Loftware NiceLabel

NiceLabel Automation 10 User Guide

Rev-2021-10

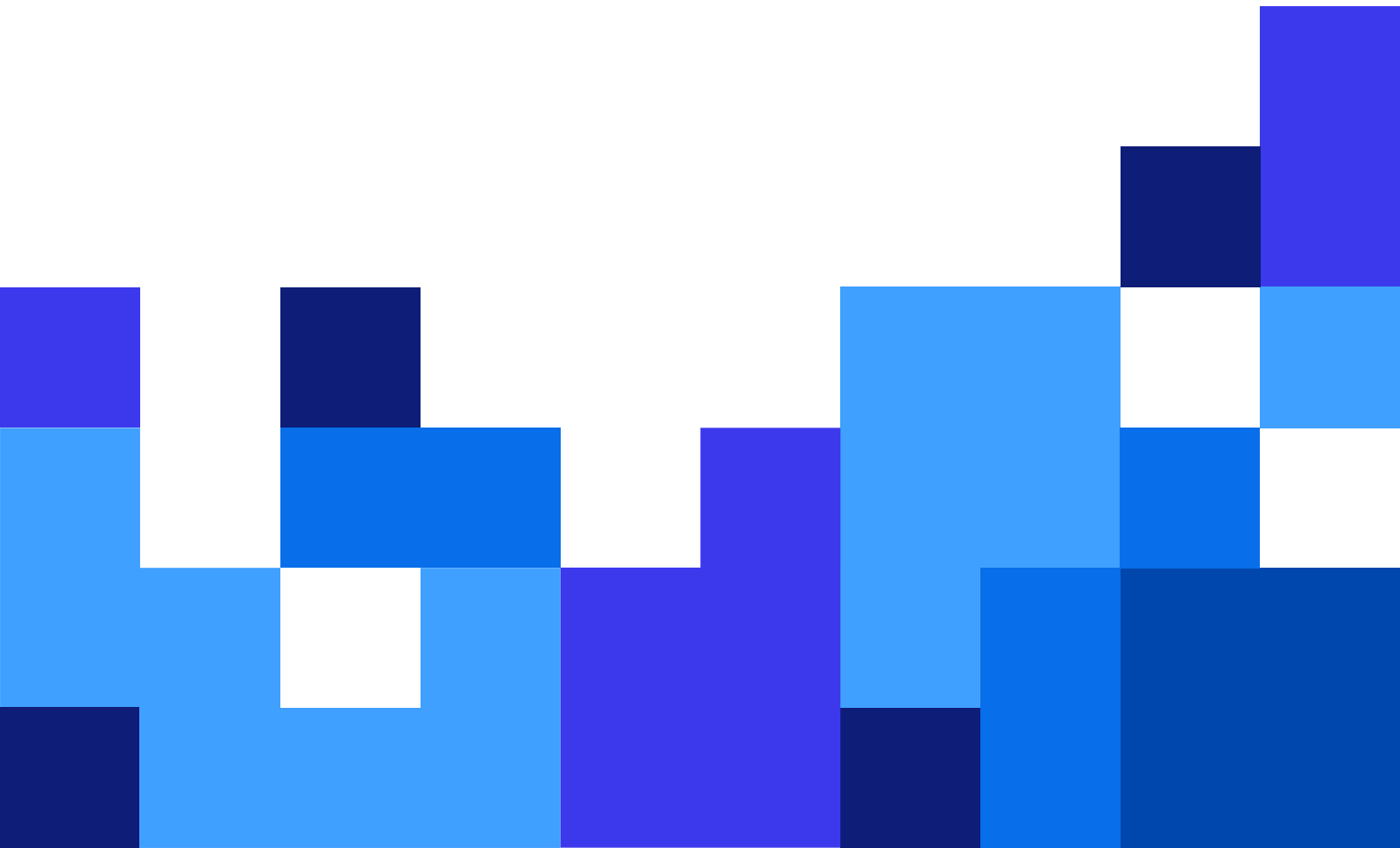


Table of Contents

1. Welcome to NiceLabel Automation 7
1.1. Architecture 8
1.2. System Requirements 9
1.3. Installation 9
1.4. Activation 10
1.5. Trial Mode 10
1.6. File Tab 11
1.6.1. Open 11
1.6.2. Compatibility with NiceWatch Products 12
1.6.3. Save 13
1.6.4. Save as 14
1.6.5. Options 14
1.6.6. About 22
1.6.6.1. Losing your Label Cloud connection 23
2. Understanding Filters 24
2.1. Configuring Structured Text Filter 26
2.1.1. Structured Text Filter 26
2.1.2. Defining Fields 27
2.1.3. Enabling Dynamic Structure 30
2.2. Configuring Unstructured Data Filter 32
2.2.1. Unstructured Data Filter 32
2.2.2. Defining Fields 35
2.2.3. Defining Sub Areas 39
2.2.4. Defining Assignment Areas 41
2.3. Configuring XML filter 43
2.3.1. XML Filter 43
2.3.2. Defining XML Fields 44
2.3.3. Defining Repeatable Elements in XML Filter 47
2.3.4. Defining XML Assignment Area 49
2.4. Configuring JSON filter 52
2.4.1. JSON Filter 52
2.4.2. Defining JSON Fields 54
2.4.3. Defining Repeatable Elements in JSON Filter 56
2.4.4. Defining JSON Assignment Area 59
2.5. Setting Label and Printer Names from Input Data 62
3. Understanding Triggers 64
3.1. File Trigger 66
3.2. Serial Port Trigger 70
3.3. Database Trigger 72
3.4. TCP/IP Server Trigger 81
3.5. TCP/IP Client Trigger 85
3.6. HTTP Server Trigger 89
3.7. Web Service Trigger 95

3.8. Cloud Trigger 105
3.8.1. Deploying Cloud Trigger with NiceLabel Cloud 105
3.8.1.1. Deployment Stages for NiceLabel Cloud 105
3.8.1.2. Configuring cloud trigger in Automation Builder 107
3.8.1.3. Setting up Cloud Trigger Access for the External integrator 109
3.8.1.4. Creating subscription on the Developer Portal 110
3.8.1.5. Calling your cloud trigger (NiceLabel Cloud deployment) 113
3.8.1.6. Quick Check if your Cloud trigger works 114
3.8.2. Deploying Cloud Trigger with your on-premise Control Center 114
3.8.2.1. Configuring cloud trigger in Automation Builder 114
3.8.2.2. Calling the cloud trigger (On-premise deployment) 117
3.9. Scheduler Trigger 117
3.9.1. General 118
3.9.2. Recurrences 118
3.10. Testing Triggers 120
3.11. Using Secure Transport Layer (HTTPS) 123
3.12. Protecting Trigger Configuration from Editing 127
3.13. Configuring Firewall for Network Triggers 127
4. Variables 129
4.1. Using Compound Values 130
4.2. Internal Variables 131
4.3. Global Variables 133
5. Actions 135
5.1. Defining actions 135
5.2. Nested actions 135
5.3. Action execution 136
5.4. Conditional actions 136
5.5. Identifying actions in configuration error state 137
5.6. Disabling actions 137
5.7. Copying actions 137
5.8. Navigating the action list 138
5.9. Describing the actions 138
5.10. General 138
5.10.1. Open Label 138
5.10.2. Print Label 140
5.10.3. Run Oracle XML Command FileXML Transform 144
5.10.4. Run SAP AII XML Command File 145
5.10.5. Run Command File 147
5.10.6. Send Custom Commands 149
5.11. Printer 151
5.11.1. Set Printer 151
5.11.2. Set Print Job Name 152
5.11.3. Redirect Printing to File 154
5.11.4. Set Print Parameter 156
5.11.5. Redirect Printing to PDF 163
5.11.6. Printer Status 165

5.11.7. Store Label to Printer 168
5.11.8. Print PDF Document 170
5.12. Variables 172
5.12.1. Set Variable 172
5.12.2. Save Variable Data 173
5.12.3. Load Variable Data 175
5.12.4. String Manipulation 177
5.13. Batch Printing 180
5.13.1. For Loop 180
5.13.2. Use Data Filter 182
5.13.3. For Every Record 185
5.14. Data & connectivity 187
5.14.1. Open Document/Program 187
5.14.2. Save Data to File 189
5.14.3. Read Data from File 191
5.14.4. Delete File 193
5.14.5. Execute SQL Statement 195
5.14.6. Send Data to TCP/IP Port 200
5.14.7. Send Data to Serial Port 202
5.14.8. Read Data from Serial Port 204
5.14.9. Send Data to Printer 206
5.14.10. HTTP Request 208
5.14.11. Web Service 211
5.15. Other 214
5.15.1. Get Label Information 214
5.15.2. Execute Script 219
5.15.2.1. Script Editor 221
5.15.3. Message 222
5.15.4. Verify License 225
5.15.5. Try 227
5.15.6. XML Transform 229
5.15.7. Group 231
5.15.8. Log Event 233
5.15.9. Preview Label 234
5.15.10. Create Label Variant 236
6. Running and Managing Triggers 239
6.1. Deploying Configuration 239
6.2. Event Logging Options 240
6.3. Managing Triggers 241
6.4. Using Event Log 243
6.5. If your configuration fails to load... 246
7. Performance and Feedback Options 249
7.1. Parallel Processing 249
7.2. Caching Files 250
7.3. Error Handling 252
7.4. Synchronous Print Mode 254

7.4.1. Asynchronous Print Mode 254
7.4.2. Synchronous Print Mode 254
7.5. Print Job Status Feedback 255
7.6. Excluding printers from automated printing 258
7.7. Using Store/Recall Printing Mode 259
7.8. High-availability (Failover) Cluster 260
7.9. Load-balancing Cluster 261
8. Understanding Data Structures 262
8.1. Binary Files 262
8.1.1. Example 262
8.2. Command Files 263
8.2.1. Example 263
8.3. Compound CSV 264
8.3.1. Example 264
8.4. Legacy Data 264
8.4.1. Example 264
8.5. Text Database 265
8.5.1. Example 265
8.6. XML Data 266
8.6.1. Examples 267
8.7. JSON Data 269
9. Reference and Troubleshooting 273
9.1. Command File Types 273
9.1.1. Command Files Specifications 273
9.1.2. CSV Command File 273
9.1.2.1. Sample CSV Command File 273
9.1.3. JOB Command File 274
9.1.3.1. Sample JOB Command File 274
9.1.4. XML Command File 275
9.1.4.1. Sample XML Command File 275
9.1.5. Oracle XML Specifications 280
9.1.5.1. XML DTD 280
9.1.5.2. Sample Oracle XML 280
9.1.6. SAP AII XML Specifications 281
9.1.6.1. Sample SAP AII XML 281
9.2. Custom Commands 282
9.2.1. Using Custom Commands 282
9.3. Access to Network Shared Resources 289
9.4. Document Storage and Versioning of Configuration Files 290
9.5. Accessing Databases 291
9.5.1. 32-bit Windows 291
9.5.2. 64-bit Windows 292
9.6. Automatic Font Replacement 292
9.7. Automating Reports 294
9.7.1. Creating temporary databases 295
9.7.2. Designing automated Reports 296

