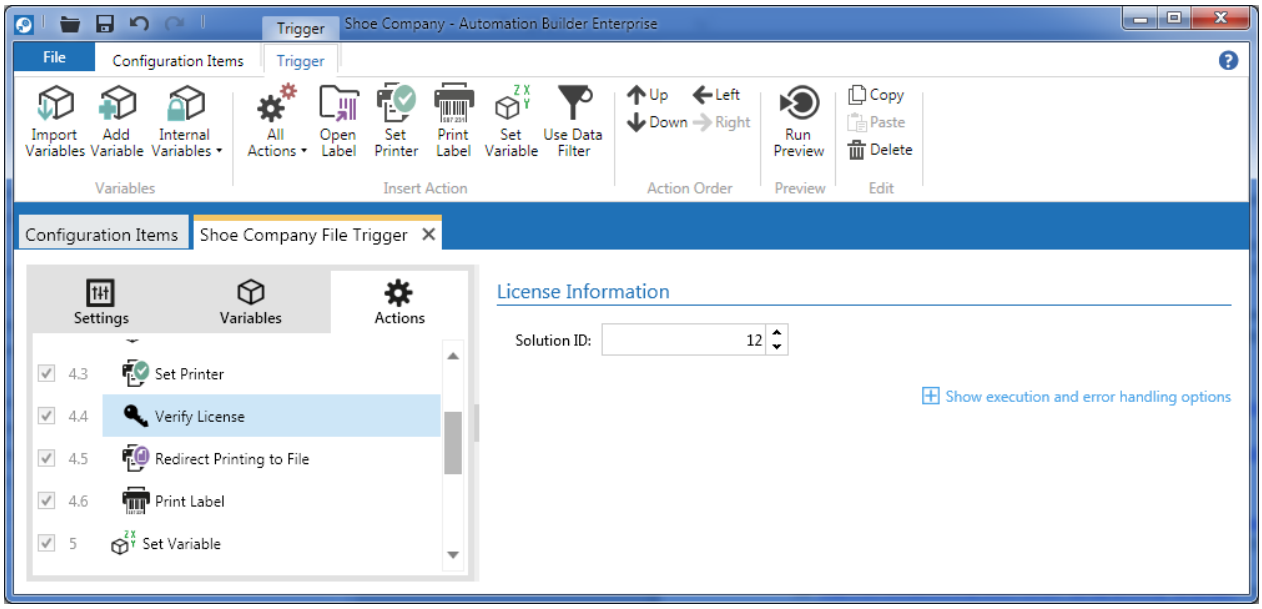


Figure 95 - Automation Builder - Verify License Action

Message Action

To add a custom message to NiceLabel Automation log (visible in the **Log** tab in Automation Manager), you can use the **Message** action. You can define a caption and the message itself. Both will be saved to the log, and can contain predefined or variable values.

Message action can help when you need to troubleshoot configuration problems that you might encounter.

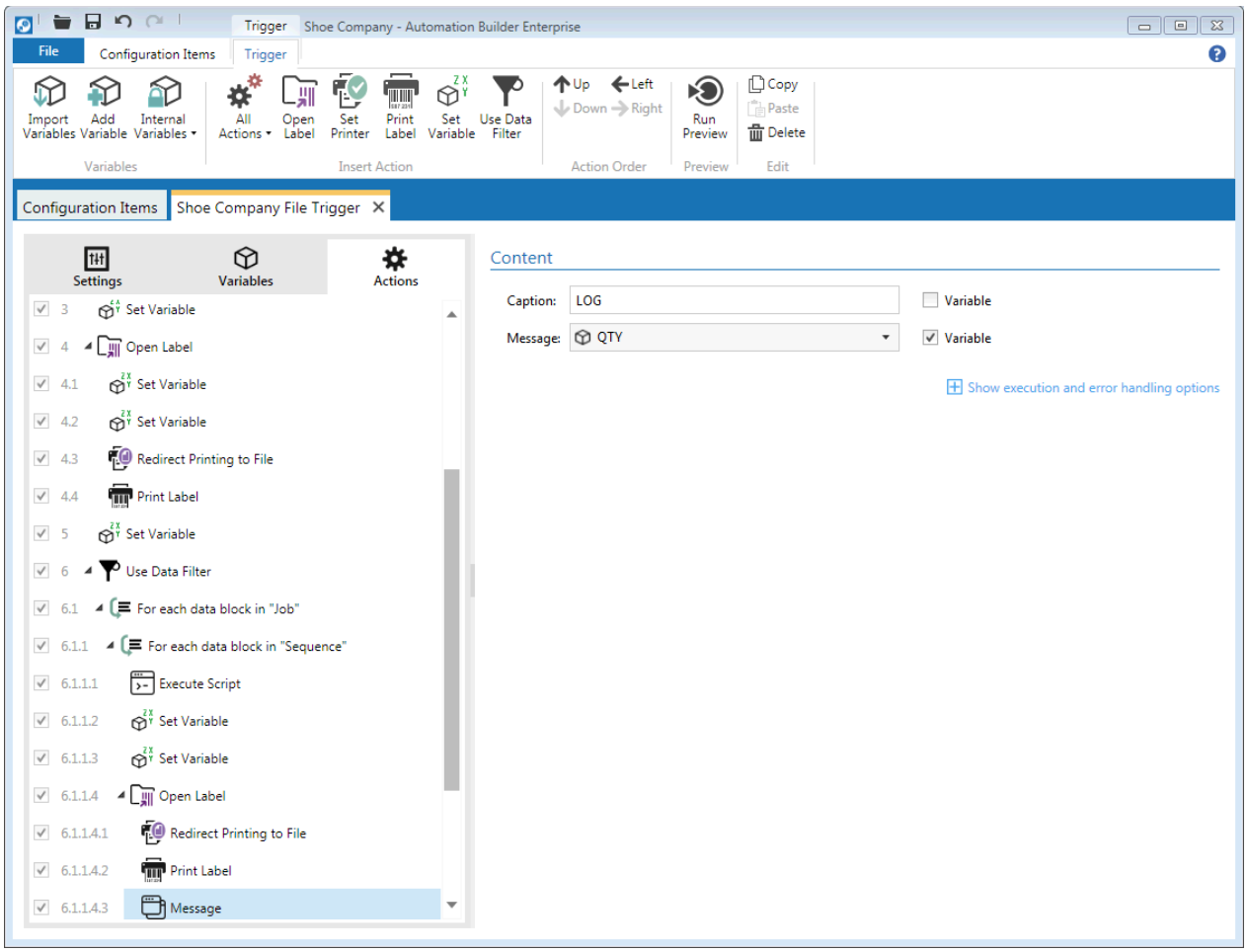


Figure 96 - Automation Builder - Message Action

Try Action

During action execution errors might occur from time to time. The **Try** action is here to help you deal with such situations. It works as follows.

- First, the actions in the **Do** part are executed.
- If no error occurs, the **On Error** part is skipped and execution of the **Try** action is finished.
- If an error occurs during execution of the actions in the **Do** part, the rest of the actions are skipped. Then the actions defined in the **On Error** part are executed, and finally execution continues after the **Try** action.

Any number of actions can be used in **Do** and **On Error** blocks.

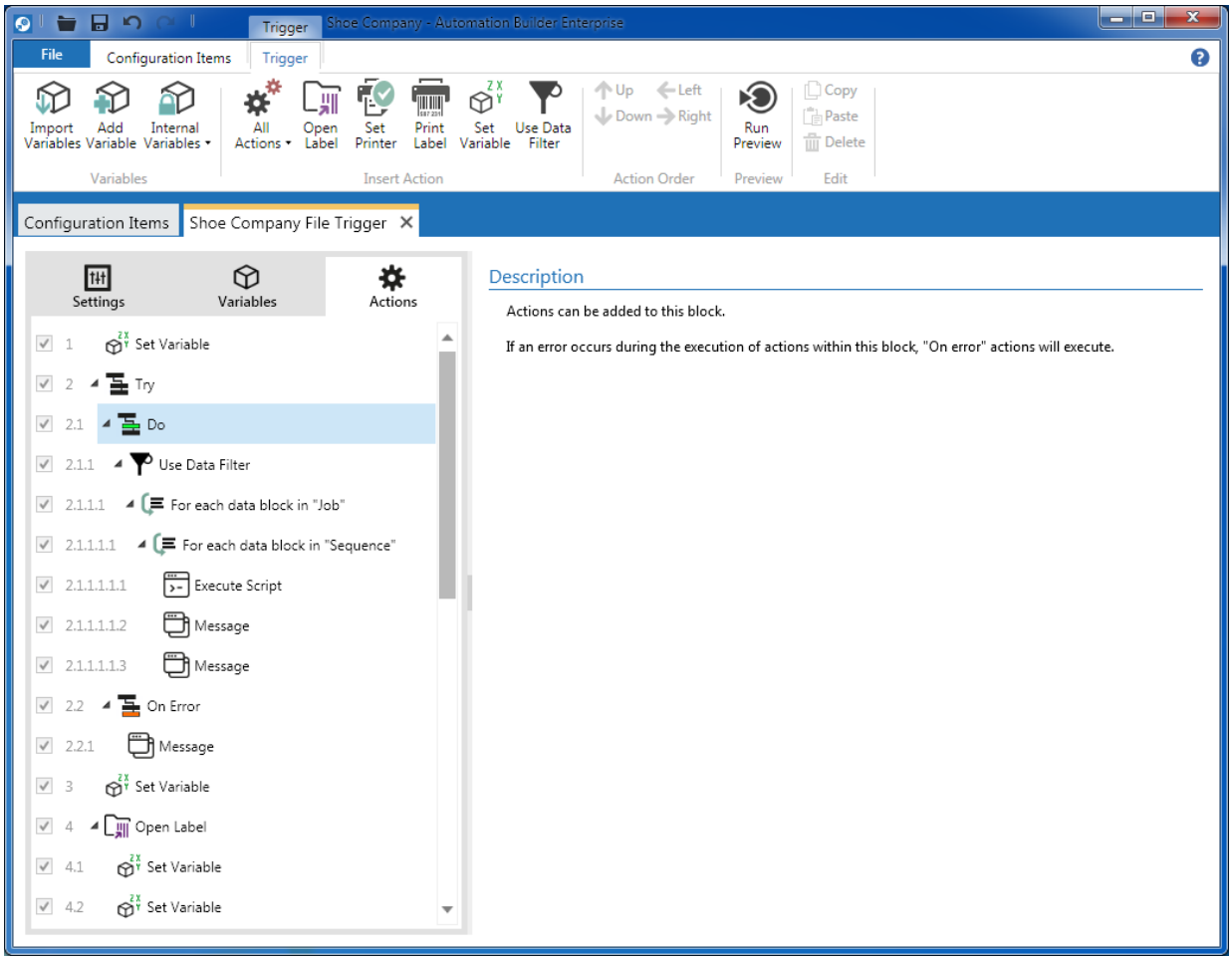


Figure 97 - Automation Builder - Try Action

XML Transform Action

XML Transform action uses the XSLT (Extensible Stylesheet Language Transformations - (Wikipedia)) transformation rules to transform XML documents into other XML documents, or other types of documents such as HTML, plain text etc.

When the action is executed, a new document is created on the contents of the existing one, and stored into selected variable, while the original document is not changed. To save the new document to the file, use **Save Data to File** action.

XML Transform action can get the original document from the data that the trigger received, from a file that is identified with a predefined file name, or a file name that comes from a variable value.

Transformation rules can come from hardcoded file, or from a variable file. You can also enter your own custom XSLT rules.

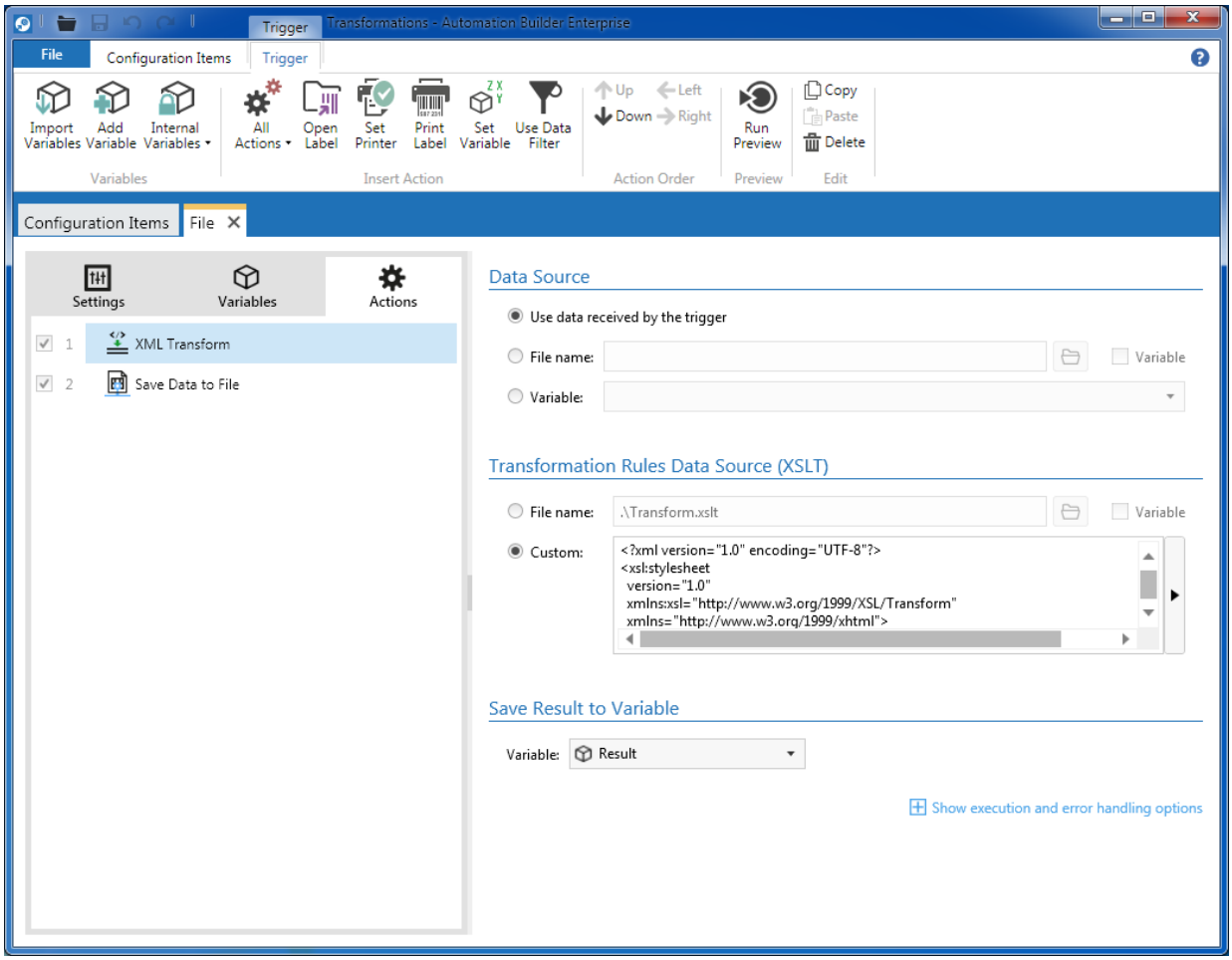


Figure 98 - Automation Builder - XML Transform Action

An example of the original XML document might look like this:

```
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family-name>Smith</family-name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family-name>Ismincius</family-name>
  </person>
</persons>
```

Using the following XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml">
```

```
  <xsl:output method="xml" indent="yes" encoding="UTF-8"/>
  <xsl:template match="/persons">
    <html>
      <head> <title>Testing XML Example</title> </head>
      <body>
        <h1>Persons</h1>
        <ul>
          <xsl:apply-templates select="person">
            <xsl:sort select="family-name" />
          </xsl:apply-templates>
        </ul>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="person">
    <li>
      <xsl:value-of select="family-name"/><xsl:text>, </xsl:text><xsl:value-of
select="name"/>
    </li>
  </xsl:template>
</xsl:stylesheet>
```

will produce the following XHTML document

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head> <title>Testing XML Example</title> </head>
  <body>
    <h1>Persons</h1>
      <ul>
        <li>Ismincius, Morka</li>
        <li>Smith, John</li>
      </ul>
  </body>
</html>
```

NiceLabel Automation and NiceLabel Enterprise Print Manager

Integration with NiceLabel Enterprise Print Manager

NiceLabel Enterprise Print Manager is a web-based solution for document storage and license management that also offers printer and print queue management, overview of print events and even alerts in case of errors. Enterprise Print Manager comes with NiceLabel Control Center Pro and NiceLabel Control Center Enterprise.