9.7.3. Creating data filters 296
9.7.4. Creating triggers for your new data filter 297
9.8. Changing Multi-threaded Printing Defaults 298
9.9. Compatibility with NiceWatch Products 299
9.10. Controlling Automation Service with Command-line Parameters 301
9.11. Database Connection String Replacement 305
9.12. Entering Special Characters (Control Codes) 306
9.13. List of Control Codes 307
9.14. Licensing and Printer Usage 308
9.15. Running in Service Mode 309
9.16. Search Order for Requested Files 311
9.17. Securing Access to your Triggers 312
9.18. Session Printing 313
9.19. Tips and Tricks for Using Variables in Actions 315
9.20. Tracing Mode 316
9.21. Understanding Printer Settings and DEVMODE 317
9.22. Using the Same User Account to Configure and Run Triggers 319

1. Welcome to NiceLabel Automation

NiceLabel Automation is an application that automates repetitive tasks. In most cases, you will be using it to integrate label printing processes into existing information systems, such as various business applications, production and packaging lines, distribution systems, and supply chains. With NiceLabel Automation, all applications across all divisions and locations in your company can print labels using authorized labels templates.

NiceLabel Automation helps you deploy and run an optimal business label printing system by synchronizing business events with label production. Automated printing without human interaction is by far the most effective way to remove user errors and to maximize performance.

Label printing automation using a trigger-based application revolves around the following three core processes:

Trigger
Triggers are a simple but powerful function that help automate your work. At its core, a trigger is a cause and effect of the statement: "If a monitored event happens, do something."

Here, we are talking about **IF .. THEN** processing. Triggers are good at handling repetitive events.

Automated label printing is triggered by a business operation. NiceLabel Automation is set to monitor a folder, file, or a communication port. If a business operation takes place, a file change or incoming data is detected. This triggers the label printing process.

Learn more about various Triggers:

- File Trigger
- Serial Port Trigger
- Database Trigger
- Scheduler Trigger
- TCP/IP Server Trigger
- TCP/IP Client Trigger
- HTTP Trigger
- Web Service Trigger
- Cloud Trigger

Data Extraction and Placement

Once a trigger starts the printing, NiceLabel Automation extracts label data and inserts it into variable objects that are placed on a label.

Data extraction Filters support:

- Structured text files
- Unstructured text files
- Various XML files
- JSON files

Action Execution

After matching the data with variable objects on a label, NiceLabel Automation starts performing actions. A basic set of actions usually includes **Open Label** and **Print Label** to print the extracted data on a label. You can also send data to custom local or network file locations, Web servers, hardware devices, and much more.

There are more than 30 actions available. Together, they cover a vast majority of scenarios that are common in today's business environments.

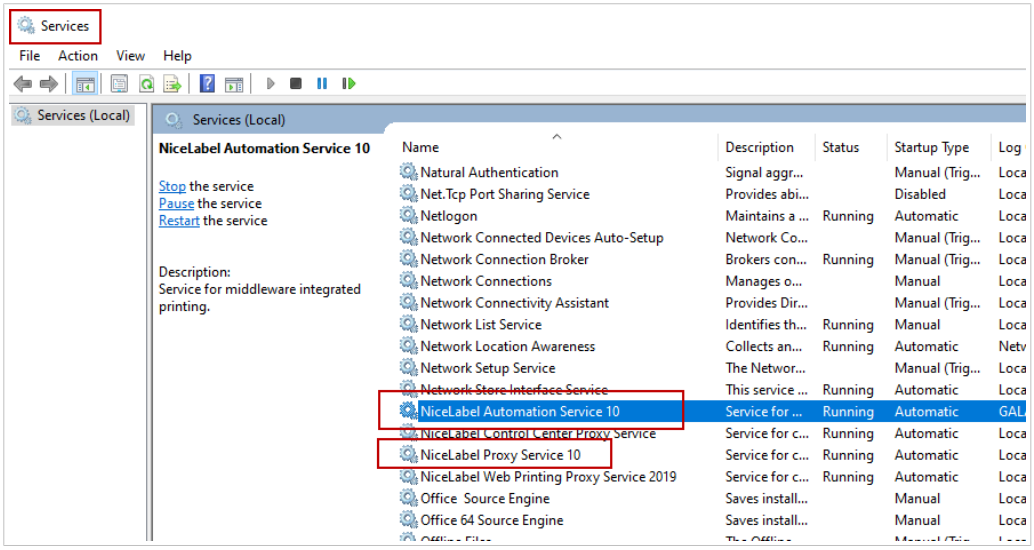
Learn more about basic and advanced printing Actions.

1.1. Architecture

NiceLabel Automation is a service-based application. Execution of all rules and actions runs as a background process under credentials of the user account that is defined for the Service.

NiceLabel Automation consists of three components.

- Automation Builder**:** Developer uses this application to create triggers, filters and actions, and to use them in a working configuration. Actions that belong to such a configuration execute after a trigger receives data. This application always runs as a 32-bit application.
- Automation Manager**:** This is the management application that monitors the execution of triggers in real time, and orders the triggers to start or stop. Automation Manager always runs as a 32-bit application.
- NiceLabel Automation **Service:** This is the 'print engine' that executes rules defined in the triggers. There are two service applications: NiceLabel Automation Service and NiceLabel Proxy Service. The Service always detects 'bitness' of the Windows machine and runs it at the same level (e.g. as a 64-bit application on 64-bit Windows), while the Proxy Service always runs as a 32-bit process.



1.2. System Requirements

NOTE

Always double-check for the latest system requirements on this web page: System Requirements.

1.3. Installation

NOTE

Below is the summarized version of the installation procedure. For more information, see the NiceLabel Automation Installation Guide.

Before you begin with the installation, make sure your infrastructure is compatible with System Requirements.

To install NiceLabel Automation, do the following:

- Download the installation file from NiceLabel website, then run the downloaded executable file.
- Insert NiceLabel DVD.
    1. Main application menu starts automatically.
       If the main application menu does not start, double click the **START . EXE** file on the DVD.
    2. Click **Install** NiceLabel.
    3. Follow **Setup Wizard** prompts.

During the installation, Setup prompts for user name under which NiceLabel Automation service is going to run under. Make sure you select a real user name, because the service inherits privileges associated with that specific user name. For more information, see section Running in Service Mode.

Version upgrading

To upgrade NiceLabel Automation to the latest release, install the new version on top of the installed one by overwriting it. During the upgrade, the old version is removed and replaced with the new one. All existing settings are preserved. Upgrade deletes the log database content.

1.4. Activation

Activate NiceLabel Automation to enable processing of configured triggers. The activation procedure requires an Internet connection – if possible on the same machine that runs the software. Use the same activation procedure to activate the trial license key.

NOTE

You can activate the software either from Automation Builder or Automation Manager.

Activation in Automation Builder

1. Run Automation Builder.
2. Go to **File > About > Activate Your License**.
   This starts the Activation Wizard.
3. Follow the on-screen instructions.

Activation in Automation Manager

1. Run Automation Manager.
2. Go to **About** tab.
3. Click **Activate Your License**.
4. Follow the on-screen instructions.

1.5. Trial Mode

Trial mode allows you to test NiceLabel Automation for up to 30 days. Trial mode offers the same feature set as a licensed version, and therefore enables complete evaluation of the product before you purchase

it. Automation Manager continuously displays the trial notification message and the number of remaining trial days. After the trial mode expires, NiceLabel Automation service stops processing triggers. The countdown of 30 days begins with the day of installation.

NOTE

You can extend the trial mode by contacting your NiceLabel reseller and requesting additional trial license key. You have to activate the trial license key. For more information, see section Activation.

1.6. File Tab

File tab serves as document management panel. The below listed options are available:

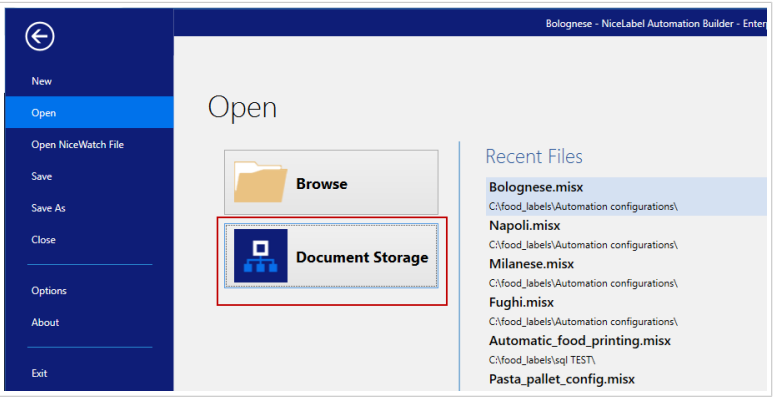
- **New**: creates a new configuration file.
- Open: opens existing configuration files.
- **Open NiceWatch File**: opens a legacy NiceLabel NiceWatch configuration.
- Save: saves the active configuration file.
- Save as: allows saving the active configuration file by defining its name and location.
- Options: opens the dialog for configuring the program defaults.
- About: provides license and software version information.
- **Exit**: closes the application.

1.6.1. Open

Open dialog allows you to open the existing configurations in Automation Builder.

Browse allows selecting the configuration files on local or connected network drives.

Document Storage opens the document storage location of the connected NiceLabel Control Center. The **Document Storage** option allows you to manage your copy of the stored configuration file.



Recent Files field lists the latest configuration files that have been edited. Click any of them to open the file.

1.6.2. Compatibility with NiceWatch Products

NiceLabel Automation allows you to load trigger configurations that were built using legacy NiceWatch products. In the majority of cases, you can run NiceWatch configuration using NiceLabel Automation without any modifications.

NiceLabel Automation products are using a new .NET based print engine optimized for performance and low memory footprint. The new print engine does not support each label design option that is available in the label designer. Each new release of NiceLabel Automation is closing the gap, but you might still come across certain unavailable features.

Resolving Incompatibility Issues
NiceLabel Automation warns you if you try to print existing label templates that contain design features which are not supported with the new print engine.

If there are incompatibilities with NiceWatch configuration file or label templates, Automation notifies you about:

- **Compatibility with trigger configuration:** While opening the NiceWatch configuration (.MIS file), NiceLabel Automation checks it against the supported features. Not all features from NiceWatch are available in NiceLabel Automation. While some are not available at all, some of them are configured differently. If the MIS file contains unsupported features, you will see them listed. Automation removes these features from the configuration.
  If you come across such a case, open the .MIS file in Automation Builder and resolve the incompatibility issues. You will have to use the available NiceLabel Automation features to re-create the removed configuration items.
- **Compatibility with label templates:** If your existing label templates contain unsupported print engine features as provided by NiceLabel Automation, you will see error messages in the **Log** pane. This information is visible in the Automation Builder (when designing triggers) or in Automation Manager (when running the triggers).
  In this case, you have to open the label file in label designer and remove unsupported features from the label.

NOTE
For more information about incompatibility issues with NiceWatch and label designers, see NiceLabel Automation Compatibility with NiceLabel Designers V6 and NiceWatch V5.

Opening NiceWatch Configuration for Editing

Open the existing NiceWatch configuration (.MIS file) in Automation Builder and edit it in Automation Builder. You can save the configuration as a .MISX file only.

To edit the NiceWatch configuration, do the following:

1. Start Automation Builder.
2. Go to **File** > **Open NiceWatch File**.
3. In the **Open** dialog box, browse for the NiceWatch configuration file (.MIS file).
4. Click **OK**.
5. If the configuration contains unsupported features, they are displayed in a list. Automation removes them from the configuration.

Opening NiceWatch Configuration for Execution

You can open NiceWatch configuration (.MIS file) in Automation Manager without conversion to the NiceLabel Automation file format (.MISX file). If the triggers from NiceWatch are compatible with NiceLabel Automation, you can start using them right away.

To open and deploy NiceWatch configuration, do the following:

1. Start Automation Manager.
2. Click **+Add** button.
3. In **Open** dialog box, change the file type into **NiceWatch Configuration**.
4. Browse for the NiceWatch configuration file (.MIS file).
5. Click **OK**.
6. Automation Manager will display the trigger from the selected configuration. To start the trigger, select it and click **Start**.

NOTE

If there are compatibility issues with NiceWatch configuration, open it in Automation Builder and reconfigure it.

1.6.3. Save

Save saves the active configuration using the same file name that was used for opening it.

NOTE

If a configuration has been opened for the first time, **Save** directs you to the **Save as** dialog.

1.6.4. Save as

Save as allows saving the active configuration file by defining its name and location. You can save your configuration files to your local disc or **Document Storage** in your NiceLabel Control Center

Recent folders field lists the folders that were recently used for saving the configuration files.

1.6.5. Options

Use settings in this dialog box to customize the application. Select a group from the left-hand panels and configure its settings.

Folders

This panel allows you to select the default storage folders for labels, forms, databases, and picture files. The default folder location is current user's Documents folder. These are also the default folders in which NiceLabel Automation searches for files if you provide file names without their full path. For more information about the file search order, see section Search order for the Requested Files.

Folder-related changes propagate to NiceLabel Automation Service within a minute. To apply changes immediately, restart the service.

NOTE

The settings that you apply here are saved into the profile of the currently logged-in user. If your NiceLabel Automation Service runs under a different user account, you have to log into Windows using that other account, and change the default label folder. You can also use Windows command-line utility RUNAS to run Automation Builder as that other user.

Language

Language panel allows selecting the NiceLabel Automation interface language. Select the appropriate language and click **OK**.

NOTE

The change is applied after you restart the application.

Global Variables

Global Variables panel defines which location with stored global variables should be used:

- **Use global variables stored on the server (NiceLabel Control Center):** Sets the global variable storage location to be on the NiceLabel Control Center.

  PRODUCT LEVEL INFO

  This option becomes available with LMS Pro or NiceLabelLMS Enterprise licenses.

- **Use global variables stored in a file (local or shared):** Sets the global variable storage location in a local or shared folder. Enter the exact path or click **Open** to locate the file.

Licensed Printers

> **NOTE**
> Keeping the information on the use of licensed printers is available with multi-user licenses.

Licensed Printers panel provides logged information about the number of printers that have been used in your printing environment.

Details on Licensed Printers group displays how many of the permitted printer seats are in use after printing on multiple printers.

- **Number of printers allowed by license:** Number of permitted printers to be used with the current NiceLabel 10 license.
- **Number of used printers in the last 7 days:** Number of printers that have been used with NiceLabel 10 during the last 7 days.

  > **TIP**
  > During a 7-day period, NiceLabel 10 license allows only the specified number of different printers to be used.

> **WARNING**
> Purchased license defines the number of allowed printers. After you exceed this number, a warning appears. After doubling the number of allowed printers, you can no longer print on additional printers.

See printing statuses in multiple columns:

- **Printer:** Name or model of the printer that was selected for the print job.

  > **NOTE**
  > If the connected printer is shared, only printer model is displayed.

- **Location:** Name of the computer from which the print job has been sent.
- **Port:** Port used by the printer.
- **Last Used:** Time passed since the last print job.
- **Reserved:** Prevents the printer from being removed after idling for more that 7 days.

NOTE

If a printer remains unused for more then 7 days, it is removed automatically unless the **Reserved** option is enabled.

Permissions panel allows you to lock printer usage on local workstation.

- **This workstation can only use reserved printers:** With this option enabled, only reserved printers are allowed for label editing and printing in NiceLabel 10.

  TIP

  Use this option to avoid exceeding the number of available licensed printer seats by printing on unwanted printers or print-to-file applications. Reserve dedicated thermal or laser labeling printers and limit printing only to them. This ensures continuous printing of labels with a multi-user license.

  This option can also be enabled using the `product.config` file:

  1. Navigate to the System folder.
     %PROGRAMDATA%\NiceLabel\NiceLabel 10
  2. Make a backup copy of the **`product.config`** file.
  3. Open **`product.config`** in a text editor. The file has an XML structure.
  4. Add the following lines:

     ```
     <Configuration>
       <Activation>
         <ReservePrinters>Example Printer Name</ReservePrinters>
       </Activation>
       <Common>
         <General>
           <ShowOnlyReservedPrinters>True</ShowOnlyReservedPrinters>
         </General>
       </Common>
     </Configuration>
     ```

  5. Save the file. The `Example Printer` is reserved.

Control Center

Control Center panel allows you to enable and configure the monitoring of events and print jobs. NiceLabel Control Center enables centralized event and print job reporting, and centralized storage of global variables.

NOTE

This tab becomes available with LMS Pro or LMS Enterprise licenses.

Address group defines which NiceLabel Control Center server should be used.

- **Control Center server address:** URL of the connected NiceLabel Control Center server. You can select from the list of automatically discovered servers on the network, or enter a server address manually.

NOTE

NiceLabel Control Center license keys on server and workstation must match to enable the connection.

Event Monitoring group defines what types of events should be logged by the connected NiceLabel Control Center.

- **Print Events:** Logs the print related events from the workstation.
- **Error Events:** Logs all reported errors.

NOTE

By default, Print Events and Error Events are logged by the NiceLabel Control Center.

- **Trigger Activity:** Logs all fired triggers.
- **Trigger Status Change Events:** Logs the trigger status changes which have been caused by the fired triggers.

Print Job Monitoring group enables you to log the completed and ongoing print jobs to NiceLabel Control Center.

- **Enable Print Job Logging to Server:** Activates print job logging.
- **Detailed printing control:** Enables monitoring of statuses that are reported by the connected printer.

NOTE

There are two requirements to make this option available:

- The printer must support bidirectional communication.
- NiceLabel printer driver must be used for printing.

Automation

These settings define the application's advanced features.

NOTE

The changes become active after you restart the application.

Service Communication

- **Service communication port:**Automation Manager controls the service using TCP/IP protocol on the selected port. If the default port is not suitable for you to be used on the computer, select another port number. Be careful not to select a port number that is already in use by another application.

Log

- **Clear log entries daily at:** Defines the start of the housekeeping process. Old events are erased from the log database.
- **Clear log entries when older than (days):** Specifies event retention in the log database. All events older than the specified number of days are erased from the database at each housekeeping event.
- **Log messages:** Specifies the scope of event recording in the log. During the development and testing phases, you might want to enable verbose logging. In this case, enable all messages to improve the tracing of trigger execution. Later on, while in production phase, minimize the amount of logging and enable logging of errors only.

Performance

-

  PRODUCT LEVEL INFO

  The functionality from this section is available in LMS Enterprise.

  Cache files from document storage and network shares: To improve the time-to-first label and to increase performance in general, NiceLabel Automation supports file caching. After you load the labels, images, and database data from network shares, all required files must be fetched before the printing process can begin.

  TIP

  If you enable local caching, the effect of network latency is reduced as label and picture files are loaded from your local disk.

  Automation service uses the following local folder to cache the remote files: `%PROGRAMDATA%\NiceLabel\NiceLabel 10\FileCache`.

  - **Refresh intervals (minutes):** Defines the time interval within which the files in the cache are synchronized with files in the original folder. This is the time limit for the system to use a version which may not be the latest.

- **Delete unused cached files after (days):** Defines the time interval after which all files are removed from cache.

> **NOTE**
> File caching supports label and picture file formats. After you enable file caching, restart Automation service to make the changes take effect.

- **Precache folders from document storage** allows you to locally cache the files from Control Center document storage folders on your computer. With enabled precaching, the content of local cache keeps synchronizing with the selected document storage folders.

> **NOTE**
> In comparison with caching, precaching reduces the printing time for your first printed label.

Add each document storage folder in a separate line.

```
/Labels/Folder1

/Labels/Folder2
```

> **NOTE**
> Your Automation Builder must be connected to the Control Center to enable the offline synchronization of cached files.

Cluster Support

> **PRODUCT LEVEL INFO**
> The functionality from this section is available in LMS Enterprise.

This setting enables support for high-availability (failover) type of cluster in NiceLabel Automation. Select the folder which both nodes in the cluster will use to share information about real-time trigger statuses.

Designer

Designer panel enables you to configure the opening behavior of NiceLabel 10.