Document Storage

Document Storage is the central part of NiceLabel Enterprise Print Manager. It provides a centralized file repository that can be access via browser, from NiceLabel designers and NiceLabel Automation. Both Automation Builder and Automation Manager can work with configuration files stored in the Storage Server. The only condition is that NiceLabel Automation gets its license from NiceLabel Enterprise Print Manager.

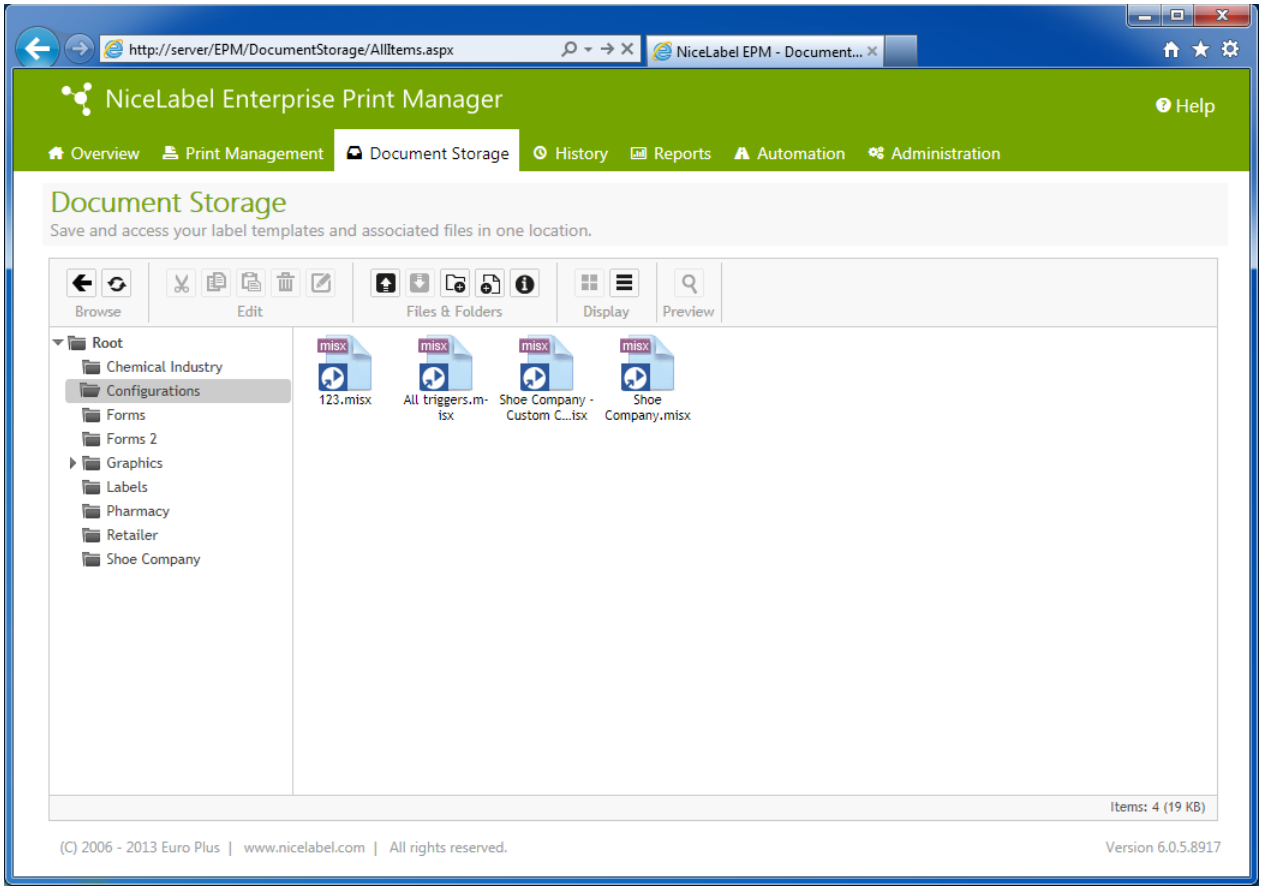


Figure 99 - NiceLabel Enterprise Print Manager - Document Storage

Procedure for opening configuration or label file from Storage Server is the same as opening file from a local drive. Select **Open Configuration** in Automation Builder or **Add Configuration** in Automation Manager to browse for a file. In the left column under Favorites you will see a **Storage Server** icon. Clicking that icon will take you to the files and folders stored in Storage Server. When you find the file you want to use just select it and click **Open** button in the dialog.

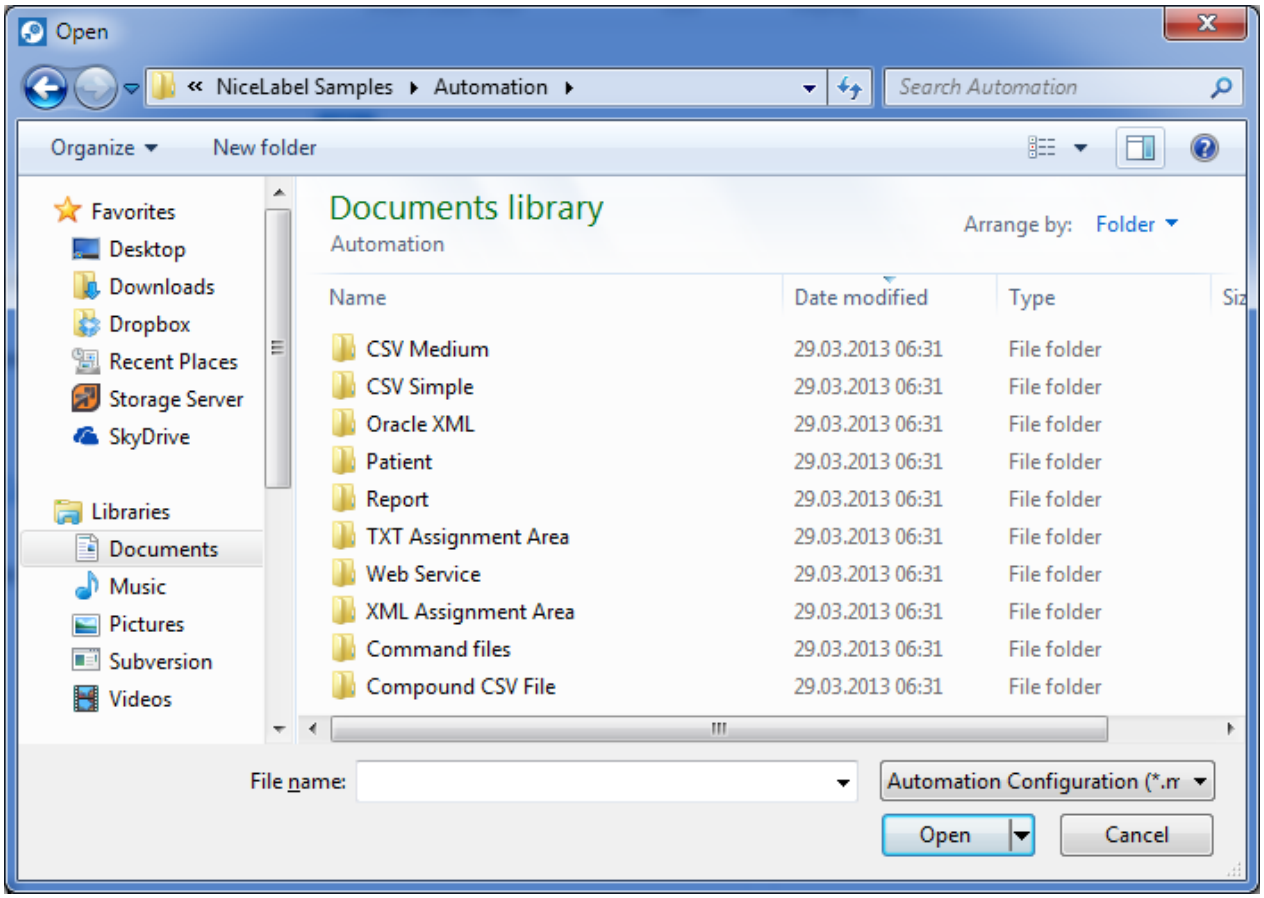


Figure 100 - Opening Files from Storage Server

File name that you would get in Automation Builder when configuring **Open Label** action for example, would look like this

```
http://server:8080/Folder/File.lbl
```

where server is name of the computer where NiceLabel Enterprise Print Manager is installed.

NiceLabel Automation fully supports all the latest Storage Server features like versioning and workflows.

Tip

If you are using NiceLabel Control Center Enterprise with revision control enabled, you can by default access only the latest revision of the label file. To print previous revision of the label file you can use Run Command File action with .job command file or Send Custom Commands action with LABEL command. The following command would open revision 13 of name.lbl file.

LABEL http://server:8080/name.lbl?v=13

Event Monitoring

NiceLabel Automation also allows you to report all print events, status changes and other activity to NiceLabel Enterprise Print Manager. This way you can have a detailed history of printing activities in your company. To enable monitoring of events you need to check the settings in NiceLabel Automation Configuration.

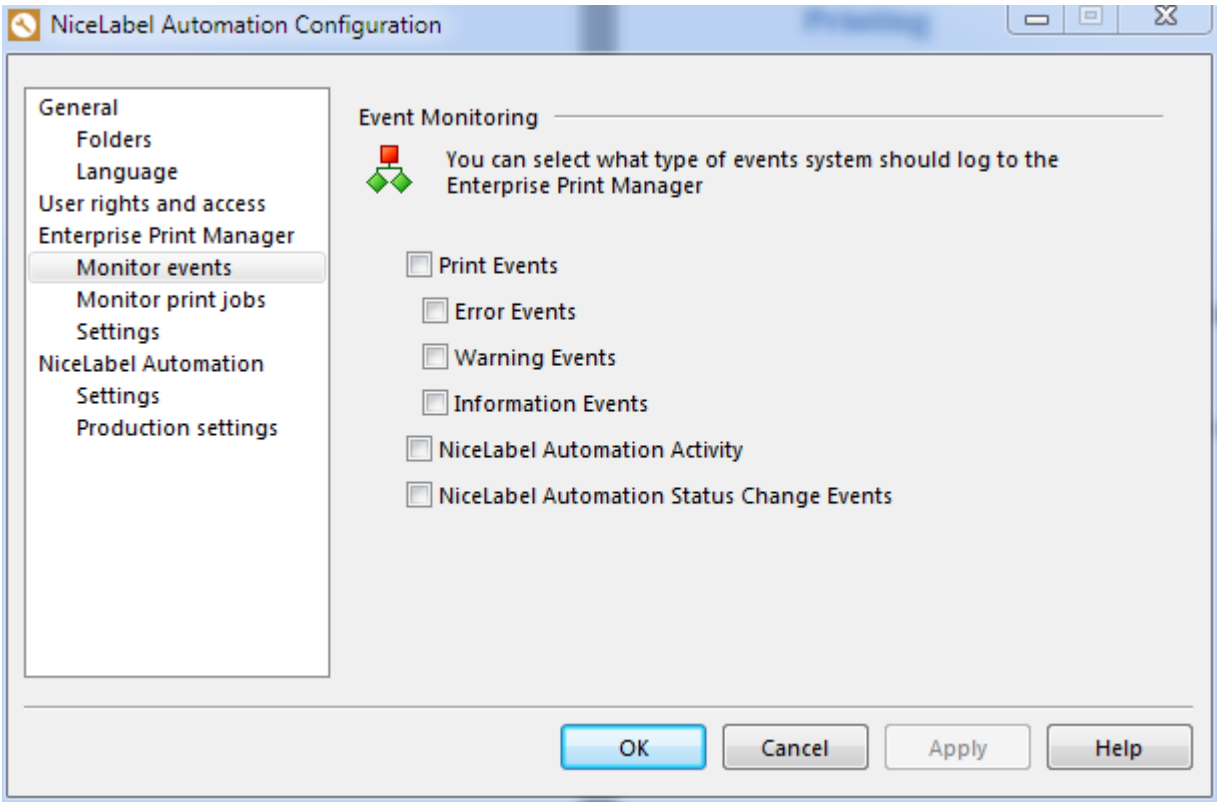


Figure 101 - NiceLabel Automation Configuration - Event Monitoring

Print Job Monitoring and Reprint

Additionally you can also monitor print jobs and report them to NiceLabel Enterprise Print Manager. If this capability is enabled in NiceLabel Automation Configuration, then NiceLabel Enterprise Print Manager can, based on those logs, prepare different reports and also provide reprinting capabilities.

To reprint a print job or parts of it, go to the **History** tab in NiceLabel Enterprise Print Manager and select **Printing** in the **History** pane on the left side of the screen. Then select the print job you would like to reprint, and click **Reprint** button in the **Commands** pane on the left.

You can reprint all label jobs in the selected print job, or only specific label jobs. If you need to, you can also select a different printer.

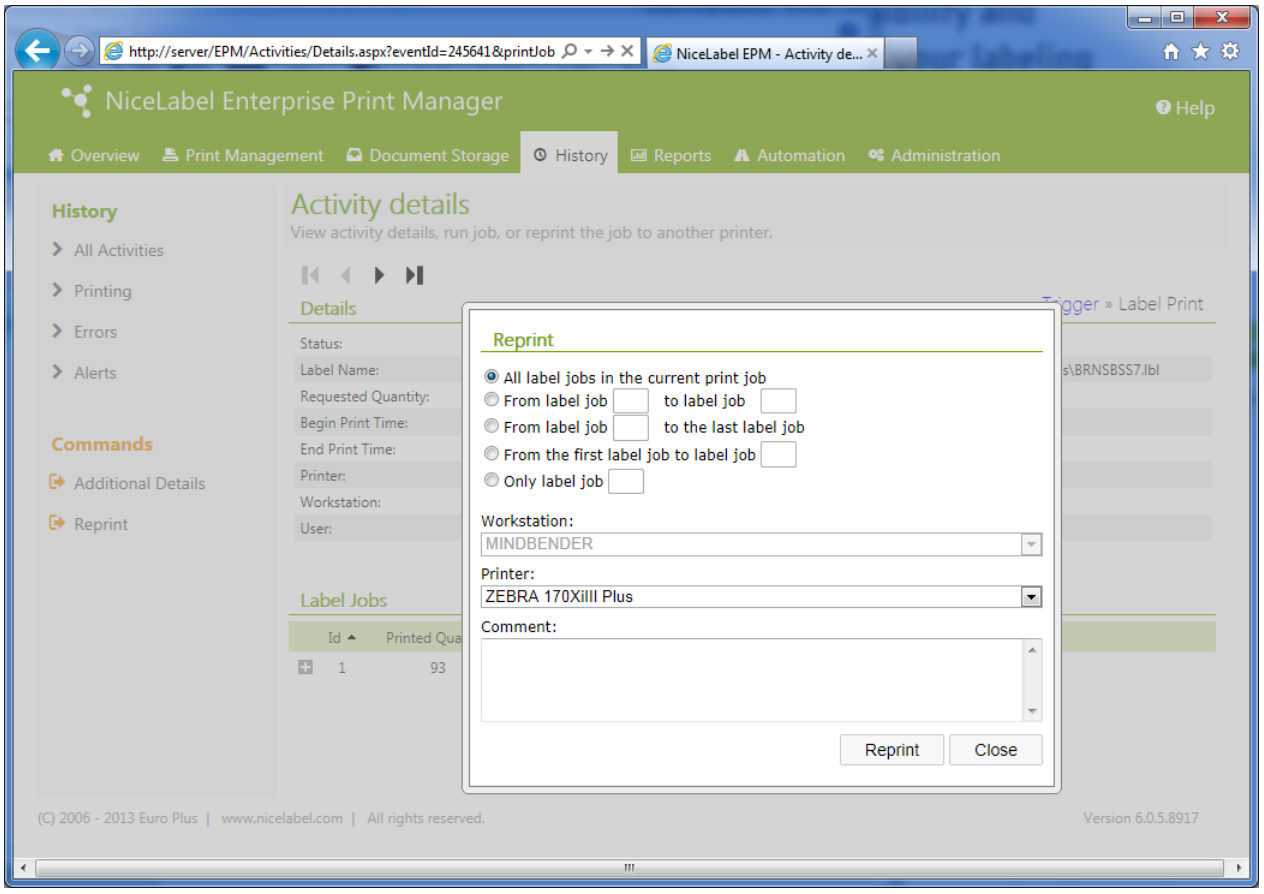


Figure 102 - NiceLabel Enterprise Print Manager – Reprint

Remotely Managing Triggers

When there are NiceLabel Automation servers connected to the NiceLabel Enterprise Print Manager, you can manage their triggers (start and stop them). This is very convenient and the only way to monitor and manage several NiceLabel Automation servers from a single location.