- **Display each document in its own window:** If enabled, additionally opened documents appear in separate windows of NiceLabel 10. This applies to both – newly created and existing documents. If you decide to disable this option, additionally opened documents appear within the currently active instance of NiceLabel 10.
- **Printer Settings Source** allows you to choose the source of printer settings.
  - **Use printer settings from the printer driver:** Select this option if you prefer printing using the printer driver settings. This option allows you to standardize the printer settings in your working environment.
  - **Use custom printer settings saved in the label:** Each label may have its own printer settings defined and saved by the user. Select this option to use custom settings for your labels while printing.

1.6.6. About

About dialog provides information about your NiceLabel product license, enables license purchasing (when in trial mode) and activation, provides software details, and enables you to change the product level of NiceLabel 10.

License information group includes:

- **Trial mode duration:** Information about the remaining days for product evaluation. This segment is no longer visible after purchasing and activating the product license.
- **Purchase License:** Button directs you to the NiceLabel online store.
- **Activate license:** Button opens the NiceLabel 10 license activation dialog. See NiceLabel 10 installation guide for details about the license activation process. After activating the license, this button is renamed to **Deactivate License** – after clicking it and confirming the deactivation, your copy of NiceLabel 10 is no longer activated.
- **Change product level:** Opens the product level selection dialog. When in trial mode, you can choose and evaluate all product levels. With an activated license, you can change your product level only to lower levels.

NOTE

Product level changes take effect after restarting the application.

- **Upgrade license:** Opens the product level upgrade dialog. See NiceLabel 10 installation guide for details about the license upgrade process.

Software information group includes information about the installed software version and build number.

1.6.6.1. Losing your Label Cloud connection

If your Automation Manager is signed in to the Label Cloud, and you lose the internet connection, you must reestablish the connection in up to five days. Without reconnecting with your Label Cloud, Automation Manager closes automatically.

After losing the internet connection, and if your computer stays offline, a warning appears in 5 days. Automation Manager closes 5 minutes after you see the warning.

After you reestablish the internet connection, open Automation Builder or Automation Manager and sign in to Label Cloud. This makes your copy active again.

WARNING

Save your work to an offline location (your computer) to prevent losing any changes.

2. Understanding Filters

NiceLabel Automation uses filters to define structure of the data received by triggers. Each time a trigger receives data, one or many filters parse the received data. This process extracts relevant values for your configuration. Each filter includes rules to identify fields within received data.

NOTE

As a result, the filter provides a list of fields and their values (**`name:value`** pairs).

Filter Types

For more information, see sections Structured Text Filter, Unstructured Data Filter, XML filter, and JSON Filter.

Data Structure

Filter complexity depends on the data structure. Files with structured data, such as CSV or XML files, make data extraction easy. In this case, field names are already defined by the data. Extraction of **`name:value`** pairs is quick. In case of data without a clear structure, it takes more time to define the extraction rules. You might come across such data when exporting documents and reports from legacy systems, intercepted communication between devices, and captured print streams.

The filter defines a list of fields that are extracted from the incoming data once you run the filter.

NiceLabel Automation supports various types of input data that can all be parsed by one of the supported filter types. Make sure you select the correct filter to match the type of incoming data. For example, you would use **Structured Text filter** for incoming CSV data, **JSON filter** for incoming JSON data, and **XML filter** for incoming XML data. For any unstructured data, you would use **Unstructured Data filter**. For more information, see section Understanding Data Structures.

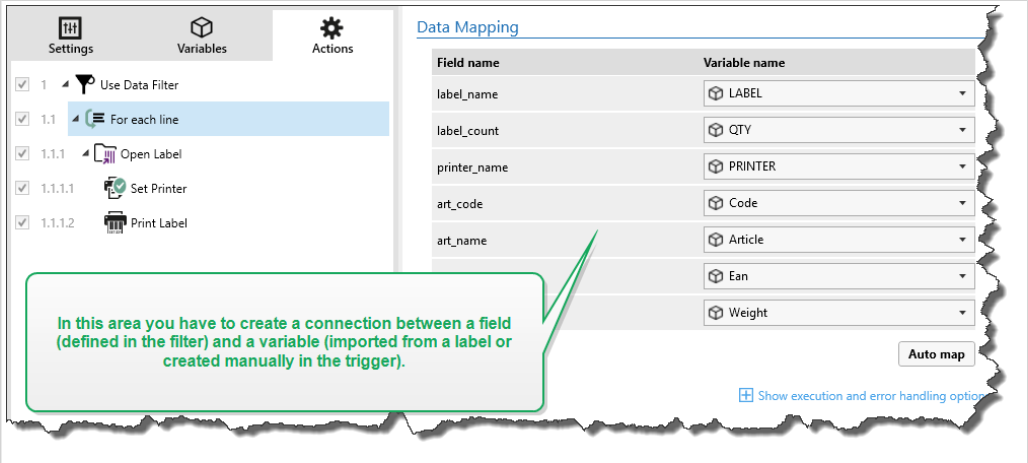
Extracting Data

Filter is a set of rules and doesn't do any extraction by itself. To run the filter, run the Use Data Filter action. This action executes filter rules against the data and extracts the values.

Each trigger type can execute as many Use Data Filter actions as necessary. If you receive compound input data that cannot be parsed by a single filer alone, define multiple filters and execute their rules using a sequence of Use Data Filter actions. Finally, use the extracted values from all actions on the same label.

Mapping Fields to Variables

To use the extracted values, you have to store them in variables. The Use Data Filter action does both – extracts values and saves them in variables. To configure this process, map each variable with the respective field. The value of the field is then saved in the mapped variable.



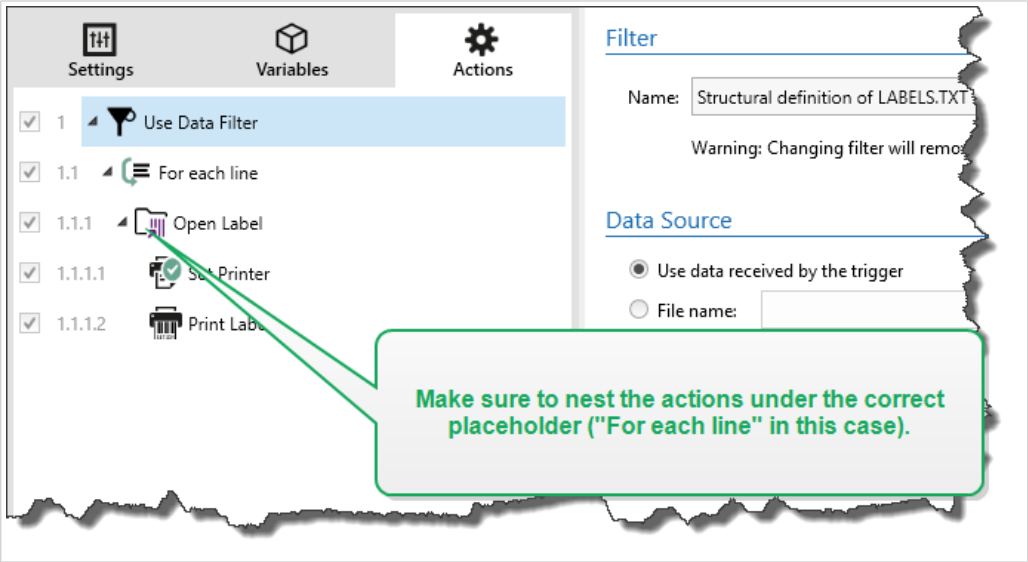
TIP

It's a good practice to define fields and variables with matching names. In this case, the auto-mapping feature links variables to the fields of the same names, eliminating the need for manual mapping.

Auto-mapping is available for all supported filter types. With auto-mapping enabled, the Use Data Filter action extracts values and automatically maps them to the variables with the same names as field names. For more information, see the section Enabling Dynamic Structure for Structured Text filter, Defining Assignment Area for Unstructured Data filter, and Defining Assignment Area for XML or JSON filters.

Running Actions with Extracted Data

Usually, you want to run actions using the extracted data, such as **Open Label**, **Print Label**, or some of the outbound connectivity actions. It is critically important that you nest your actions under the **Use Data Filter** action. This ensures that the nested actions are run for each data extraction.



Example

If you have a CSV file with 5 lines, the nested actions also run for 5 times; once for each data extraction. If the actions are not nested, they only execute once and only contain data from the last data extraction. As for the example above, the 5th CSV line would be printed leaving out the first four lines unprinted. If you use Sub Areas, make sure you nest your action under the correct placeholder.

2.1. Configuring Structured Text Filter

2.1.1. Structured Text Filter

To learn more about filters in general, see section Understanding Filters.

Use this filter whenever you receive a structured text file. These are the text files in which the fields are identified by one of the following methods:

- **Fields are delimited by a character:** Usual delimiter characters are comma or semicolon. CSV (comma separated values) is a typical example of a delimited file.
- **Fields contain fixed number of characters:** In other words, fields are defined by fixed-width columns.

For examples of structured text data, see section Text Database.

Defining Structure

To define the structure of a text file, you have the following options:

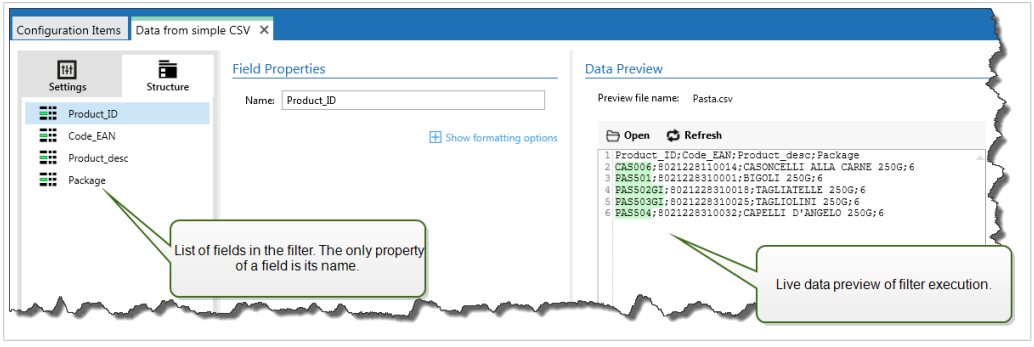
- **Importing structure using the Text File Wizard:** In this case, click the **Import Data Structure** button in the ribbon and follow the on-screen instructions. After you finish the wizard, the type of text database and all fields are defined. If the first line of data contains field names, the Wizard can import them. This is the recommended method, if the trigger always receives data whose structure does not change.
- **Manually defining the fields:** In this case, you have to manually define the type of data (delimited fields or fixed-width fields) and then define the field names. For more information, see section Defining Fields.
- **Dynamically read the fields:** In this case, the trigger might receive data that is structured in a different way. An example for this are new field names – dynamic structure eliminates the need to update the filter for each structural change. Support for dynamic structure automatically reads all data fields, no matter if new fields exist or if some of the old fields are missing. It maps them automatically with the variables using the same names. For more information, see section Enabling Dynamic Structure.