In the screenshot below you can see that currently there is only one NiceLabel Automation connected to NiceLabel Enterprise Print Manager, so you can view and manage only triggers for that particular NiceLabel Automation instance.

For every trigger you can see the current state (started or stopped), configuration name, type of the trigger and its name.

By clicking the check box at the beginning of each row you select the trigger. Then you can click the **Stop** or **Start** button to change state of the trigger.

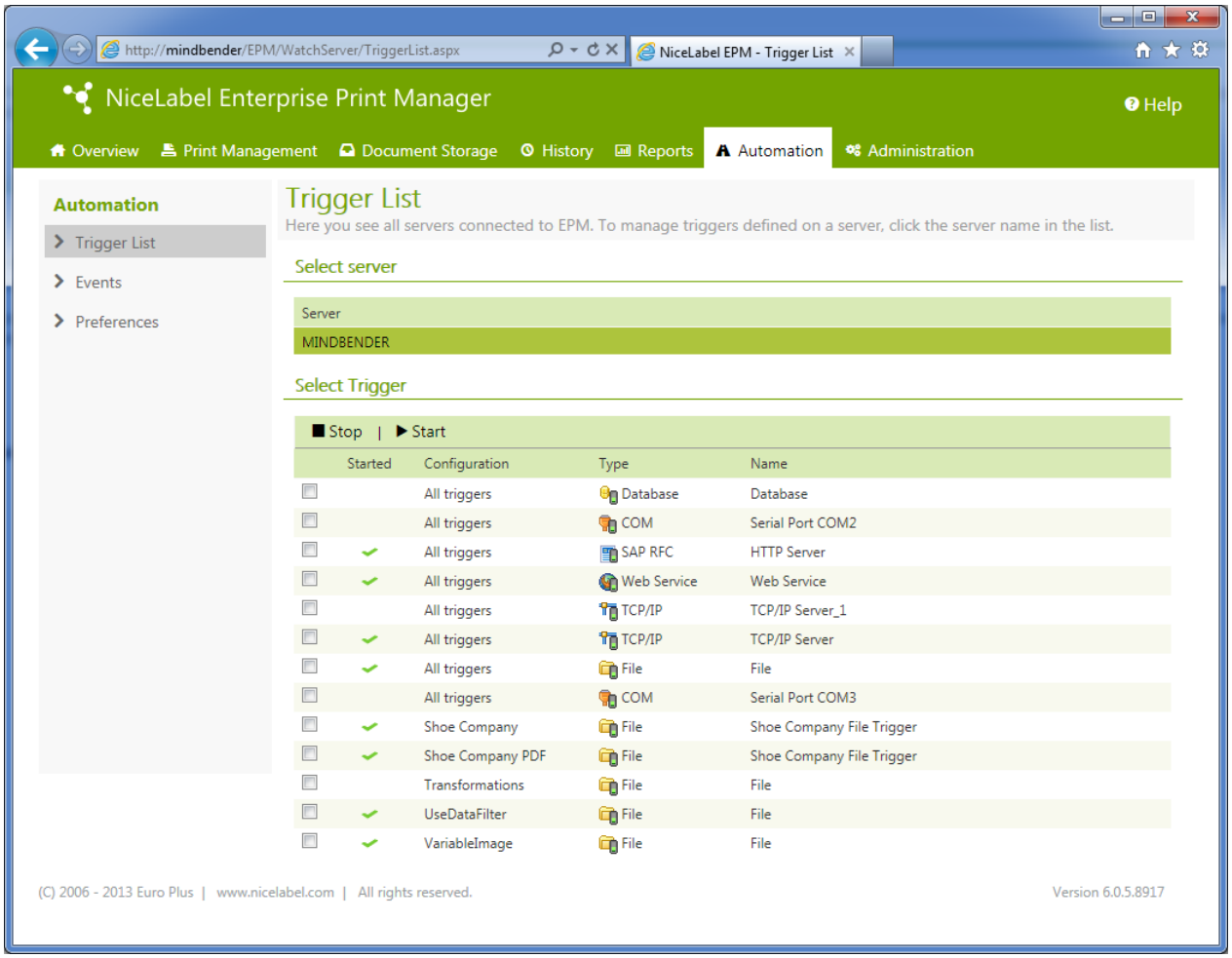


Figure 103 - NiceLabel Enterprise Print Manager - Trigger List

Settings

Usually you would store NiceLabel Automation or NiceLabel settings locally. But these local settings can also be synchronized with other instances of NiceLabel Automation or NiceLabel. This process can be done manually or automatically.

In NiceLabel Automation Configuration use **Store to Server** and **Retrieve from Server** buttons to manually synchronize settings with NiceLabel Enterprise Print Manager. These settings can then be retrieved by another instance of NiceLabel Automation.

To enable automatic synchronization you have to select **Global (Automatic synchronization)** option in the **Settings location** pane.

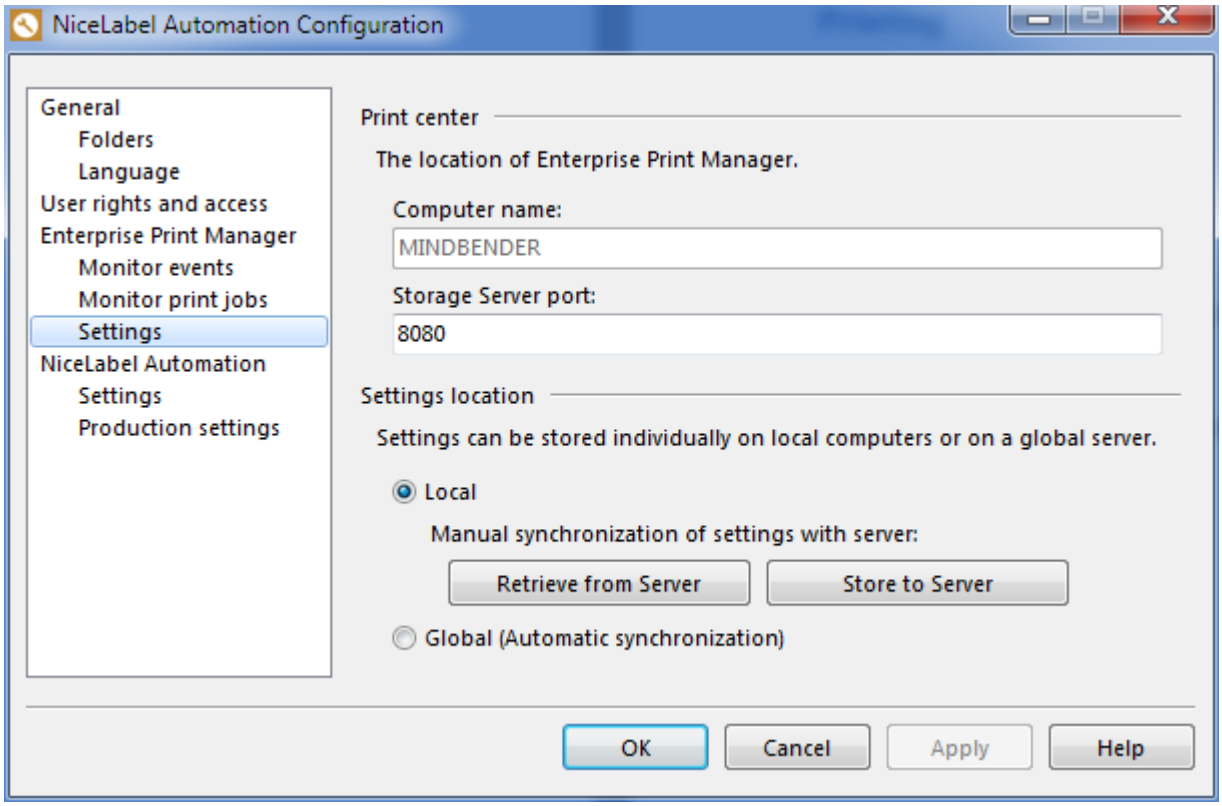


Figure 104 - NiceLabel Automation Configuration - Enterprise Print Manager Settings

Licensing

Licensing

Trial Mode

After installing NiceLabel Automation, it allows you to run it in trial mode. NiceLabel Automation in trial mode has the same functionality, as if you were running a licensed version, so you can fully evaluate the product.

Trial period starts the day you install NiceLabel Automation and it lasts 30 days. During this time, Automation Manager will display a notification message indicating the number of remaining days until the trial expires. After the trial period expires, NiceLabel Automation will no longer fire triggers.

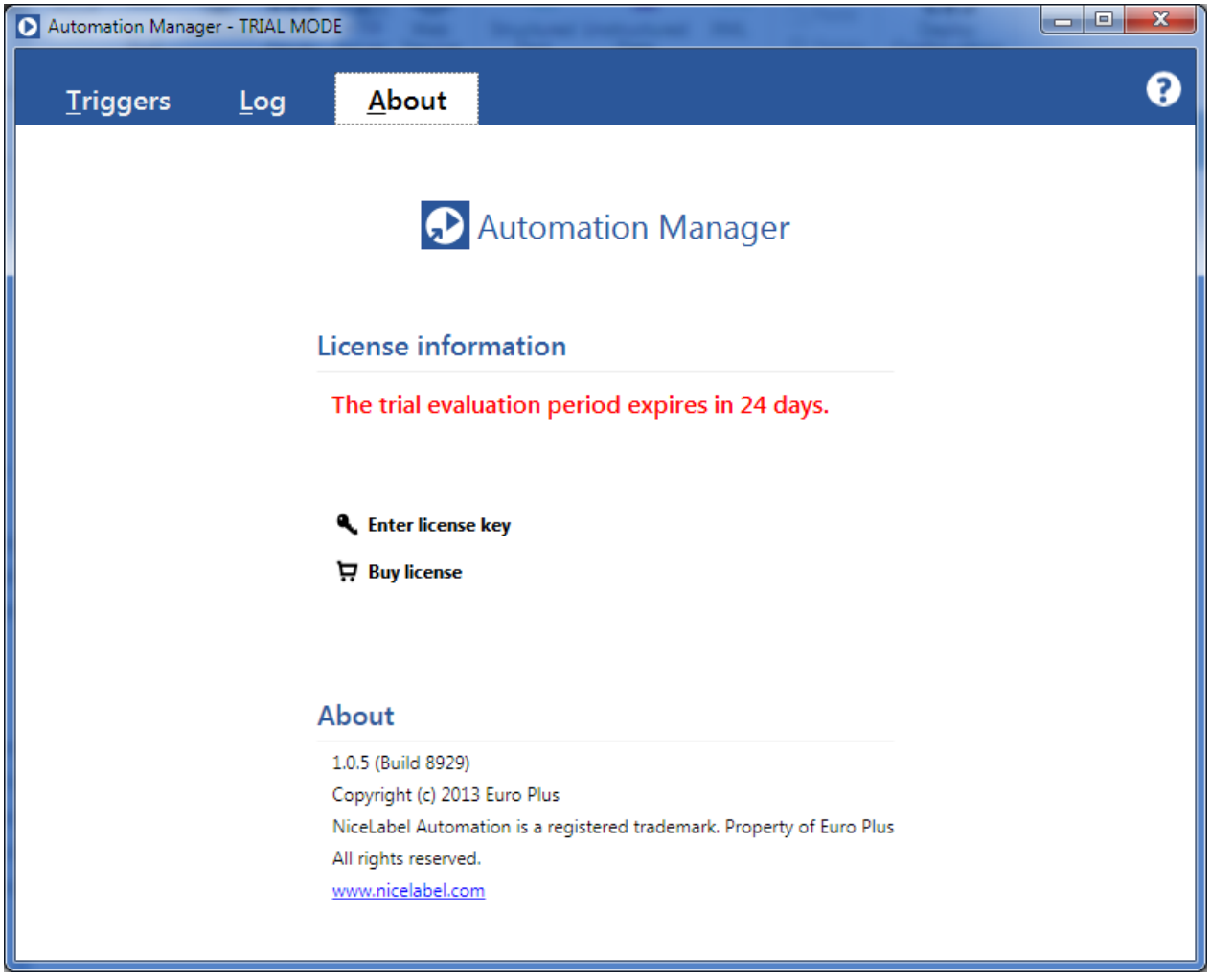


Figure 105 - Automation Manager - Trial Mode

If you need to extend the trial period beyond 30 days, you can contact your NiceLabel reseller and request another trial license key.

Activation

To fully use NiceLabel Automation, you have to activate It first. You can activate NiceLabel Automation from both Automation Manager and Automation Builder. In both cases a wizard will guide you through the activation process.

There are two possible activation methods:

- **Single User Software Key**
- **Enterprise Print Manager License Server**

Single User Software Key

When you select **Single User Software Key** as activation method for NiceLabel Automation you have to enter information about yourself and a key number that you received when you purchased the product.

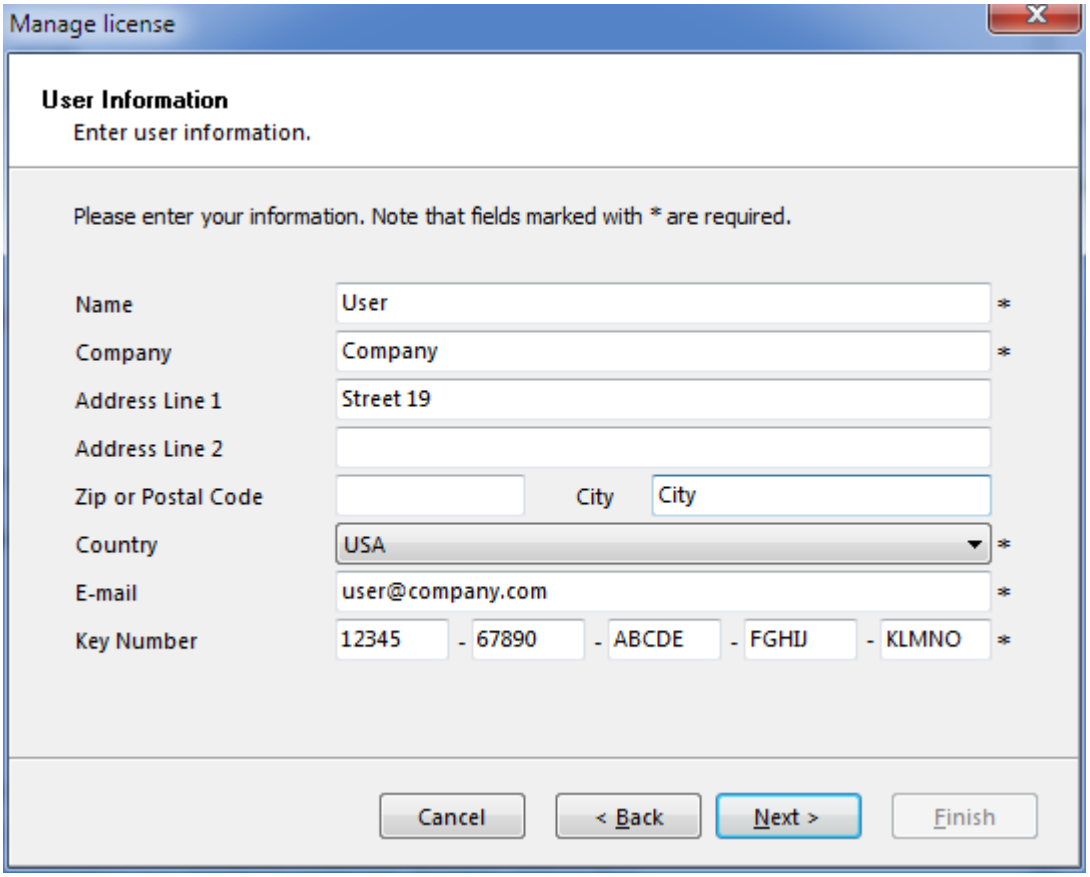


Figure 106 - Activation - User Information

After entering all the required data and clicking **Next** button, you will move to the next step in the wizard. **Registration Number** is automatically generated, while you have to provide the **Activation Code**. Activation Code can be acquired automatically, by clicking **Connect to the Activation server** link (if you have Internet access), or you can enter it manually. To obtain the Activation Code manually you have to

go to the activation web page. Once on the activation web page, enter the key number and registration number, and the activation code will be generated for you.