Data Preview section simplifies the configuration. In the preview pane, the result of a defined filter rule highlights the area along with every configuration change. Data Preview enables you to check what data is extracted with each rule.

2.1.2. Defining Fields

For structured text files, the definition of fields is very straightforward. There are two options:

- **Delimiter defines the fields:** In this case, you have a delimiting character, such as comma or semicolon separating the fields. You just have to define the field names in the same order as they appear in the data received by a trigger.
- **Fixed-width fields:** In this case, define the field names in the same order as they appear in the data received by a trigger, and define the number of characters the field would occupy. That many characters are going to be read from the data for this field.



Data Preview

This section provides preview of the field definition. If the defined item is selected, the preview highlights its placement in the preview data.

- **Preview file name:** Specifies the file that contains sample data that is going to be parsed by the filter. The preview file is copied from filter definition. If you change the preview file name, the new file name is saved.
- **Open:** Selects another file upon which you want to execute the filter rules.
- **Refresh:** Re-runs the filter rules upon the contents of the preview file name. Automation updates the Data Preview section with the result.

Formatting Options

This section defines string manipulation functions that apply on the selected variables or fields. You can select one or several functions. These functions are applied in the same order as selected in the user interface – from top to bottom.

- **Delete spaces at the beginning:** Deletes all space characters (decimal ASCII code 32) from the beginning of a string.
- **Delete spaces at the end:** Deletes all space characters (decimal ASCII value 32) from the end of a string.
- **Delete opening and closing character:** Deletes the first occurrence of the selected opening, and closing characters that are found in a string.

Example

If you use "{" for the opening character and "}" for the closing character, the input string `{{selection}}` converts to `{selection}`.

- **Search and replace:** Executes standard search and replace function upon the provided values for *find what* and *replace with*. You can also use regular expressions.

  NOTE

  There are several implementations of the regular expressions in use. uses the .NET Framework syntax for the regular expressions. For more information, see the Knowledge Base article.

- **Replace non printable characters with spaces:** Replaces all control characters in the string with space character (decimal ASCII code 32). The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Delete non printable characters:** Deletes all control characters in the string. The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Decode special characters:** Special characters (or control codes) are characters that are not available on the keyboard, such as Carriage Return or Line Feed. uses a notation to encode such characters in human-readable form, such as <CR> for Carriage Return and <LF> for Line Feed. For more information, see section Entering Special Characters (Control Codes).
  This option converts special characters from syntax into actual binary characters.

Example

When you receive the data sequence "<CR><LF>", uses it as a as plain string of 8 characters. Enable this option to interpret and use the received data as two binary characters **CR** (Carriage Return - ASCII code 13) and **LF** (Line Feed - ASCII code 10).

- **Search and delete everything before:** Finds the provided string and deletes all characters from the beginning of the data until the string. The found string itself can also be deleted.

- **Search and delete everything after:** Finds the provided string and deletes all characters from the string until the end of the data. The found string itself can also be deleted.
- **Change case**: Changes all characters in your strings to uppercase or lowercase.

2.1.3. Enabling Dynamic Structure

Structured Text filter has the ability to automatically identify fields and their values within the received data. This eliminates the need for manual *variable-to-field* mapping.

Dynamic structure functionality is helpful if the trigger receives data with changing structure. In such cases, main data structure remains unchanged (e.g., fields are delimited by a comma), or retains the same structure, but **the order** and/or **the number** of fields changes. There might be new fields, or some of the old fields could no longer be available. Because of enabled **Dynamic structure**, filter automatically identifies structure of the received file. At the same time, filter reads field names and values (**`name:value`** pairs) from the data. This eliminates the need for manual mapping of fields to variables.

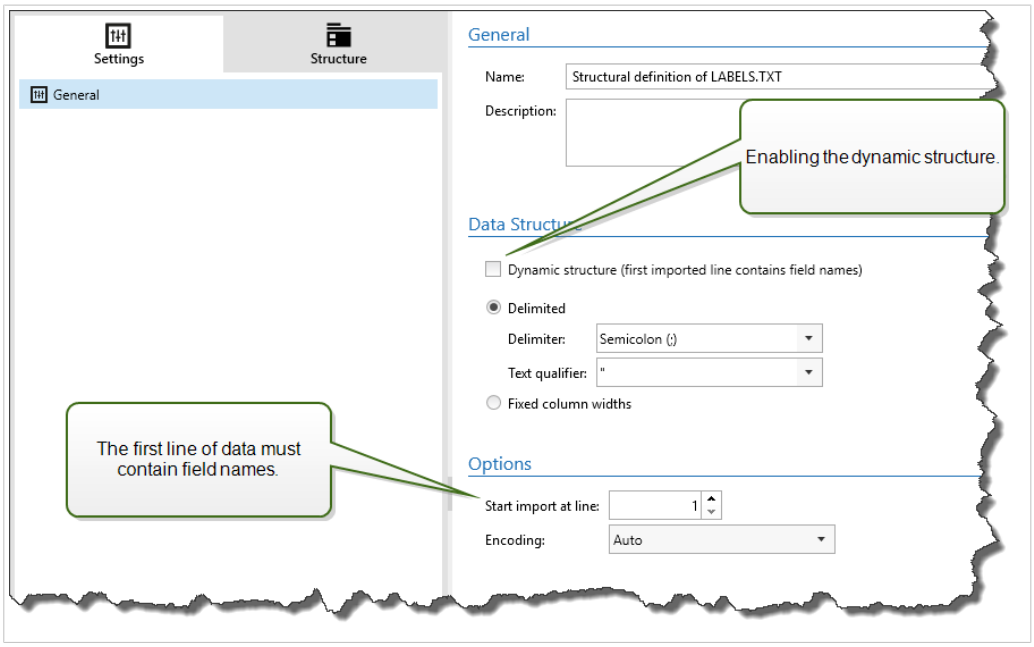
Use Data Filter action does not offer any mapping possibilities, because it performs the mapping dynamically. You don't even have to define label variables in the trigger configuration. The action assigns field values to the label variables of the same name without requiring the variables to be imported from the label. However, this rule applies to the Print Label action alone. If you want to use the field values in any other action, you have to define variables in the trigger, while still keeping the automatic *variable-to-field* mapping.

NOTE

There is no error if the field available in the input data doesn't have a matching label variable. NiceLabel Automation silently ignores the missing variables.

Configuring the dynamic structure

To configure the dynamic structure, enable the **Dynamic structure** option in Structured Text filter properties.



- The first line of data must contain field names.
- The line that you select for **Start import at line** must be the line with the field names (usually the first line in data).
- Data structure must be delimited.
- You can format the data, if necessary.

Formatting Options

This section defines string manipulation functions that apply on the selected variables or fields. You can select one or several functions. These functions are applied in the same order as selected in the user interface – from top to bottom.

- **Delete spaces at the beginning:** Deletes all space characters (decimal ASCII code 32) from the beginning of a string.
- **Delete spaces at the end:** Deletes all space characters (decimal ASCII value 32) from the end of a string.
- **Delete opening and closing character:** Deletes the first occurrence of the selected opening, and closing characters that are found in a string.

Example

If you use "{" for the opening character and "}" for the closing character, the input string **`{{selection}}`** converts to **`{selection}`**.

- **Search and replace:** Executes standard search and replace function upon the provided values for *find what* and *replace with*. You can also use regular expressions.

  > **NOTE**
  > There are several implementations of the regular expressions in use. uses the .NET Framework syntax for the regular expressions. For more information, see the Knowledge Base article.

- **Replace non printable characters with spaces:** Replaces all control characters in the string with space character (decimal ASCII code 32). The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Delete non printable characters:** Deletes all control characters in the string. The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Decode special characters:** Special characters (or control codes) are characters that are not available on the keyboard, such as Carriage Return or Line Feed. uses a notation to encode such characters in human-readable form, such as <CR> for Carriage Return and <LF> for Line Feed. For more information, see section Entering Special Characters (Control Codes).
  This option converts special characters from syntax into actual binary characters.

Example

When you receive the data sequence "<CR><LF>", uses it as a as plain string of 8 characters. Enable this option to interpret and use the received data as two binary characters **CR** (Carriage Return - ASCII code 13) and **LF** (Line Feed - ASCII code 10).

- **Search and delete everything before:** Finds the provided string and deletes all characters from the beginning of the data until the string. The found string itself can also be deleted.
- **Search and delete everything after:** Finds the provided string and deletes all characters from the string until the end of the data. The found string itself can also be deleted.
- **Change case**: Changes all characters in your strings to uppercase or lowercase.

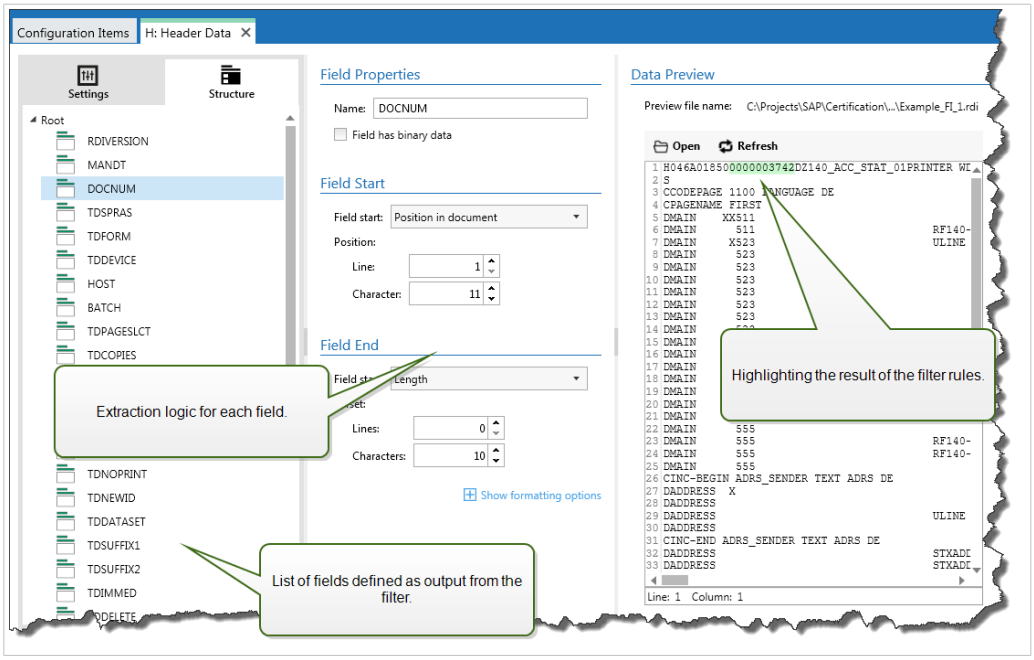
2.2. Configuring Unstructured Data Filter

2.2.1. Unstructured Data Filter

To learn more about filters in general, see section Understanding Filters.

Use this filter whenever a trigger receives non-structured data, such as documents and reports exported from legacy system, intercepted communication between devices, and captured print stream. The filter allows you to extract individual fields, fields in repeatable sub areas, and even **`name-value`** pairs.

For examples of the structured text data, see sections Legacy Data, Compound CSV and Binary Files.



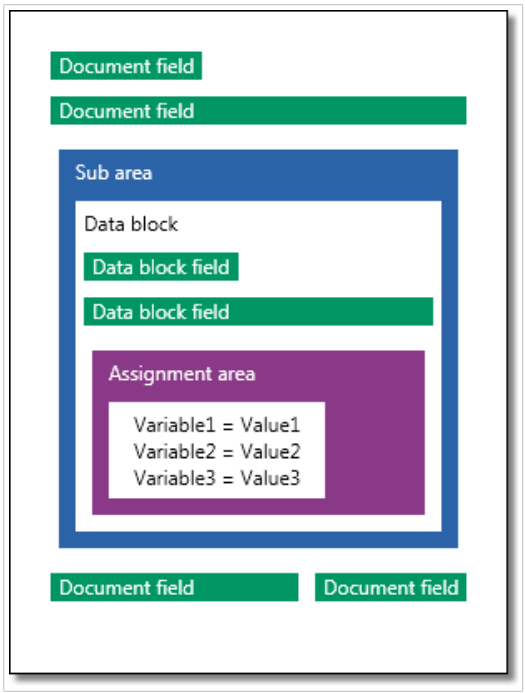
Defining Structure

The items you can use to configure the filter:

- **Field:** Specifies the location of field data between field-start and field-end locations. There are various options to define the field location, from hard-coding the position to enabling relative placements. You must map the defined fields to respective variables in the Use Data Filter action. For more information, see section Defining Fields.
- **Sub area:** Specifies the location of repeatable data. Each sub area defines at least one data block, which in turn contains data for labels. There can be sub areas defined within sub areas, which allows you to define complex structures. You can define fields within each data block. You must map the defined fields to respective variables in the action. For each sub area, Automation defines a new level of placeholder within the Use Data Filter action, so you can map variables to fields of that level. For more information, see section Defining Sub Areas.
- **Assignment area:** Specifies the location of repeatable data containing the **`name-value`** pairs. Automation reads field names and their values simultaneously. Automation also automatically performs mapping to variables. Use this feature to accommodate filter to the changing input data, eliminating the maintenance time. You can define the assignment area at the root level of the document, or inside the sub area. For more information, see section Defining Assignment Areas.

Data Preview section simplifies the configuration. The result of a defined filter rule gets highlighted in the preview area along with every configuration change. You can observe which data items would be extracted with each rule.

The fields can be defined at the root level as document fields. The fields can be defined inside a data block. The **`name-value`** pairs can be defined inside assignment area.



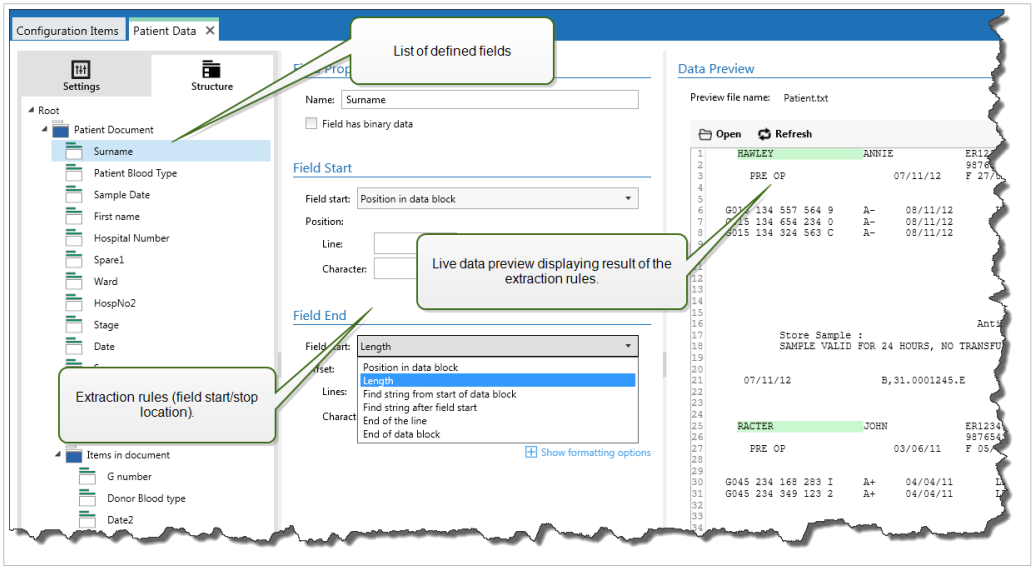
General

This section defines general properties of the unstructured data filter.

- **Name:** Specifies the filter name. Use a descriptive name that identifies the role of the filter in a configuration. You can change it anytime.
- **Description:** Allows you to describe the purpose of this filter. You can use it to write a short explanation about what does the filter do.
- **Encoding:** Specifies the encoding of the data this filter works with.
- **Ignore empty lines in data blocks:** Specifies not to raise errors if the filter extracts empty field values from the data blocks.

2.2.2. Defining Fields

After you define a field, you have to define its name and rules for extracting field values from the data. When the filter executes, the extraction rules apply to the input data and assign result to the field.



Field Properties

- **Name:** Specifies the unique name of the field.
- **Field has binary data:** Specifies that the field contains binary data. Don't enable this option unless you really expect to receive binary data.

Field Start

- **Position in document:** Hard-coded position in the data determines the start/end point. The coordinate origin is upper left corner. The character in the defined position is included in the extracted data.
- **End of document:** The start/end point is at the end of the document. You can also define an offset from the end for a specified number of lines and/or characters.
- **Find string from start of document:** Position of searched-for string defines the start/end point. After Automation finds the required string, the next character determines the start/end point. The searched-for string is not included in the extracted data. The default search is case sensitive.
    - **Start search from absolute position:** You can fine-tune searching by changing the start position from data-start (position 1,1) to an offset. Use this feature to skip searching at the beginning of data.
    - **Occurrence:** Specifies which occurrence of the search string should be matched. Use this option if you don't wait to set start/stop position after the first found string.
    - **Offset from string:** Specifies the positive or negative offset after the searched-for string.

    ### Example

    You would define the offset to include the searched-for string in the extracted data.

Field End

- **Position in document:** Hard-coded position in the data determines the start/end point. The coordinate origin is upper left corner. The character in the defined position is included in the extracted data.
- **End of document:** The start/end point is at the end of the document. You can also define an offset from the end for a specified number of lines and/or characters.
- **Find string from start of document:** Position of searched-for string defines the start/end point. After Automation finds the required string, the next character determines the start/end point. The searched-for string is not included in the extracted data. The default search is case sensitive.
    - **Start search from absolute position:** You can fine-tune searching by changing the start position from data-start (position 1,1) to an offset. Use this feature to skip searching at the beginning of data.
    - **Occurrence:** Specifies which occurrence of the search string should be matched. Use this option if you don't wait to set start/stop position after the first found string.
    - **Offset from string:** Specifies the positive or negative offset after the searched-for string.

    ### Example

    You would define the offset to include the searched-for string in the extracted data.

- **Find string after field start:** The start/stop end point is defined by the position of the searched-for-string as in the option **Find string from start of document**, but the search starts after the start position of the field/area, not at the beginning of the data.
- **Length:** Specifies the length of the data in lines and/or characters. The specified number of lines and/or characters will be extracted from the start position.
- **End of the line:** Sets the configuration to extract the data from the start position until the end of the same line. You can define a negative offset from the end of the line.

Formatting Options

This section defines string manipulation functions that apply on the selected variables or fields. You can select one or several functions. These functions are applied in the same order as selected in the user interface – from top to bottom.

- **Delete spaces at the beginning:** Deletes all space characters (decimal ASCII code 32) from the beginning of a string.
- **Delete spaces at the end:** Deletes all space characters (decimal ASCII value 32) from the end of a string.
- **Delete opening and closing character:** Deletes the first occurrence of the selected opening, and closing characters that are found in a string.

Example

If you use "{" for the opening character and "}" for the closing character, the input string `{{selection}}` converts to `{selection}`.

- **Search and replace:** Executes standard search and replace function upon the provided values for *find what* and *replace with*. You can also use regular expressions.

  NOTE

  There are several implementations of the regular expressions in use. uses the .NET Framework syntax for the regular expressions. For more information, see the Knowledge Base article.

- **Replace non printable characters with spaces:** Replaces all control characters in the string with space character (decimal ASCII code 32). The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Delete non printable characters:** Deletes all control characters in the string. The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Decode special characters:** Special characters (or control codes) are characters that are not available on the keyboard, such as Carriage Return or Line Feed. uses a notation to encode such characters in human-readable form, such as <CR> for Carriage Return and <LF> for Line Feed. For more information, see section Entering Special Characters (Control Codes).
  This option converts special characters from syntax into actual binary characters.

Example

When you receive the data sequence "<CR><LF>", uses it as a as plain string of 8 characters. Enable this option to interpret and use the received data as two binary characters **CR** (Carriage Return - ASCII code 13) and **LF** (Line Feed - ASCII code 10).

- **Search and delete everything before:** Finds the provided string and deletes all characters from the beginning of the data until the string. The found string itself can also be deleted.