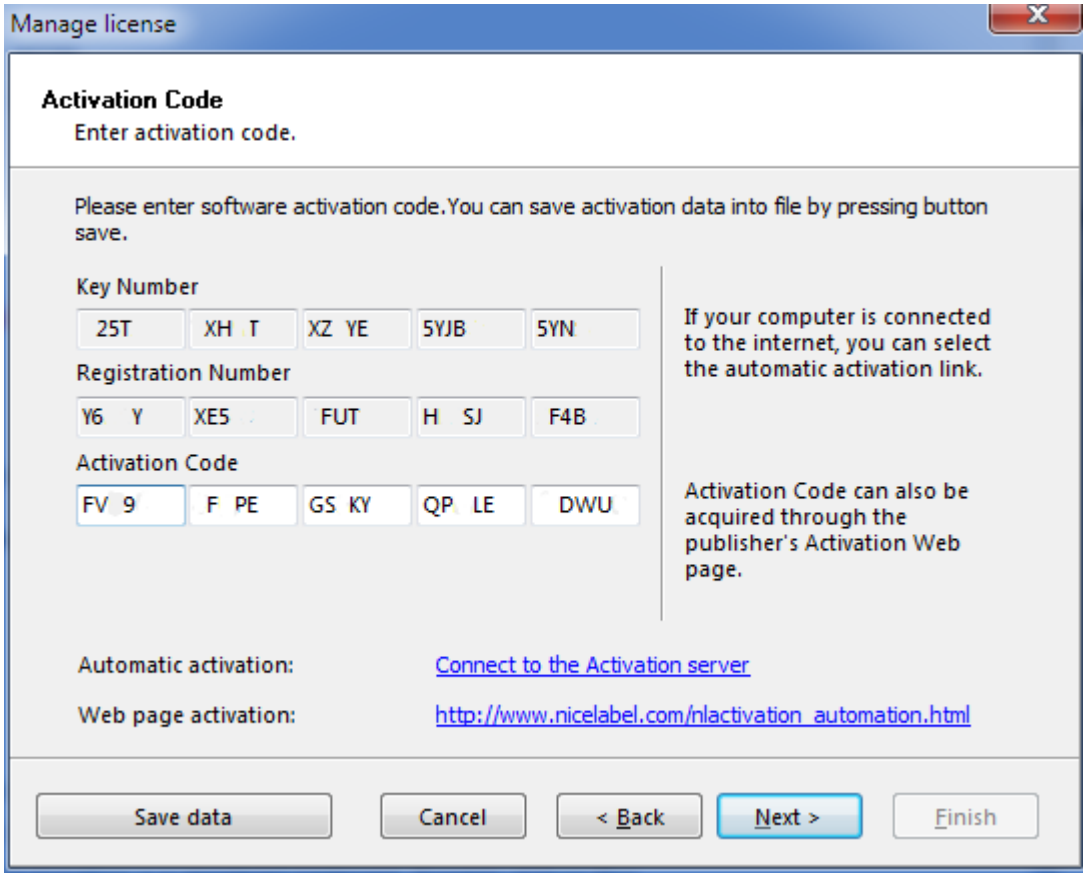


Figure 107 - Activation - Entering Activation Code

When NiceLabel Automation is successfully activated you get the confirmation message.

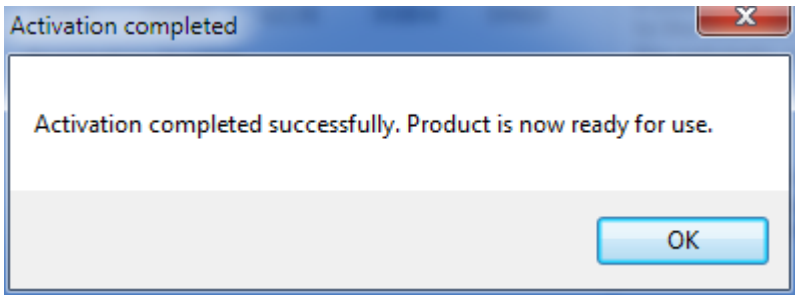


Figure 108 - Activation Completed

Enterprise Print Manager License Server

If you already have NiceLabel Enterprise Print Manager with NiceLabel Automation license, select **Enterprise Print Manager License Server**. Activation in this case is even simpler. All you need to do is enter the name of NiceLabel Enterprise Print Manager server and click **Next**.

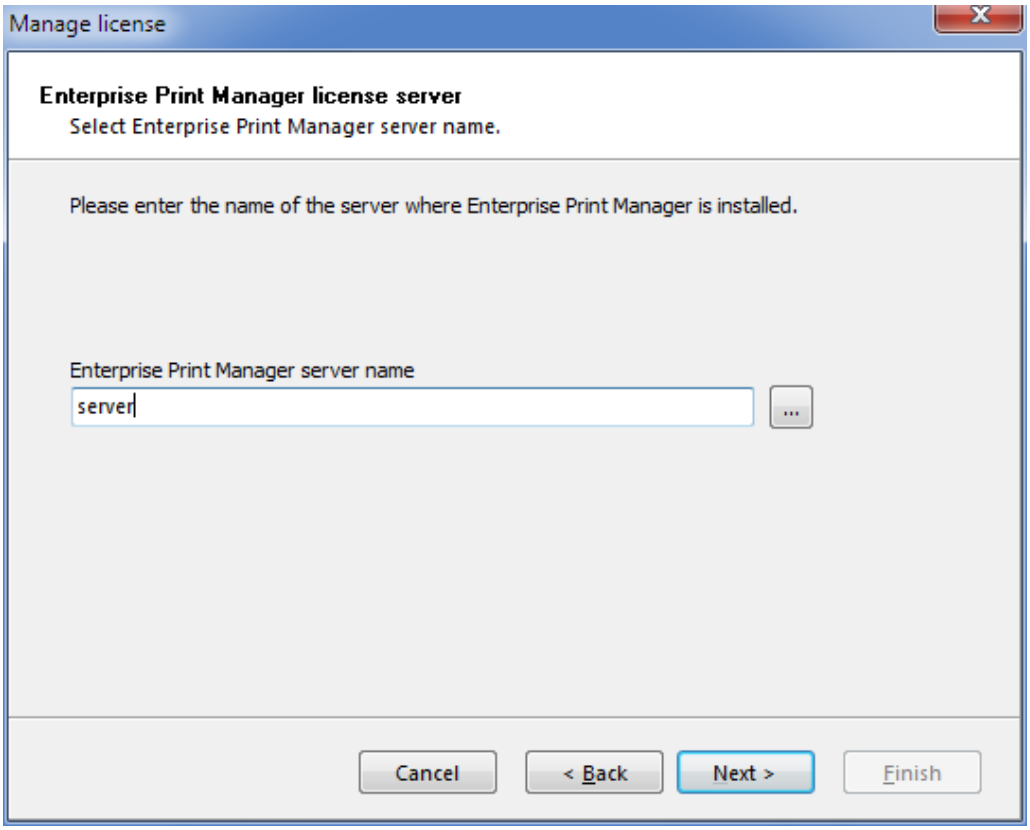


Figure 109 - Activation - Enterprise Print Manager License Server

When activation procedure is completed, some of NiceLabel Automation settings will change. Print jobs and printing events will be logged to the server. For possible consequences of these changes, see the section about performance tips and tricks.

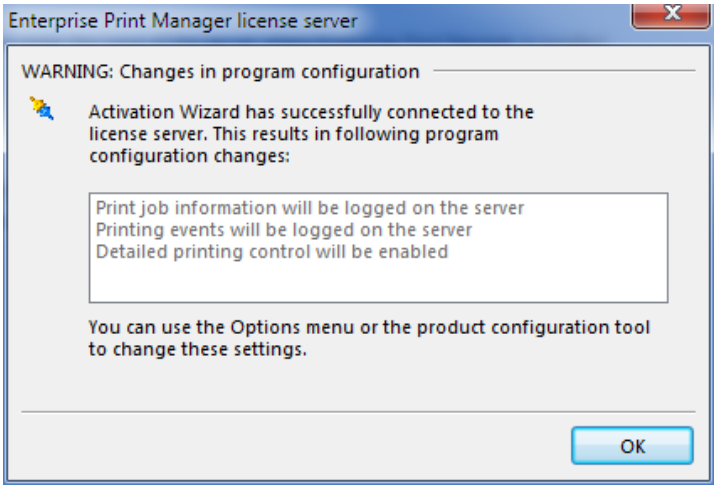


Figure 110 - Enterprise Print Manager - Settings Changes

Offline Mode

If the NiceLabel Enterprise Print Manager server that acts as the licensing server is not accessible, and NiceLabel Automation cannot verify the license, it will fall into offline mode. You have 24 hours to restore the connection to the licensing server, otherwise NiceLabel Automation will stop executing triggers.

Automation Manager will notify you about such problem with a message in the information pane, as seen below.

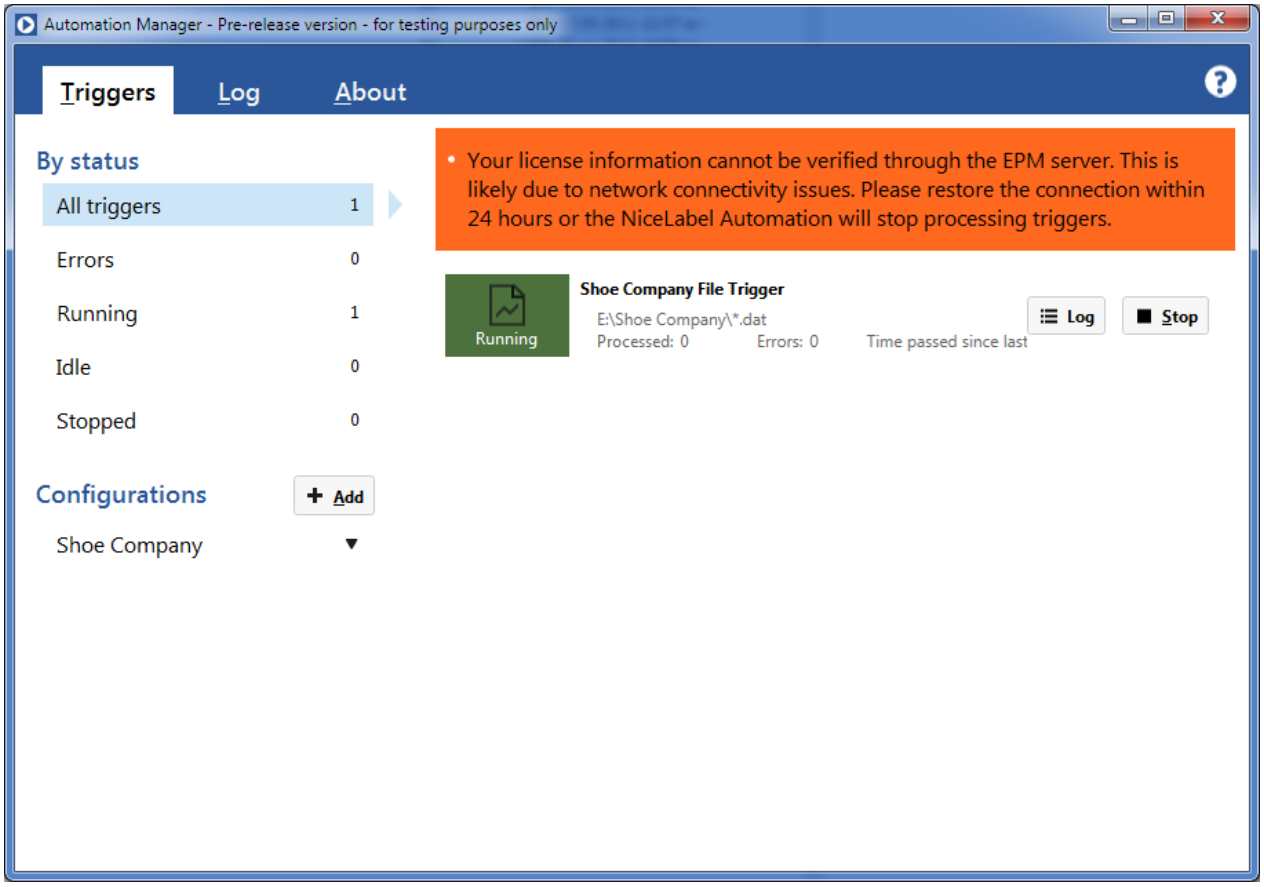


Figure 111 - Automation Manager - Offline Mode

Differences between different editions of NiceLabel Automation

There are three editions of NiceLabel Automation:

- Automation Easy
- Automation Pro
- Automation Easy

The three editions differ in a number of available triggers, filters and actions. Additionally Automation Easy can only be activated using **Single User Software Key**, while only Automation Enterprise has

support for clusters. For more details about available triggers, filters and actions, see chapters about those topics.

Automation Builder and Different Editions of NiceLabel Automation

Automation Builder is not affected by the license level that NiceLabel Automation is activated with. Even if you have bought NiceLabel Automation Easy license, you can still design configuration files that will work with NiceLabel Automation Pro or NiceLabel Automation Enterprise. You just might not be able to run them in Automation Manager.

To set product level, Automation Builder should work with, go to the **File** menu, click **Tools** and select **Product Level**. In the menu you can then select the produce level you want to define configuration for. As already mentioned, not all triggers, filters and actions are available in every edition of NiceLabel Automation.

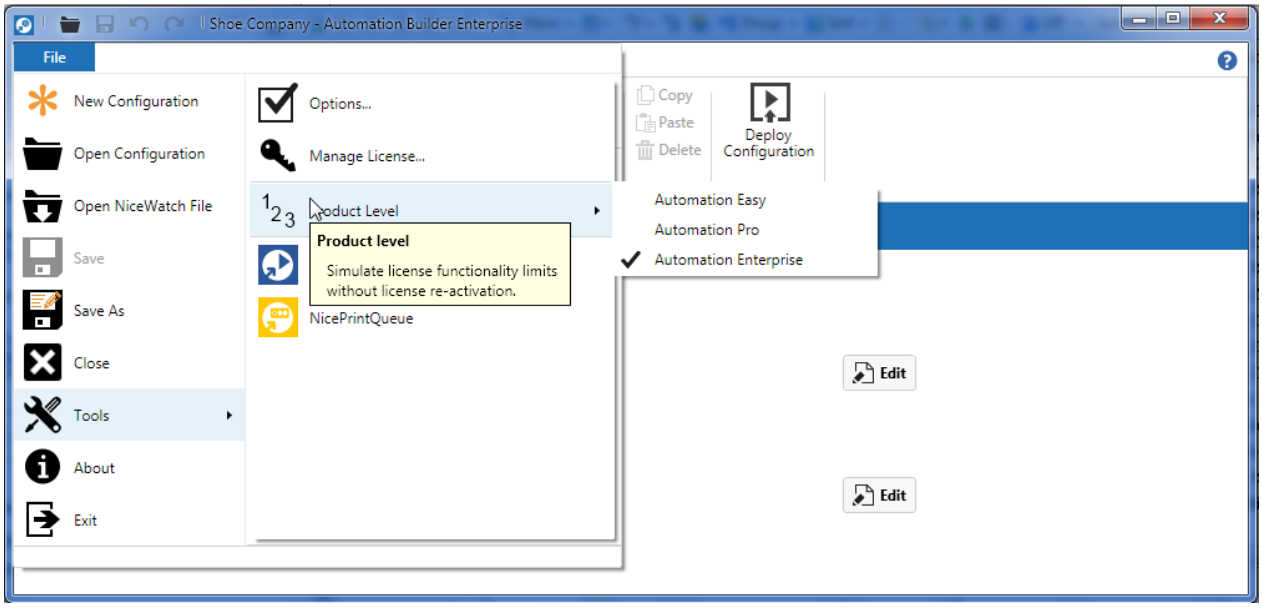


Figure 112 - Automation Builder - Product Level

Clusters

Failover Cluster

Definition

A failover cluster is a group of independent computers that work together to increase the availability and scalability of clustered roles (formerly called clustered applications and services). The clustered servers (called nodes) are connected by physical cables and by software. If one or more of the cluster nodes fail, other nodes begin to provide service (a process known as failover). In addition, the clustered roles are proactively monitored to verify that they are working properly. If they are not working, they are restarted or moved to another node. With the Failover Clustering feature, users experience a minimum of disruptions in service.

(Microsoft Corp.)

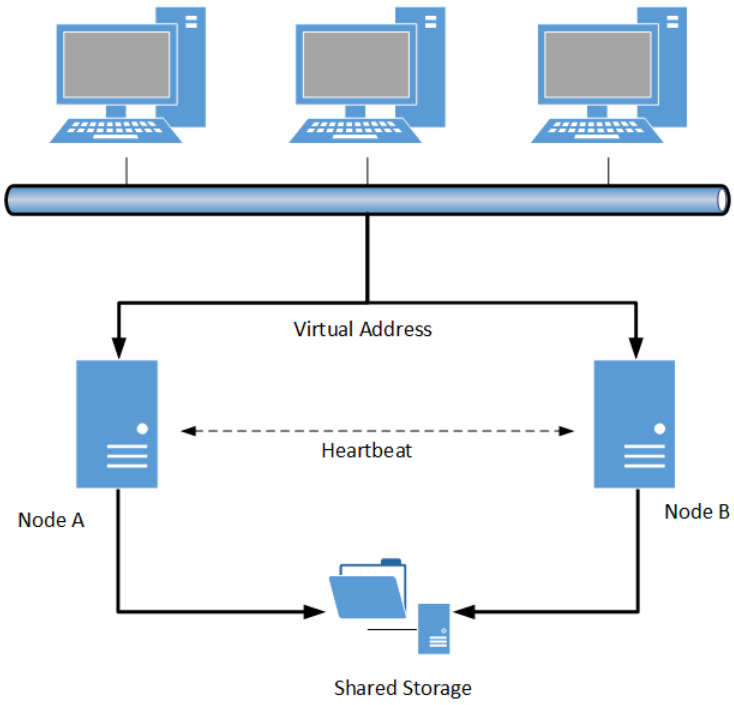


Figure 113 - Failover Cluster

The Enterprise edition of NiceLabel Automation supports failover (or high-availability) clusters. To set up NiceLabel Automation in such environment, there are a few steps you need to take.

1. Install and activate NiceLabel Automation on each clustered server (node).

2. Set up Failover Clustering feature in Windows Server. Note that clustering is not available in all editions of Windows Server.
3. Enable failover cluster support in NiceLabel Automation Configuration and define a folder where files, needed by NiceLabel Automation to support failover cluster, are copied to. This folder should be on a server that all cluster nodes can access and NiceLabel Automation has full access to it. You can find the settings under the **Cluster Support** category.
4. Set NiceLabel Automation Proxy Service and NiceLabel Automation Service startup type to manual.
5. Set the start parameter for NiceLabel Automation Service to configuration file that will be run when the service starts. Configuration files should be stored in a location that is accessible by all nodes.
6. Configure the cluster to start NiceLabel Automation on the backup node, when the main server becomes unresponsive.

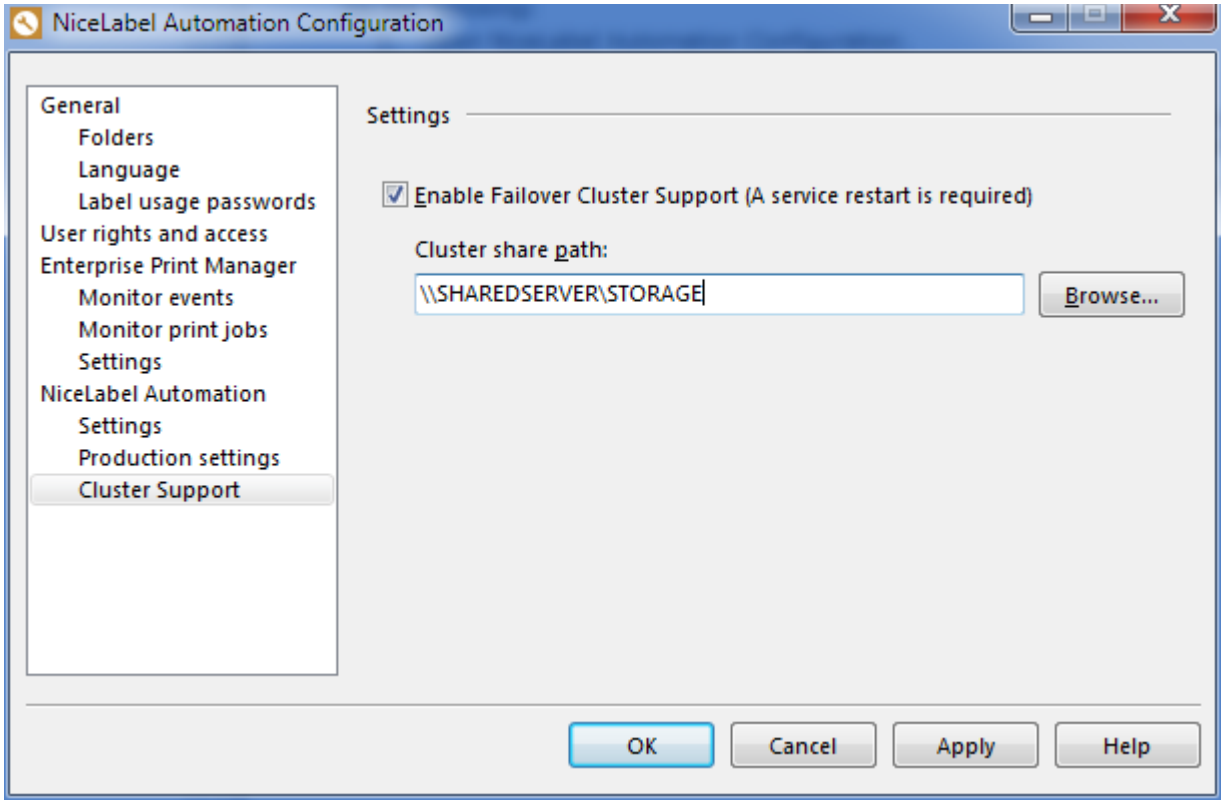


Figure 114 - NiceLabel Automation Configuration - Failover Cluster Support

Load-Balanced Cluster

Definition

Load balancing is a computer networking method to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to achieve optimal resource utilization, maximize throughput, minimize response time, and avoid overload. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy.

(Load balancing (computing))

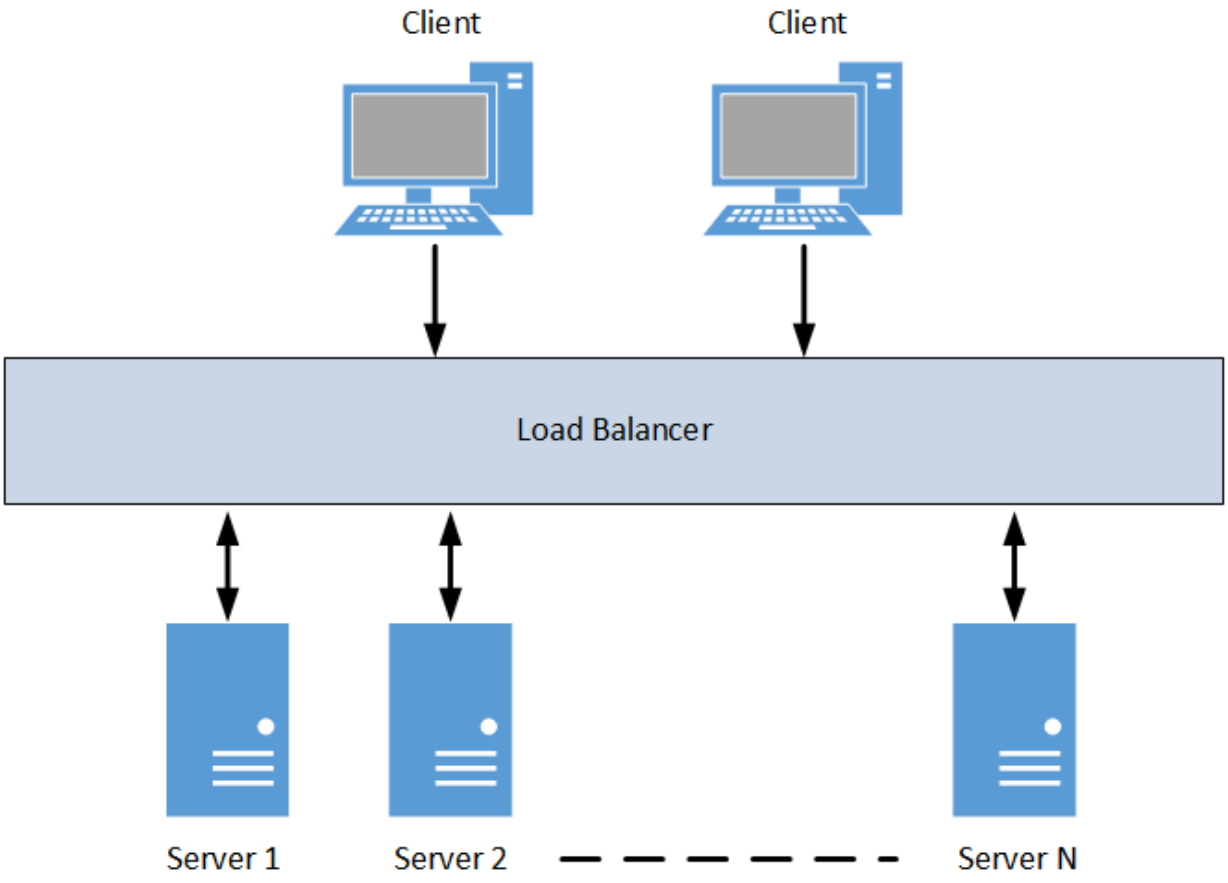


Figure 115 - Load-Balanced Cluster

Network Load Balancing clusters provide scalability and high availability for TCP-based services and applications by combining several servers into a single cluster. By using Network Load Balancing feature

of Windows Server to build a group of cloned, or identical, clustered computers, you can enhance the availability of these servers.

When the load balancer receives a request from the client, one of the servers in the group processes the request. Every server is capable of handling the request independently. If any server is unavailable due to error or maintenance, other servers can still serve requests without being affected. (Microsoft Corp.)

In NiceLabel Automation, this means all TCP-based triggers can take advantage of load balancing. This includes **TCP/IP Server Trigger**, **HTTP Server Trigger** and **Web Service Trigger**.

There are no special settings in NiceLabel Automation to support load-balanced cluster. When you have all the hardware and cluster configuration in place, you need to install NiceLabel Automation on all servers in the cluster and then load the same configuration file in all instances.

Differences with NiceWatch

What Has Changed in Comparison with NiceWatch?

NiceWatch is an automated printing application available in NiceLabel Suite and NiceWatch Enterprise. It can be seen as predecessor to NiceLabel Automation. NiceLabel Automation is completely new product so there are several differences between the two of them.

- In NiceWatch you could configure triggers and manage them all in one application. NiceLabel Automation now has two separate modules: Automation Builder for configuring, and Automation Manager for managing.
- In NiceWatch you had a possibility to run the application as either Windows service or as standalone application. NiceLabel Automation is always running as service.
- NiceWatch used NiceEngine as its printing engine. NiceEngine is a separate module that is accessed via technology called OLE Automation. In NiceLabel Automation the print engine is embedded in the application and there are no outside modules being loaded. This brings massive performance and resource benefits.
- E-mail trigger is not supported in NiceLabel Automation.
- HL7 support available in NiceWatch Enterprise is not available in NiceLabel Automation.
- **Send XML to SAP** and **Close Label** actions are not available in NiceLabel Automation.
- When configuring triggers in NiceWatch you could enable error event that would cause trigger to save incoming data to a backup file. In NiceLabel Automation you can use **Try** action instead. **Try** action will however not be automatically generated when you import NiceWatch configuration file in Automation Builder.
- When you open NiceWatch configuration that has triggers with filters in Automation Builder, you will notice that trigger and filter are now two separate entities connected together by **Use Data Filter** action. This allows for better flexibility, as you can define filter only once and then use it several different triggers.
- **SAP AII** trigger from NiceWatch Enterprise is now reconfigured as **HTTP** trigger in NiceLabel Automation.
- **Save incoming data** setting of the File trigger in NiceWatch is now a separate **Save Data to File** action.
- **Printer Status Report** action is reconfigured when loaded in Automation Builder.

There are also several differences that are related to label designs. Some of the label design features that were available with NiceWatch, are not available in NiceLabel Automation 1.0:

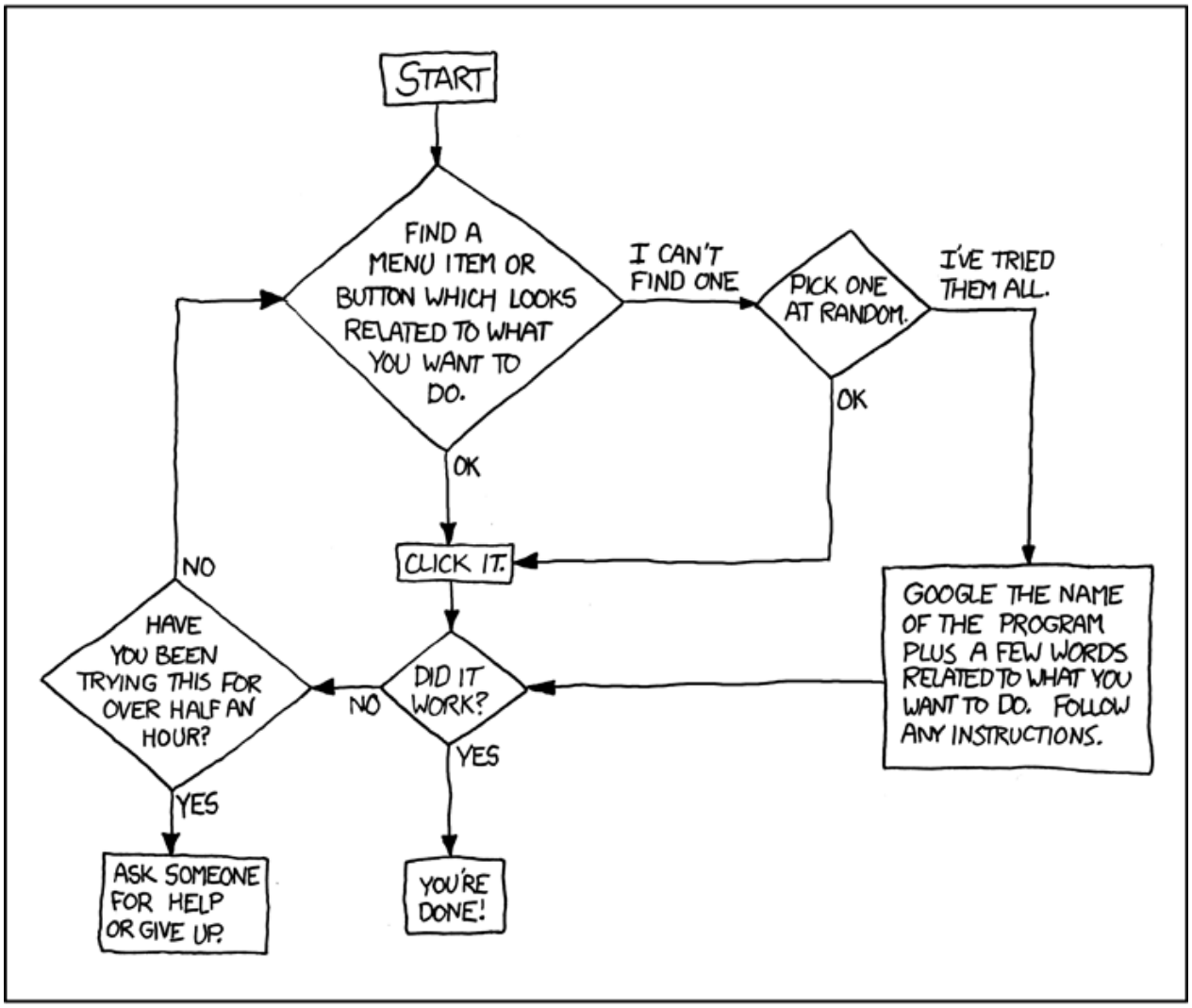
- Barcodes (Royal Mail, Australia Post 4-state, USPS Intelligent mail, Codabar, Code128 Pharmacy, Code-32, Code-93, Kix, MSI, Plessey, Anker, Pharmacode, all Postnet barcodes, 2D Pharmacode, Codablock F, GridMatrix)
- Functions (User defined check-digit algorithms, Currency exchange, Lookup table, PIATS Code)
- Graphical formats (.tga, .pxm, .jp2, .j2k, .pcx, .psd)
- Clip art images that come with NiceLabel
- RFID

- Object formatting (**Best fit** option in RTF object, **Variable height** in RTF object, **Mirror** for Text and Text Box objects, **Character spacing** for Text and Text Box objects, **Line spacing** for Text Box object, **Stacked text** for Text object and Internal printer fonts in Text Box object)
- Label formatting (multiband printing, header / tail labels, bath printing, RTL printing, smart replacement of printer fonts, store / recall printing mode).
- Rounded label corners are not visible in label preview in Automation Builder.
- Label and its properties cannot be manipulated using scripting language.

Troubleshooting

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS, AND OTHER "NOT COMPUTER PEOPLE."

WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:



PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN. CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

Figure 116 - Technical Support Cheat Sheet – by Randal Munroe, http://xkcd.com/627

Things to be Aware of

Labels Need to be Saved with the Latest Version of NiceLabel

Label files (.lbl) that you want to use with NiceLabel Automation need to be saved with either NiceLabel 5.4.4 or later, otherwise NiceLabel Automation might not work correctly.