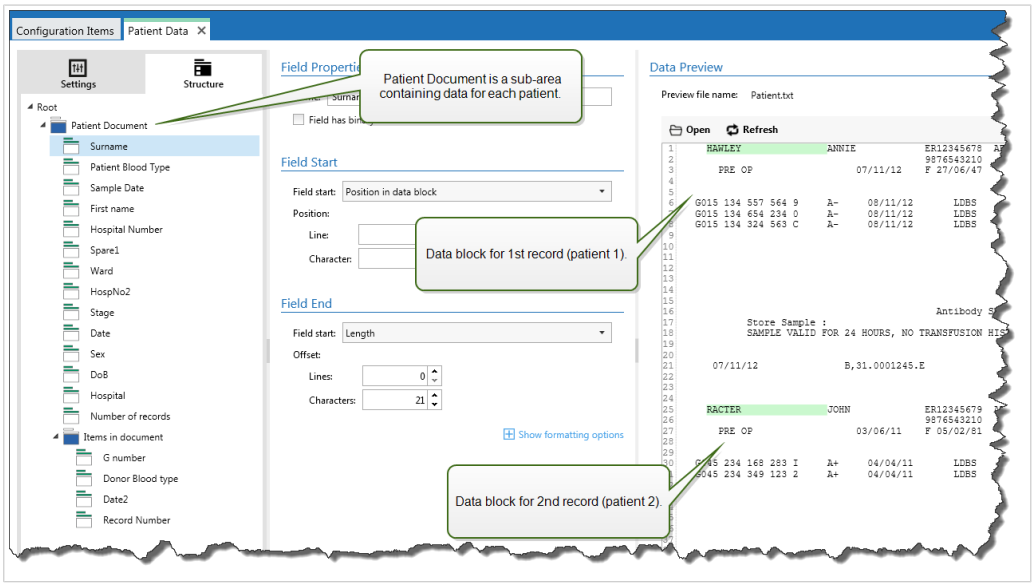
- **Search and delete everything after:** Finds the provided string and deletes all characters from the string until the end of the data. The found string itself can also be deleted.
- **Change case**: Changes all characters in your strings to uppercase or lowercase.

2.2.3. Defining Sub Areas

Sub area is a section of data that includes several blocks of data identified by the same extraction rule. Each data block provides data for a single label.

All data blocks must be identified by the same configuration rule. Each data block can contain another sub area. You can define an unlimited number of nested sub areas within parent sub areas.

If the filter contains definition of a sub area, the Use Data Filter action displays sub areas with nested placeholders. All actions nested below such placeholder execute only for data blocks at this level. You can print different labels with data from different sub areas.



Configuring Sub Area

The sub area is defined with similar rules as individual fields. Each sub area is defined by the following parameters.

- **Sub Area Name:** Specifies the name of the sub area.
- **Data Blocks:** Specifies how to identify data blocks within the sub area. Each sub area contains at least one data block. Each data block provides data for a single label.
    - **Each block contains fixed number of lines:** Specifies that each data block in a sub area contains the provided fixed number of lines. Use this option if you know that each data block contains exactly the same number of lines.
    - **Blocks start with a string:** Specifies that data blocks begin with the provided string. All content between the two provided strings belongs to a separate data block. The content between last string and the end of the data identifies the last data block.
    - **Blocks end with a string:** Specifies that data blocks end with the provided string. All content between the two provided strings belongs to a separate data block. The content between the beginning of data and the first string identifies the first data block.
    - **Blocks are separated by a string:** Specifies that data blocks are separated with the selected string of characters. All content between the two selected strings belongs to a separate data block.
- **Beginning of First Data Block:** Specifies the starting position of the first data block. At the same time, it defines the starting position of the sub area. Usually, the starting position is also the beginning of the received data. The configuration parameters are the same as for defining fields. For more information, see section Defining Fields.
- **End of Last Data Block:** Specifies the ending position of the last data block. At the same time, it defines the ending position of the sub area. Usually, ending position is at the end of the received data. The configuration parameters are the same as for defining fields. For more information, see section Defining Fields.

Configuring Fields Inside Sub Area

Fields inside the sub area are configured using the same parameters as fields defined at root level. For more information, see section Defining Fields.

NOTE

The field lines numbers refer to the position within data block, not position within the input data.

Data Preview

This section provides preview of the field definition. If the defined item is selected, the preview highlights its placement in the preview data.

- **Preview file name:** Specifies the file that contains sample data that is going to be parsed by the filter. The preview file is copied from filter definition. If you change the preview file name, the new file name is saved.
- **Open:** Selects another file upon which you want to execute the filter rules.
- **Refresh:** Re-runs the filter rules upon the contents of the preview file name. Automation updates the Data Preview section with the result.

2.2.4. Defining Assignment Areas

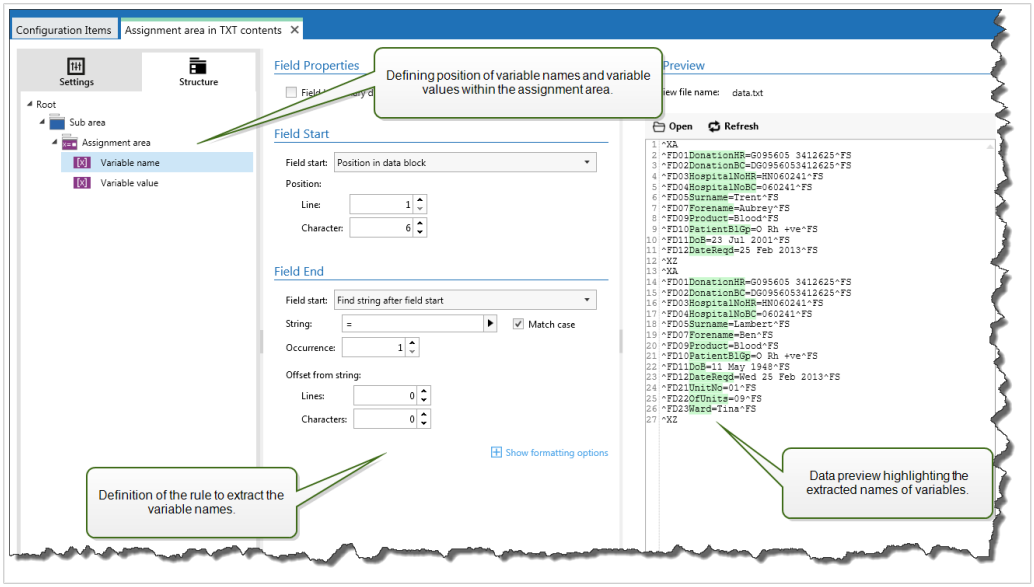
Unstructured Data filter automatically identifies fields and their values in the received data. This makes manual *variable to field* mapping unnecessary.

Dynamic structure functionality is helpful if the trigger receives data with changing structure. In such cases, main data structure remains unchanged (e.g., fields are delimited by a comma), or retains the same structure, but **the order** and/or **the number** of fields changes. There might be new fields, or some of the old fields could no longer be available. Because of enabled **Dynamic structure**, filter automatically identifies structure of the received file. At the same time, filter reads field names and values (**`name:value`** pairs) from the data. This eliminates the need for manual mapping of fields to variables.

Use Data Filter action does not offer any mapping possibilities, because it performs the mapping dynamically. You don't even have to define label variables in the trigger configuration. The action assigns field values to the label variables of the same name without requiring the variables to be imported from the label. However, this rule applies to the Print Label action alone. If you want to use the field values in any other action, you have to define variables in the trigger, while still keeping the automatic *variable-to-field* mapping.

NOTE

There is no error if the field available in the input data doesn't have a matching label variable. NiceLabel Automation silently ignores the missing variables.



Configuring Assignment Area

The assignment area is configured using the same procedure as sub area. For more information, see section Defining Sub Areas. The assignment area can be defined on the root data level, appearing just once. Or it can be configured inside a sub area, so it executes for each data block in the sub area.

Configuring Fields in Assignment Area

When you create an assignment area, the filter automatically defines two placeholders. These two placeholders define the **`name:value`** pair.

- **Variable name:** Specifies the field whose content is taken as variable name (**`name`** component in a pair). Configure the field using the same procedure as for document fields. For more information, see section Defining Fields.
- **Variable value:** Specifies the field whose contents is taken as variable value (**`value`** component in a pair). Configure the field using the same procedure as for document fields. For more information, see section Defining Fields.

Example

The area between ^XA and ^XZ is assignment area. Every line in assignment area provides a **`name:value`** pair. **Name** is defined as the value between 6th character in the line and equal character. **Value** is defined as the value between "equals" character and end of the line with negative offset of three characters.

```
^XA
^FD01DonationHR=G095605 3412625^FS
^FD02DonationBC=DG0956053412625^FS
^FD03HospitalNoHR=HN060241^FS
^FD04HospitalNoBC=060241^FS
^FD05Surname=Hawley^FS
^FD07Forename=Annie^FS
^FD09Product=Blood^FS
^FD10PatientBlGp=O Rh +ve^FS
^FD11DoB=27 June 1947^FS
^FD12DateReqd=25 Dec 2012^FS
^XZ
```

For more information, see section Examples.

2.3. Configuring XML filter

2.3.1. XML Filter

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

To learn more about filters in general, see section Understanding Filters.

Use this filter whenever a trigger receives XML-encoded data. The filter allows you to extract individual fields, fields in repeatable sub areas, and even **`name-value`** pairs. XML structure defines elements and sub elements, attributes and their values, and textual values (element values).

While you can define the structure of the XML file by yourself, NiceLabel recommends you to import the structure from the existing sample XML file. Click **Import Data Structure** button on the ribbon. After you import the XML structure, the Data Preview section displays the XML contents. It also highlights the elements and attributes that you define as output fields.

For XML data examples, see section XML Data.

Defining Structure

To use the XML items, configure them as:

- **Variable value:** Specifies that you want to use the selected item as a field and that you will map its value to respective variables in Use Data Filter action. For more information, see section Defining XML Fields.
    - **Optional element:** Specifies that the element is not mandatory. This corresponds with XML schema (XSD file) attribute `minOccurs=0`. The variable mapped to such field will have an empty value if the element does not appear in the XML.
- **Data block:** Specifies that the selected element occurs multiple times and provides data for a single label. Data block can be defined as repeatable area, assignment area, or both.
    - **Repeatable area:** Specifies that you want to extract values from all repeatable data blocks, not just from the first one. You can define fields within each data block. Map the defined fields to respective variables in Use Data Filter action. For more information, see section Defining Repeatable Elements.
    - **Assignment area:** Specifies that the data block contains `name-value` pairs. Field names and their values are read simultaneously. Mapping to variables is done automatically. Use this feature to accommodate the filter to changing input data, eliminating the maintenance time. For more information, see section Defining XML Assignment Area.

The Data Preview section simplifies the configuration. The result of a defined filter rule is highlighted in the preview area.

To change the previewed XML data, click **Open** and browse for a new sample XML file.