Changing Label Designs while NiceLabel Automation Service is Running

If you change the label design that is already used by one of the running triggers you need to reload configuration because changes to label designs are not automatically detected. There are three ways to do that:

- Reload configuration in Automation Manager,
- Change and save the configuration file in the Automation Builder, or
- Restart the NiceLabel Automation Service

Changing Default Folders in Automation Configuration

If you change locations of default folders in Automation Configuration, you need to restart the NiceLabel Automation Service, because those changes are not automatically picked up.

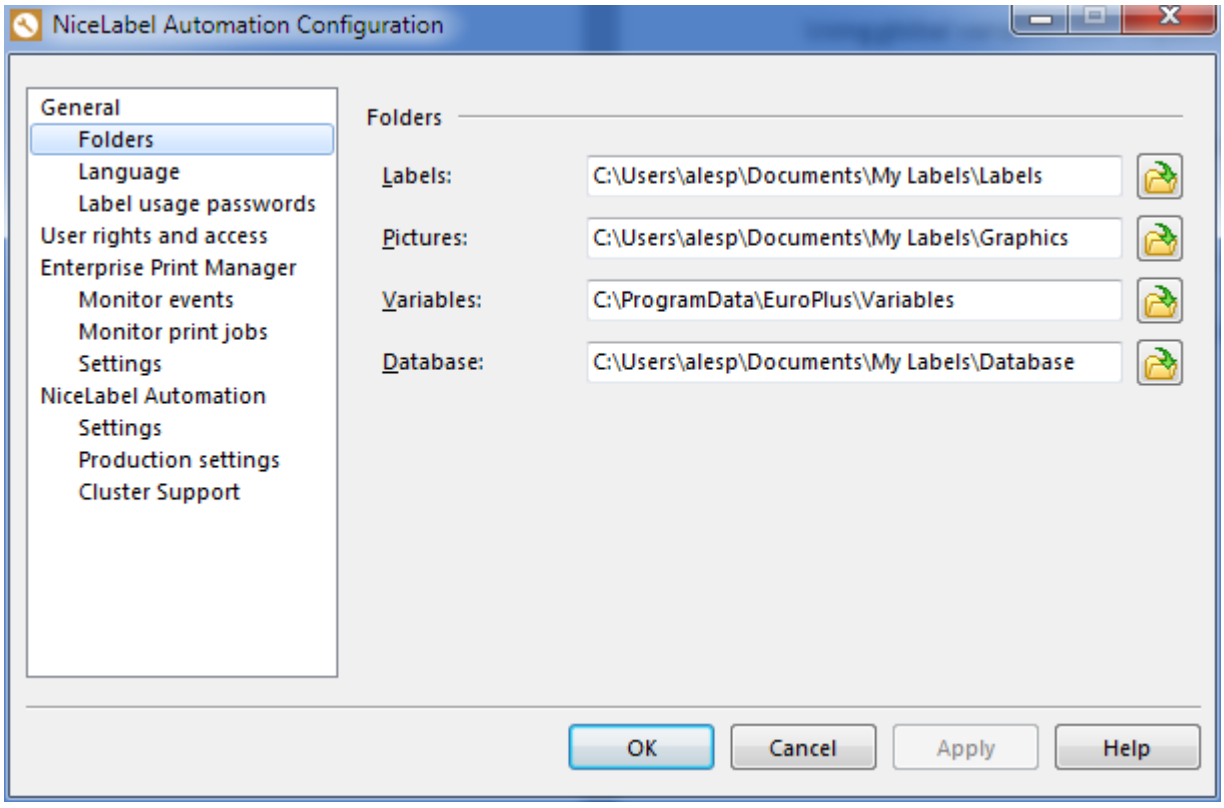


Figure 117 - Automation Configuration - Folders

Using the Latest Version of NiceLabel Drivers

Several stability issues (this issues were manifested only when heavily printing on several printers at the same time using the same printer driver) were fixed in the latest version of NiceLabel Drivers, so it is suggested that you download the latest printer drivers and use them. For more information check the NiceLabel web page at www.nicelabel.com.

Performance Tips and Tricks

Caching Remote Files

Configuration files, labels, and graphics can be stored locally, or remotely in NiceLabel Enterprise Print Manager Document Storage, or on a network share. When NiceLabel Automation tries to use remote files, it has to transfer them via the network and - depending on how fast your network infrastructure is - that might take some time.

In order to improve performance in such scenarios, NiceLabel Automation offers you caching of remote files. When caching of remote files is enabled, NiceLabel Automation will store those remote files locally and refresh them periodically. Label designs, graphics and file-based databases can be cached.

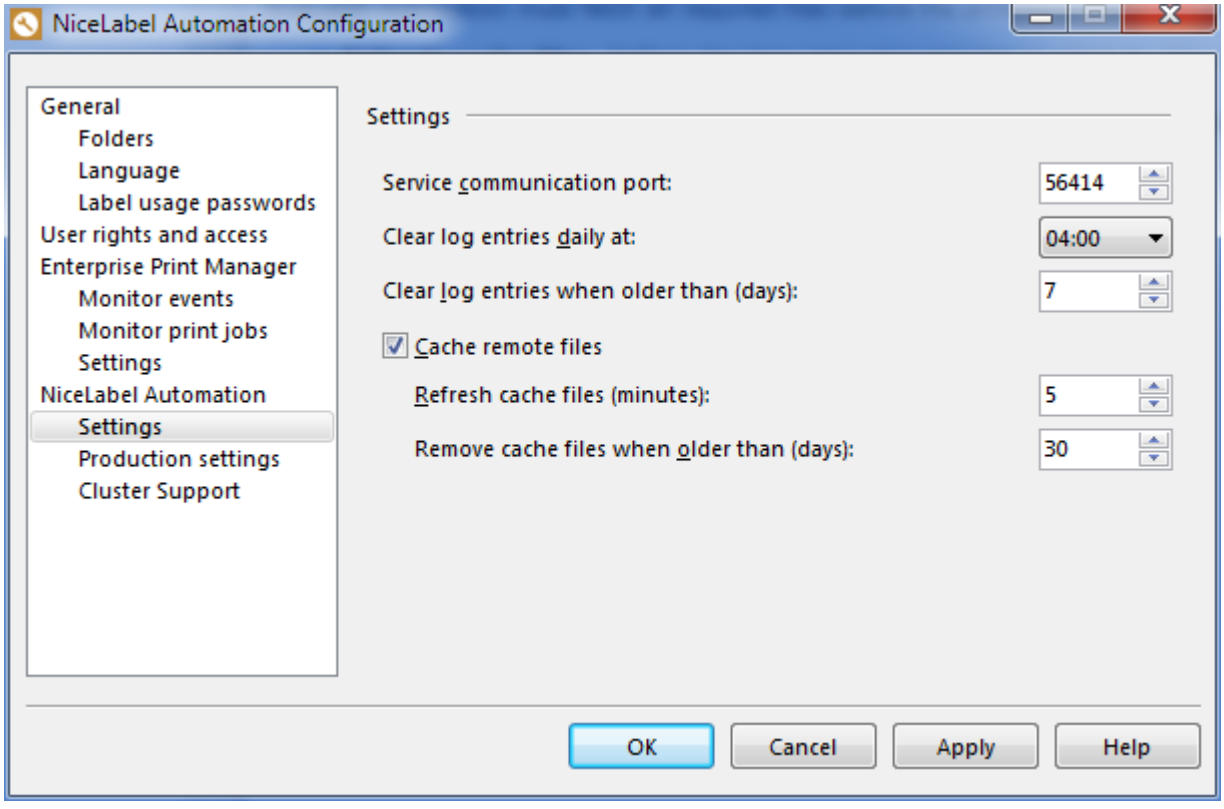


Figure 118 - NiceLabel Automation Configuration - Cache Remote Files

With settings in NiceLabel Automation Configuration you can define how often should the files stored in the cache be refreshed, and how much time can they be stored in the cache without modification, before they are removed from the cache.

Loading Large Configuration Files

If you have really large configuration files with hundreds of triggers, data filters and variables, it can take some time for such files to load. In such situations it might be wise if you split one large configuration file into several smaller ones that will load faster and will also be easier to manage. Also you might reconsider how the configuration is designed. Are you sure you really need all those triggers? Could you simplify the configuration using variables?

Printing

Multi-core Printing

These days most of the computers have multi-core processors (multiple central processing units on a single chip). NiceLabel Automation can utilize all those cores. By default, half of the cores are used for triggers, processing filters and executing actions. NiceLabel Automation can have several printing processes running at the same time. It will take the number of available cores on the computer, divide it by 2 and use that number as the number of available printing processes. Most of the time that should be enough.

If you realize you need to allocate more (or less) printing processes, you can do so, but you have to be aware that the change might affect the performance of other modules in NiceLabel Automation.

To tell NiceLabel Automation how many processes it should use for printing, you have to manually change the *product.config* file. You can usually find it in the "c:\ProgramData\EuroPlus\NiceLabel Automation\system.net" folder (on Windows 7) or "c:\Documents and Settings\All Users\Application Data\EuroPlus\NiceLabel Automation\system.net" (on Windows XP). MaxConcurrentPrintProcesses and MaxConcurrentSessionPrintProcesses settings need to be added to the configuration file.

```
<configuration>
  <IntegrationService>
    <MaxConcurrentPrintProcesses>4</MaxConcurrentPrintProcesses>
    <MaxConcurrentSessionPrintProcesses>4</MaxConcurrentSessionPrintProcesses>
  </IntegrationService>
</configuration>
```

When you save the file, NiceLabel Automation will automatically pick up the changes and adjust its processing accordingly.

Tip

Before making any changes to the *product.config* file, you should make a backup copy.

Action Hierarchy

Make sure you do not put **Open Format** action inside **For loop** or any other action that would open label more than once (unless you need to change the label every iteration).

Label Design Considerations

There are lots of factors that can affect printing speed. One of them is also the design of the label itself.

If you are using lots of graphical objects (pictures, texts using TrueType fonts), or several large ones, the processing of such labels may take a lot of time and consume lots of memory.

When printing with NiceLabel printer drivers it is almost always a best practice to use printer internal elements. This means that you should use printer fonts instead of TrueType fonts, like Arial or Verdana. Each printer family has its own set of fonts.

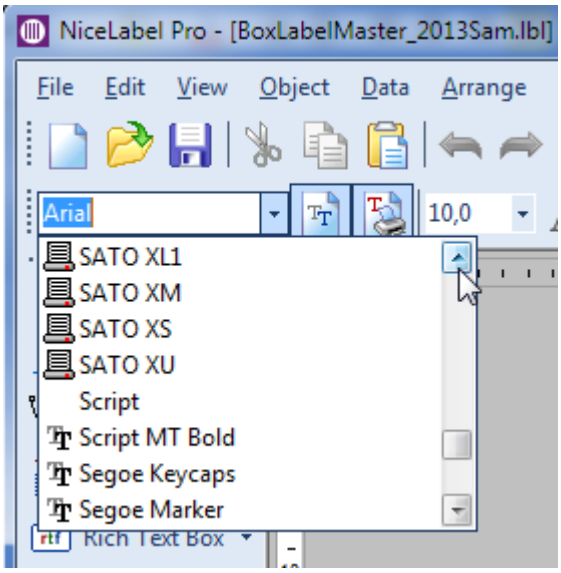


Figure 119 - NiceLabel Pro - Printer and TrueType Fonts

In the screenshot above you can see both printer and TrueType fonts. Since label was designed for SATO printer, you can see a few SATO fonts that selected printer supports.

Other label design elements like barcodes, rectangles, etc. can also be printed as printer internal elements. This means that much shorter commands can be sent to the printer and as a consequence better performance can be achieved. In the picture below you can see two barcodes. The left one will be printed as internal printer element (better printing performance), and the one on the right will be printed as graphics. You can distinguish which element is marked as printer element with a small printer icon in the bottom right corner of the object and by a thin green line around it.

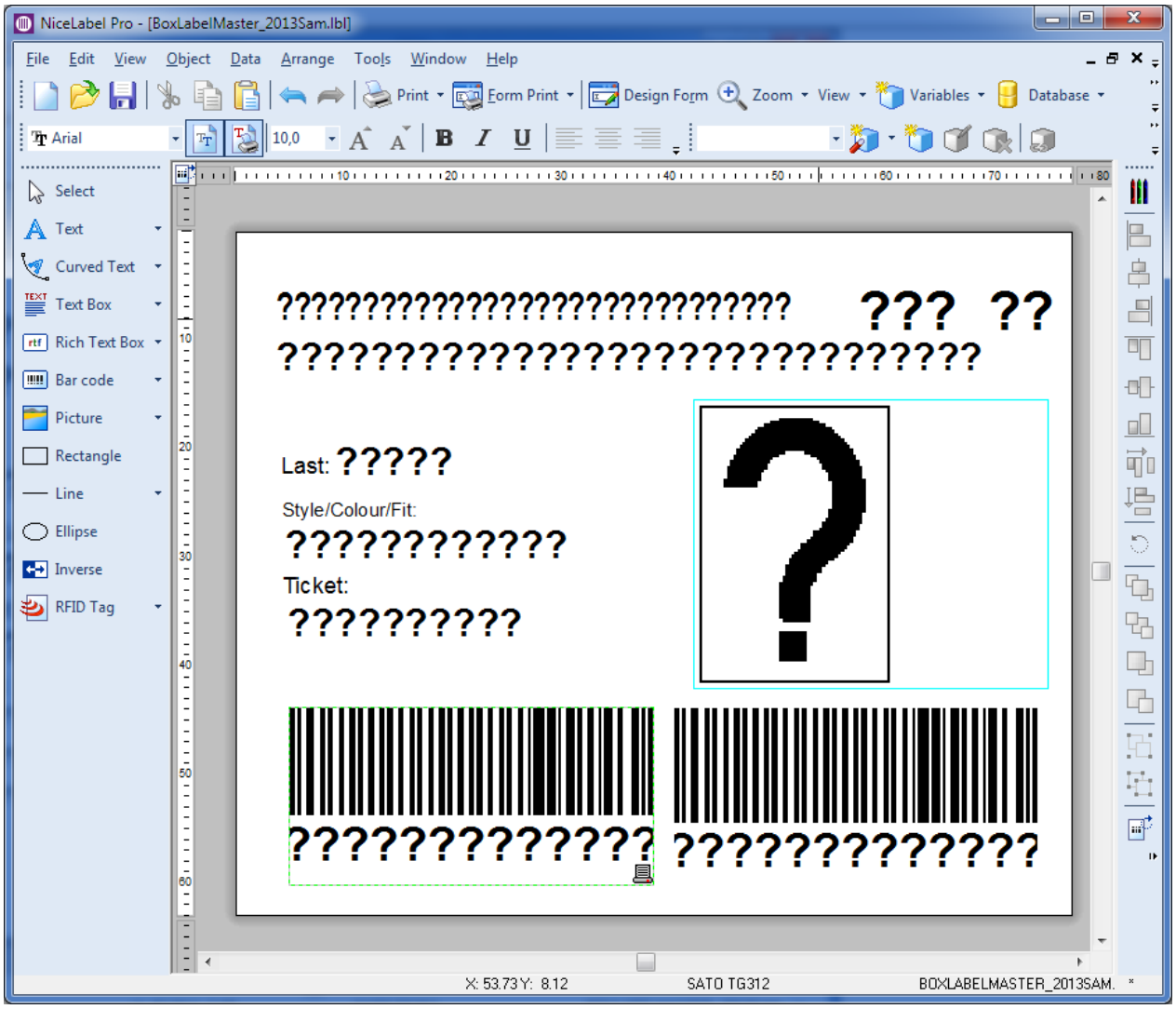


Figure 120 - NiceLabel Pro - Printer Internal Elements

Using global variables can also impact performance. Because their values cannot be stored in memory as for other types of variables, they have to be saved to a disk. Writing and reading from disk is always slower than just fetching the value from computer memory.

Logging to NiceLabel Enterprise Print Manager

If you are processing numerous triggers and printing large quantities of labels with lots of variables, and you log all those events and print jobs to NiceLabel Enterprise Print Manager, you might notice that NiceLabel Automation performance might not be the same as when NiceLabel Enterprise Print Manager logging is not enabled. In such situations I would recommend disabling logging of events that you do not need to track, or even disabling logging to NiceLabel Enterprise Print Manager altogether.

HTTP Server Trigger

By default HTTP Server Trigger returns the response immediately and then starts processing filters and actions. This allows NiceLabel Automation to handle large number of HTTP requests.

If you configure HTTP Server trigger to wait for the trigger execution to finish, then you can also set the maximum number of concurrent requests that NiceLabel Automation will allow.

TCP/IP Server Trigger and Web Service Trigger

Every TCP/IP Server trigger allows 10 concurrent connections. This means that 10 clients can connect to the trigger at the same time and send data to it. If your hardware can handle more clients at the same time, you can increase that number.

The same recommendation is valid for Web Service trigger, where you can define how many concurrent calls it will handle.

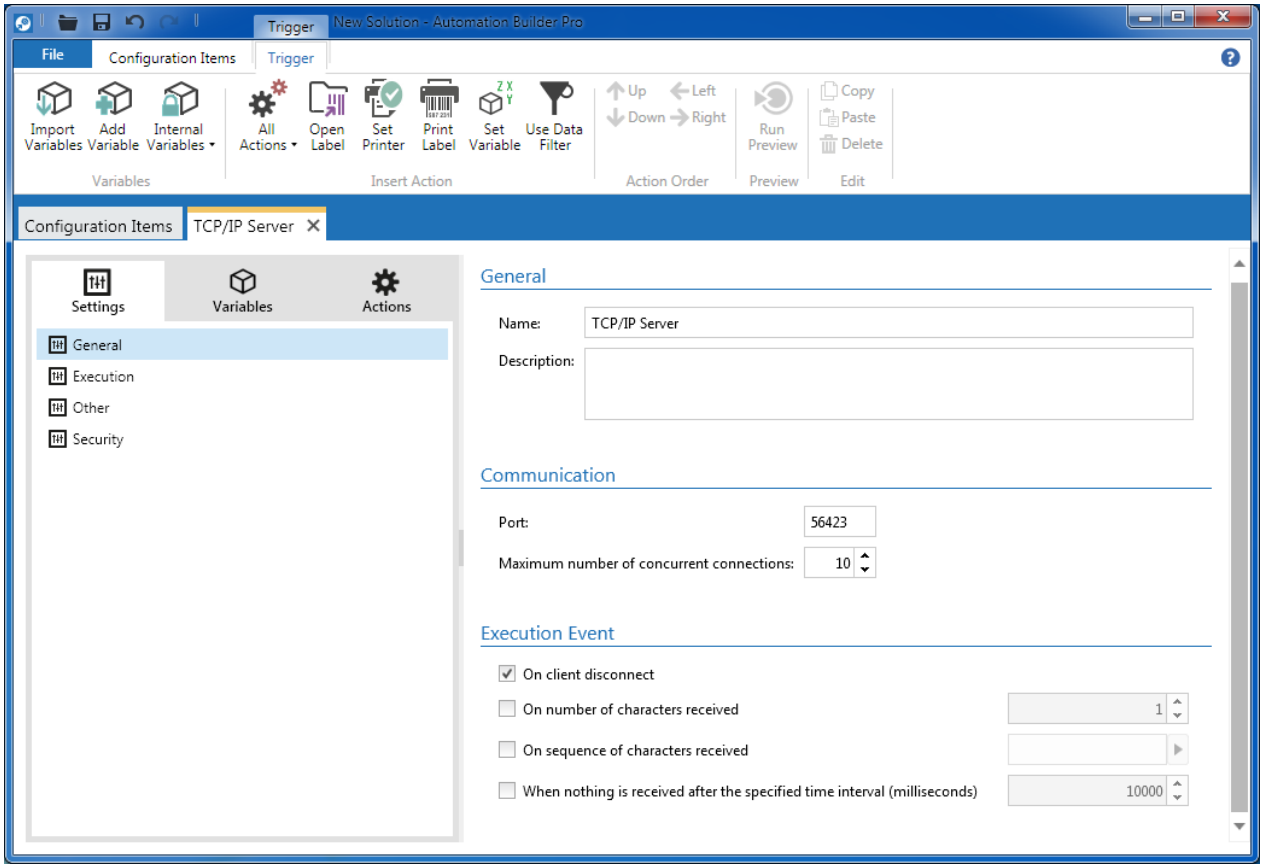


Figure 121 - TCP/IP Server Trigger - Concurrent Connections

When Things Go Wrong

Automation Manager Log

Automation Manager provides a detailed log of the activities that are happening in NiceLabel Automation and should be the first thing you check when things are not going as you expected them to. From the log you can see when triggers were started, stopped or executed, which actions were executed and when, along with action details like ID (very useful for finding the problematic action in Automation Builder) and description. To easily see log entries with problems, errors are colored in red and orange is used for warnings.

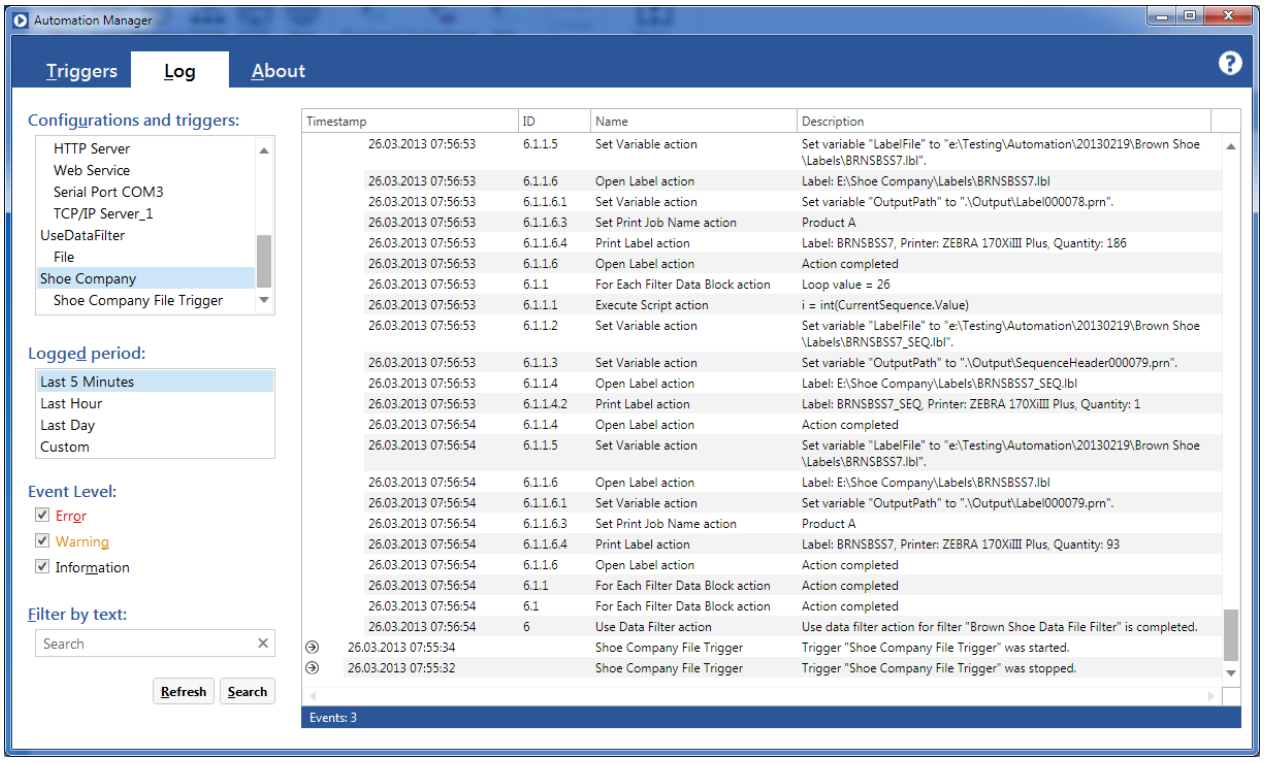


Figure 122 - Automation Manager - Log

Use Save Variable Data and Message Actions

When you have problems with variable values not being printed correctly or not being printed at all, you can use **Save Variable Data** and **Message** actions to help you determine what the cause is. Configure **Save Variable Data** action so that it will save only the variable you have problems with, or are used in actions that do not work as you expected them to. **Message** action also allows you to add an entry to the Automation Manager log with the value of variable. By adding one of these two actions before the Print Label action, you will see if the correct values are being printed.

Event Log

Sometimes an error happens that the authors of NiceLabel Automation did not expect. In such situations the error is saved in the Windows Event Log. You can view these logs by opening **Event Viewer**. NiceLabel Automation logs can be found in the **Application** section under the **Windows Logs**.

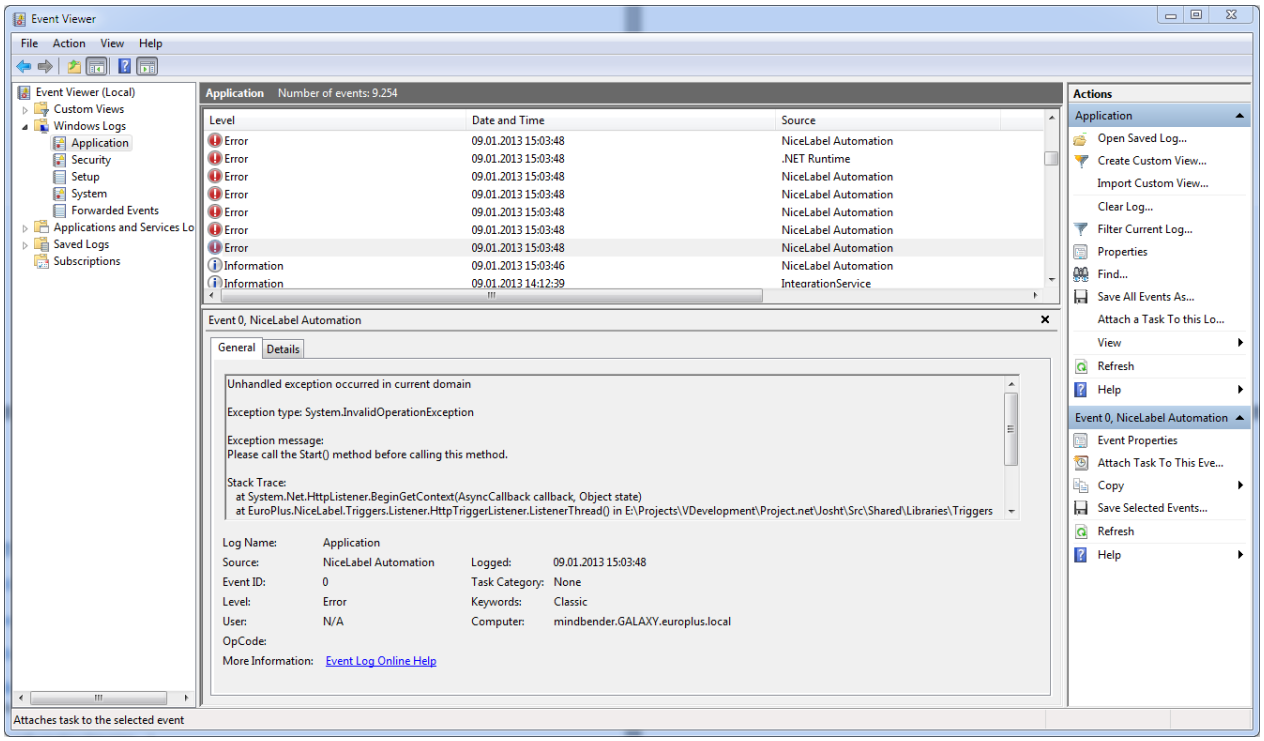


Figure 123 - Event Viewer

Contacting Support Team

At some point, when you encounter a problem that you cannot solve by yourself, you may need to contact your support representative for additional assistance.

Before contacting support it would be very helpful, if you collected information about the problem, the computer and operating system, you are running NiceLabel Automation on. You can use the following checklist:

- What happened? Write down exactly the error message that you get or make a screenshot of the error.
- When was the error discovered for the first time?
- What did you expect to happen?
- Have you discovered the problem earlier?
- Could the error be reproduced?
- How often did the error occur?
- Have you tried to use NiceLabel Automation this way earlier?
- If your problem is coming and going, please describe circumstances and under what conditions the error appears.

- Which version of NiceLabel Automation are you using?
- Which version of NiceLabel was used to design the label?
- Can you provide the files that you were using when the issue occurred (data files, configuration files, label files, etc.)?
- Which printer are you using?
- Which version of printer driver are you using?
- Which operating system are you using? In which language?
- Are you using firewall? Are all the required ports (ports used by triggers and actions) opened?
- Which version of Microsoft .NET framework is installed?

Appendices

Appendix A: Keyboard Shortcuts in Automation Builder

Shortcut	Action
ALT + F4	Close Automation Builder
Alt or F10	Select the active tab in the ribbon and activate the access keys
Alt or F10, Left or Right Arrow	Activate the active tab in the ribbon and move to another tab
CTRL + C	Copy (trigger, filter, variable, action)
CTRL + F4 or CTRL + W	Close current tab or close configuration if there are no tabs open
CTRL + N	Create new configuration
CTRL + O	Open an existing configuration
CTRL + S	Save an existing configuration
CTRL + V	Paste (trigger, filter, variable, action)
CTRL + Y	Redo
CTRL + Z	Undo
DELETE	Remove (trigger, filter, variable, action)
F1	Open help
F5	Refresh trigger preview

Table 5 - Keyboard Shortcuts in Automation Builder

Appendix B: Keyboard Shortcuts in Automation Manager

Shortcut	Action
ALT + A	Go to **About** tab (In English language only)
ALT + F4	Close Automation Manager
ALT + L	Go to **Log** tab (In English language only)
ALT + T	Go to **Triggers** tab (In English language only)
Alt or F10	Select the active tab in the ribbon and activate the access keys
CTRL + A	Select all triggers
CTRL + O	Add configuration
F1	Open help
F5	Refresh log
SPACE	Start or stop selected triggers

Table 6 - Keyboard Shortcuts in Automation Manager

Appendix C: NiceCommands

NiceLabel supports 24 commands that allow you to automate the printing process. You can use these commands in .JOB command files that you can run from NiceForm, NiceWatch, or from NiceLabel itself. NiceLabel Automation supports most of them. In NiceLabel Automation they can be used in **Run Command File** and **Send Custom Commands** actions.

Comment

If you want to **comment** a line so it will not be executed, put a semicolon (;) at the beginning of that line. NiceLabel Automation and NiceLabel will ignore such lines.

```
; LABEL shoes.lbl
```

CREATEFILE

CREATEFILE creates an ASCII text file that contains text "NiceLabel". The command might be used to create a file to signal some other application that some other command (e.g. **PRINT**) has finished processing.

```
CREATEFILE "C:\Tmp\ttt1.dat"
```

DELETEFILE

To delete arbitrary file use **DELETEFILE**. Command is often used in combination with **CREATEFILE**.

```
DELETEFILE "C:\Tmp\ttt1.dat"
```

IGNOREERROR

When IGNOREERROR is used it tells NiceLabel Automation and NiceLabel not to end the printing process if one of the following situations occur:

- Variable that does not exist is used
- Variable is set to incorrect value
- Label does not exist
- Label is not accessible
- Printer does not exist
- Printer is not accessible

```
IGNORERRROR ON
IGNOREERROR OFF
```

LABEL

LABEL command opens label design so that it can be printed. If label is stored on a network share you should use UNC notation. If file name contains space character, you should enclose it in double quotes.

```
LABEL shoes.lbl
LABEL "D:\Labels\Client 1\Product.lbl"
LABEL "\\Server\Labels\Client 1\Product.lbl"
```

MESSAGEBOX

In NiceLabel using **MESSAGEBOX** produces message box shown to the user. In NiceLabel Automation the command will add an entry to the Automation Manager log.

```
MESSAGEBOX "Short Message"
MESSAGEBOX "Short Message", "Caption"
```

PORT

To redirect printing to file, you can use the **PORT** command and pass a file name as parameter. If you add **APPEND** parameter to it, NiceLabel and NiceLabel Automation will not overwrite the file if it already exists, but will append to it.

The same port will be used until the last command is processed, or until the next **PORT** command is used.

```
PORT "D:\Output\Product.pnr"
PORT "D:\Output\Product.pnr", APPEND
```

PRINT

PRINT command is used to initiate printing. It takes one required (quantity) and three optional parameters (number of skipped labels, number of identical label copies and number of label sets).

Quantity parameter can either be a number of labels, or **VARIABLE** or **UNLIMITED**.

If optional parameters are omitted, then the following defaults are used:

- Number of skipped labels: 0
- Identical label copies: 1
- Number of label sets: 1

```
PRINT 100
PRINT VARIABLE
PRINT UNLIMITED
PRINT 10, 3
PRINT 10, 3, 2
PRINT 10, 3, 2, 5
```

PRINTER

PRINTER sets which printer should be used when printing. By default printer saved in label design is used. If printer name contains space character, you need to enclose it in double quotes.

```
PRINTER "Avery AP 5.4 300DPI"
PRINTER "\\SERVER\Altec TTP-343 Plus"
```

PRINTJOBNAME

PRINTJOBNAME command allows you to change the name of print job that appears in Windows Spooler.

```
PRINTJOBNAME "Small label"
```

SETPRINTPARAM

There are occasions when you need to set the paper tray you would like to use, or set printer speed and darkness, or calibrate the position of label objects. **SETPRINTPARAM** allows you to do that.

```
SETPRINTPARAM PAPERBIN = "TRAY 1"
SETPRINTPARAM PRINTSPEED = 12
SETPRINTPARAM PRINTDARKNESS = 95
SETPRINTPARAM PRINTOFFSETX = -12
SETPRINTPARAM PRINTOFFSETY = 10
```

SET

SET command is used to set variable values. Besides the obvious parameter – value, the command also supports two optional parameters: step and change after number of labels. These two optional parameters are used for setting counters. They define how much should the counter be increased (or decreased) and after how many labels the change should be done.

If the variable name or variable value contains space or comma character, you need to enclose it in text qualifier characters. By default these are double quotes. Text qualifier can be controlled by using **TEXTQUALIFIER** command.

If you need to set value that contains more than one line, you should encode the line delimiter by using \r\n.

```
SET Name = "Tony"
SET "Full Name" = "Derek Jetter"
SET Counter1 = 100, 1, 3
SET RiskPhrases = "Explosive when dry"\r\n"May cause fire"\r\n"Flammable"
```

TEXTQUALIFIER

Text qualifier is a character that encloses values. It is usually used when the value contains spaces.

```
TEXTQUALIFIER #
SET ProductDescription = #EPAK 12"X10 7/32"#
```

SESSIONSTART, SESSIONPRINT, SESSIONEND

SESSIONSTART, **SESSIONPRINT** and **SESSIONEND** commands need to be used together when you want to have “session print” as it is known in NiceLabel.

Usually when you use several **PRINT** commands in a row, new print job will be created for each of those commands. You can verify that by opening Windows Spooler for selected printer and observing how many documents will appear there. On thermal transfer printers this usually does not represent a problem,. However, if you want to print on a sheet of paper with more than one label (e.g. ZWECKFORM on A4 sheet of paper), every PRINT command will start on a new sheet of paper, instead of just continuing to print on a previous one where not all labels are used yet.

SESSIONSTART, **SESSION PRINT** and **SESSIONEND** solve that problem. **SESSIONSTART** starts the session print process, **SESSIONPRINT** prints the labels, and **SESSIONEND** closes the session.

There are some limitations when using "session print":

- You cannot change label within one session
- You can print only on one printer within one session
- You should set all variables used on a label before every SESSIONPRINT, because otherwise values from previous print are used

SESSIONPRINT command only takes two parameters: quantity and number of skipped labels (optional parameter).

```
PRINTER "SATO CL408e"
SESSIONSTART
SET Name = "Derek"
SET "Last Name" = "Jetter"
SET Position = "Shortstop"
SET Team = "New York Yankees"
SESSIONPRINT 10,2
SET Name = "Buster"
SET "Last Name" = "Posey"
SET Position = "Catcher"
SET Team = "San Francisco Giants"
SESSIONPRINT 8
SESSIONEND
```

Not Supported Commands

QUIT, **RETURN**, **LOGIN**, **LABELCLOSE**, **EXPORTLABEL**, **OEMTOANSI**, **SETDATABASE** and **SETTABLE** commands that NiceLabel recognizes, are not supported in NiceLabel Automation.

Note

NiceCommands are not case sensitive, so you can either use LABEL or label. NiceLabel Automation (and NiceLabel) will process both.

Appendix D: JOB Command Files

JOB command files contain NiceCommands described in the previous section. An example of such file would look like this:

```
; Graphics for shoes
LABEL "e:\shoe company\labels\brnsbss7.lbl"
PRINTER "ZEBRA 170XiIII Plus"
SET GraphicsPath = "e:\shoe company\graphics"
SET WIDTH = "WW"
SET DESC = "XXXXXXXXXXXXXXXXXXXXX"
SET PICTURE = "life28079"
SET PO = "XXXXXXXXX"
SET SAMPLE = "XXXXXXXXXX"
SET SEASON = "XX"
SET SIZE = "XXX"
SET UPC = "000000000000"
PRINT "99"
; Graphics for clothes
LABEL "e:\shoe company\labels\brnsbss7.lbl"
PRINTER "ZEBRA 170XiIII Plus 2"
SET GraphicsPath = "e:\clothing company\graphics"
SET WIDTH = "11"
SET DESC = "YYYYYYYYYYYYYYYYYYY"
SET PICTURE = "shirt28079"
SET PO = "123456"
SET SAMPLE = "XXXXXXXXXX"
SET SEASON = "WT"
SET SIZE = "M"
SET UPC = "000000000001"
PRINT "100"
```

Appendix E: CSV Command Files

CSV command files allow you to use a subset of NiceCommands in a CSV (comma separated values) format. **LABEL**, **PRINTER**, **PORT**, and implicitly **SET** and **PRINT** commands are supported. By implicitly I mean that every line automatically means a print action and just by adding a column with variable name, that variable will be set.

Order of columns in the file is not important, but you do have to ensure that values in all rows follow the same structure.

```
@Label, @Printer, @Quantity, @Skip, @IdenticalCopies, @NumberOfSets, @Port,"Graphics"
"e:\labels\brnsbss7.lbl", "ZEBRA 170XiIII Plus", 1, 0, 1, 1, , "e:\graphics\"
"e:\labels\brnsbss8.lbl", "DATAMAX DMX H-4212", 10, 0, 1, 1, , "e:\graphics 2\"
"e:\labels\brnsbss6.lbl", "TEC B-472", 100, 3, 2, 4, "c:\tmp\out.prn", "e:\graphics\"
```

As you can see from the sample above the following commands / columns can be used:

- Value in **@Label** column defines the name of the label design used.
- **@Printer** sets which printer will be used.
- If **@Port** is set, then printing will be redirected to a file or to a port that is different from the one, that is set in the printer driver.
- **@Quantity** defines how many labels should be printed.
- Value in **@Skip** field tells NiceLabel Automation how many labels should be skipped before the first one is printed.
- **@IdenticalCopies** specifies how many copies should be printed for each unique label.
- **@NumberOfSets** informs NiceLabel Automation how many times should the whole printing process for selected label be repeated.

All other columns define variable values that will then be used on a printed label. In the sample above only one variable (Graphics) is set.

Appendix F: XML Command Files

```
<nice_commands quit="false">
  <label name="e:\shoe company\labels\brnsbss7.lbl" close="true">
    <print_job printer="TEC B-472" quantity="1" skip="0" identical_copies="1"
job_name="Big">
      <variable name="GraphicsPath">e:\shoe company\graphics\</variable>
      <variable name="WIDTH">WW</variable>
      <variable name="DESC">XXXXXXXXXXXXXXXXXXXX</variable>
      <variable name="PICTURE">life28079</variable>
      <variable name="PO">XXXXXXXX</variable>
      <variable name="SAMPLE">XXXXXXXXX</variable>
      <variable name="SEASON">XX</variable>
      <variable name="SIZE">XXX</variable>
      <variable name="UPC">000000000000</variable>
    </print_job>
    <session_print_job printer="TEC B-452" >
      <session quantity="10">
        <variable name="GraphicsPath">e:\shoe company\graphics\</variable>
        <variable name="WIDTH">WW</variable>
        <variable name="DESC">YYYYYYYYYYYYYYYYY</variable>
        <variable name="PICTURE">life25000</variable>
        <variable name="PO">YYYYYYYYY</variable>
        <variable name="SAMPLE">YYYYYYYYY</variable>
        <variable name="SEASON">YY</variable>
        <variable name="SIZE">YYY</variable>
        <variable name="UPC">111111111111</variable>
      </session>
    <session quantity="12">
        <variable name="GraphicsPath">e:\shoe company\graphics\</variable>
        <variable name="WIDTH">WW</variable>
        <variable name="DESC">ZZZZZZZZZZZZZZZZZ</variable>
        <variable name="PICTURE">life20000</variable>
        <variable name="PO">ZZZZZZZZZZ</variable>
        <variable name="SAMPLE">ZZZZZZZZZZ</variable>
        <variable name="SEASON">ZZ</variable>
        <variable name="SIZE">ZZZ</variable>
        <variable name="UPC">222222222222</variable>
      </session>
    </session_print_job>
  </label>
</nice_commands>
```

NiceLabel Automation also enables you to use command files in a form of an XML file. The following XML elements can be used in an XML command file:

- **<nice_commands>** is the root element of the XML document and as such there should be only one instance of the element in the file.
- **<label>** element defines which label file will be used. Use the **name** attribute to specify the path to the file. There should be at least one **<label>** element in the file and it should be under the root element.
- **<print_job>** specifies a single print job that is defined by the following attributes:
  - **printer**

- **quantity**
- **skip**
- **identical_copies**
- **number_of_sets**
- **job_name** specified the name of the print job visible in Windows Spooler.
- **print_to_file** defines the file name where NiceLabel Automation should redirect printing (this is an optional value)
- If **print_to_file_append** is set to true, then print stream will be added to the file defined in the print_to_file attribute.
- Set **clear_variable_values** to true if you want to reset variables to their default values after the printing is done.

- **<session_print_job>** can contain one or more **<session>** elements. That allows you to implement session printing from XML command files. You can use the same attributes as with **<print_job>** element with the exception of the **quantity** attribute that has to be used with **<session>** element.

If you know anything about XML schema definition (XSD - (Wikipedia)) files, then the following schema might help you in understanding the structure of XML command files.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://tempuri.org/XMLSchema.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/XMLSchema.xsd"
xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="nice_commands">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="label" maxOccurs="unbounded" minOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="print_job" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="database" maxOccurs="unbounded" minOccurs="0">
                      <xs:complexType>
                        <xs:simpleContent>
                          <xs:extension base="xs:string">
                            <xs:attribute name="name" type="xs:string" use="required" />
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="table" maxOccurs="unbounded" minOccurs="0">
                      <xs:complexType>
                        <xs:simpleContent>
                          <xs:extension base="xs:string">
                            <xs:attribute name="name" type="xs:string" use="required" />
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="variable" maxOccurs="unbounded" minOccurs="0">
                      <xs:complexType>
```

```
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="name" type="xs:string" use="required" />
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="quantity" type="xs:string" use="required" />
              <xs:attribute name="printer" type="xs:string" use="optional" />
              <xs:attribute name="skip" type="xs:integer" use="optional" />
              <xs:attribute name="identical_copies" type="xs:integer" use="optional"
/>
              <xs:attribute name="number_of_sets" type="xs:integer" use="optional" />
              <xs:attribute name="job_name" type="xs:string" use="optional" />
              <xs:attribute name="print_to_file" type="xs:string" use="optional" />
              <xs:attribute name="print_to_file_append" type="xs:boolean"
use="optional" />
              <xs:attribute name="clear_variable_values" type="xs:boolean"
use="optional" />
            </xs:complexType>
          </xs:element>
          <xs:element name="session_print_job" maxOccurs="unbounded" minOccurs="0">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="database" maxOccurs="unbounded" minOccurs="0">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="name" type="xs:string" use="required" />
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
                <xs:element name="table" maxOccurs="unbounded" minOccurs="0">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="name" type="xs:string" use="required" />
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
                <xs:element name="session" minOccurs="1" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="variable" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="name" type="xs:string"
use="required" />
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="quantity" type="xs:string" use="required" />
```

```
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="printer" type="xs:string" use="optional" />
                <xs:attribute name="skip" type="xs:integer" use="optional" />
                <xs:attribute name="job_name" type="xs:string" use="optional" />
                <xs:attribute name="print_to_file" type="xs:string" use="optional" />
                <xs:attribute name="print_to_file_append" type="xs:boolean"
use="optional" />
                <xs:attribute name="clear_variable_values" type="xs:boolean"
use="optional" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="name" type="xs:string" use="required" />
          <xs:attribute name="close" type="xs:boolean" use="required" />
          <xs:attribute name="clear_variable_values" type="xs:boolean" use="optional" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="quit" type="xs:boolean" use="required" />
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Acronyms

Acronym	Meaning
CSV	Comma Separated Values
DTD	Document Type Definition
EPM	Enterprise Print Manager
ERP	Enterprise Resource Planning (Wikipedia)
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure (Wikipedia)
IP	Internet Protocol (Wikipedia)
MSCA	Mobile Supply Chain Architecture
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UNC	Uniform Naming Convention
URL	Uniform Resource Locator
WMS	Warehouse Management (Oracle)
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Bibliography

Gilbert, H. (n.d.). *Introduction to TCP/IP.* Retrieved April 10th, 2013, from http://www.yale.edu/pclt/COMM/TCPIP.HTM

Internet Assigned Numbers Authority (IANA). (n.d.). *Service Name and Transport Protocol Port Number Registry.* Retrieved 04 08, 2013, from Internet Assigned Numbers Authority (IANA): https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

Load balancing (computing). (n.d.). Retrieved March 30th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Load_balancing_(computing)

Microsoft. (n.d.). *.NET Framework Regular Expressions.* Retrieved March 13th, 2013, from MSDN: http://msdn.microsoft.com/en-us/library/hs600312(v=vs.100).aspx

Microsoft Corp. (n.d.). *Failover Clustering Overview.* Retrieved March 30th, 2013, from TechNet: http://technet.microsoft.com/en-us/library/hh831579.aspx

Microsoft Corp. (n.d.). *Load-Balanced Cluster.* Retrieved March 30th, 2013, from MSDN: http://msdn.microsoft.com/en-us/library/ff648960.aspx

Oracle. (n.d.). *Label Printing Integration Kit.* Retrieved March 12th, 2013, from Oracle Warehouse Management Implementation Guide: http://docs.oracle.com/cd/E18727_01/doc.121/e13434/T210618T210847.htm

Python (programming language). (n.d.). Retrieved March 14th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Python_(programming_language)

Wikipedia. (n.d.). *Comma-Separated Values.* Retrieved March 9th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Comma-separated_values

Wikipedia. (n.d.). *Document Type Definition.* Retrieved April 10th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Document_Type_Definition

Wikipedia. (n.d.). *For Loop.* Retrieved March 14th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/For_loop

Wikipedia. (n.d.). *HTTP.* Retrieved March 9th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Http

Wikipedia. (n.d.). *HTTP Secure.* Retrieved March 13th, 2013, from http://en.wikipedia.org/wiki/HTTP_Secure

Wikipedia. (n.d.). *Internet Protocol Suite.* Retrieved March 9th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Internet_protocol_suite

Wikipedia. (n.d.). *List of HTTP status codes.* Retrieved March 9th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Wikipedia. (n.d.). *List of TCP and UDP port numbers.* Retrieved 03 2013, from Wikipedia: http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

Wikipedia. (n.d.). *SAP ERP.* Retrieved March 12th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/SAP_ERP

Wikipedia. (n.d.). *Serial Port.* Retrieved March 14th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Serial_port

Wikipedia. (n.d.). *SOAP.* Retrieved March 9th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/SOAP

Wikipedia. (n.d.). *Uniform Naming Convention.* Retrieved March 11th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Uniform_Naming_Convention#Uniform_Naming_Convention

Wikipedia. (n.d.). *Unstructured Data.* Retrieved March 20th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Unstructured_data

Wikipedia. (n.d.). *VBScript.* Retrieved March 14th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Vbscript

Wikipedia. (n.d.). *Web Service.* Retrieved March 9th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Web_service

Wikipedia. (n.d.). *XML.* Retrieved March 14th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/XML

Wikipedia. (n.d.). *XML Schema.* Retrieved April 10th, 2013, from Wikipedia: http://en.wikipedia.org/wiki/XML_Schema_(W3C)

Wikipedia. (n.d.). *XSLT.* Retrieved March 15th, 2013, from Wikiepdia: http://en.wikipedia.org/wiki/Xslt

World Wide Web Consortium (W3C). (n.d.). *Web Services Description Language (WSDL) 1.1.* Retrieved March 9th, 2013, from World Wide Web Consortium (W3C): http://www.w3.org/TR/wsdl

Table of Figures

Tables

Index

H

I

L

N

P

Q

R

S

T

U

V

W

X

Attributions

Cover Page Image

Photo used on the cover of this book was taken by Mark J. Sebastian (Flickr: Warpspeed Drive-by (#70265)) [CC-BY-SA-2.0 (http://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons.

Tech Support Cheat Sheet

Tech support cheat sheet image used in the Troubleshooting chapter is by Randal Munroe, HTTP://XKCD.COM/627/.