2.3.2. Defining XML Fields

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

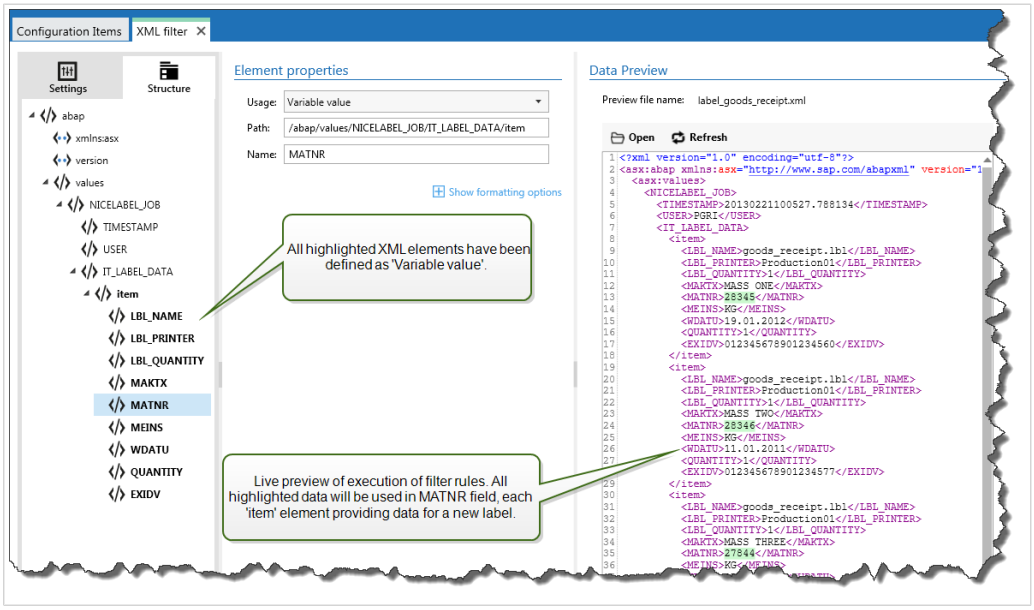
If you define XML fields, you make values of selected items automatically available. Filter definition makes such fields available for mapping to variables in actions. This allows you to extract values of elements or attributes.

To define an item value as field, do the following:

1. Select the element or attribute in the structure list.
2. For **Usage**, select **Variable value**.
3. The item is displayed on the structure list with bold letters, indicating that it is in use.

4. The element or attribute name is used as output field name.

5. The Data Preview section highlights the value of the selected item.



Formatting Options

This section defines string manipulation functions that apply on the selected variables or fields. You can select one or several functions. These functions are applied in the same order as selected in the user interface – from top to bottom.

- **Delete spaces at the beginning:** Deletes all space characters (decimal ASCII code 32) from the beginning of a string.
- **Delete spaces at the end:** Deletes all space characters (decimal ASCII value 32) from the end of a string.
- **Delete opening and closing character:** Deletes the first occurrence of the selected opening, and closing characters that are found in a string.

Example

If you use "{" for the opening character and "}" for the closing character, the input string `{{selection}}` converts to `{selection}`.

- **Search and replace:** Executes standard search and replace function upon the provided values for *find what* and *replace with*. You can also use regular expressions.

  NOTE

  There are several implementations of the regular expressions in use. uses the .NET Framework syntax for the regular expressions. For more information, see the Knowledge Base article.

- **Replace non printable characters with spaces:** Replaces all control characters in the string with space character (decimal ASCII code 32). The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Delete non printable characters:** Deletes all control characters in the string. The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Decode special characters:** Special characters (or control codes) are characters that are not available on the keyboard, such as Carriage Return or Line Feed. uses a notation to encode such characters in human-readable form, such as <CR> for Carriage Return and <LF> for Line Feed. For more information, see section Entering Special Characters (Control Codes).
  This option converts special characters from syntax into actual binary characters.

Example

When you receive the data sequence "<CR><LF>", uses it as a as plain string of 8 characters. Enable this option to interpret and use the received data as two binary characters **CR** (Carriage Return - ASCII code 13) and **LF** (Line Feed - ASCII code 10).

- **Search and delete everything before:** Finds the provided string and deletes all characters from the beginning of the data until the string. The found string itself can also be deleted.

- **Search and delete everything after:** Finds the provided string and deletes all characters from the string until the end of the data. The found string itself can also be deleted.
- **Change case**: Changes all characters in your strings to uppercase or lowercase.

Data Preview

This section provides preview of the field definition. If the defined item is selected, the preview highlights its placement in the preview data.

- **Preview file name:** Specifies the file that contains sample data that is going to be parsed by the filter. The preview file is copied from filter definition. If you change the preview file name, the new file name is saved.
- **Open:** Selects another file upon which you want to execute the filter rules.
- **Refresh:** Re-runs the filter rules upon the contents of the preview file name. Automation updates the Data Preview section with the result.

2.3.3. Defining Repeatable Elements in XML Filter

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

If an XML element occurs in XML data for multiple times, it is a repeatable element. Usually, a repeatable element contains data for a single label. To indicate that you want to use data from all repeatable elements, and not just from the first one, define the element as a **Data block** and enable the **Repeatable element** option. If the filter contains definition of elements defined as data block / repeatable element, the Use Data Filter action displays repeatable elements with nested placeholders. All actions nested below such a placeholder execute only for data blocks at this level.

Example

The **<item>** element is defined as **Data block** and **Repeatable element**. This instructs the filter to extract all occurrences of the **<item>** element, not just the first one. In this case, the **<item>** should be defined as the sub-level in **Use Data Filter** action. You must nest the actions Open Label and Print Label under this sub-level placeholder, so they are going to be looped for as many times as there are occurrences of the **<item>** element. As shown in the example below, for three times.

```
<?xml ver sion="1.0" encoding="utf-8"?>
<asx:abap xmlns:asx="http://www.sap.com/abapxml" ver sion="1.0">
 <asx:values>
  <NICELABEL_JOB>
   <T IMEST AMP>20130221100527.788134</T IMEST AMP>
   <USER>PGRI</USER>
   <IT _LABEL_DAT A>
   <item>
      <LBL_NAME>goods_r eceipt.nlbl</LBL_NAME>
      <LBL_PRINT ER>Pr oduction01</LBL_PRINT ER>
      <LBL_QUANT IT Y>1</LBL_QUANT IT Y>
      <MAKT X>MASS ONE</MAKT X>
      <MAT NR>28345</MAT NR>
      <MEINS>KG</MEINS>
      <WDAT U>19.01.2012</WDAT U>
      <QUANT IT Y>1</QUANT IT Y>
      <EXIDV>012345678901234560</EXIDV>
   </item>

   <item>
      <LBL_NAME>goods_r eceipt.nlbl</LBL_NAME>
      <LBL_PRINT ER>Pr oduction01</LBL_PRINT ER>
      <LBL_QUANT IT Y>1</LBL_QUANT IT Y>
      <MAKT X>MASS T WO</MAKT X>
      <MAT NR>28346</MAT NR>
      <MEINS>KG</MEINS>
      <WDAT U>11.01.2011</WDAT U>
      <QUANT IT Y>1</QUANT IT Y>
      <EXIDV>012345678901234577</EXIDV>
   </item>

   <item>
      <LBL_NAME>goods_r eceipt.nlbl</LBL_NAME>
      <LBL_PRINT ER>Pr oduction01</LBL_PRINT ER>
      <LBL_QUANT IT Y>1</LBL_QUANT IT Y>
      <MAKT X>MASS T HREE</MAKT X>
      <MAT NR>27844</MAT NR>
      <MEINS>KG</MEINS>
      <WDAT U>07.03.2009</WDAT U>
```

```
      <QUANT IT Y>1</QUANT IT Y>
      <EXIDV>012345678901234584</EXIDV>
    </item>

  </IT _LABEL_DAT A>
 </NICELABEL_JOB>
 </asx:values>
</asx:abap>
```

2.3.4. Defining XML Assignment Area

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

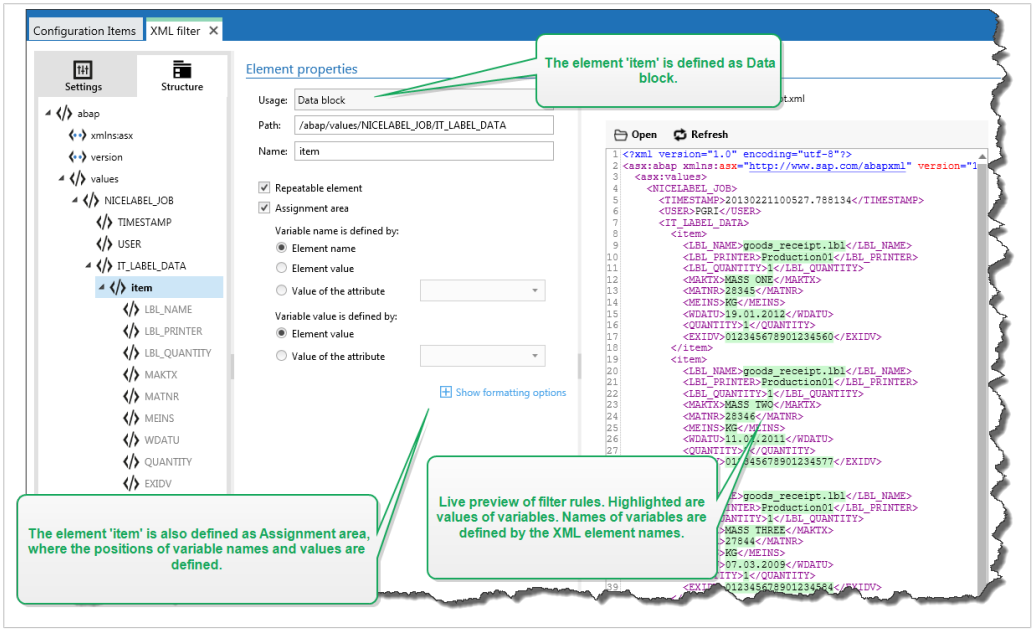
XML filter automatically identifies fields and their values in the received data. This eliminates the need for manual *variable-to-field* mapping.

Dynamic structure functionality is helpful if the trigger receives data with changing structure. In such cases, main data structure remains unchanged (e.g., fields are delimited by a comma), or retains the same structure, but **the order** and/or **the number** of fields changes. There might be new fields, or some of the old fields could no longer be available. Because of enabled **Dynamic structure**, filter automatically identifies structure of the received file. At the same time, filter reads field names and values (**`name:value`** pairs) from the data. This eliminates the need for manual mapping of fields to variables.

Use Data Filter action does not offer any mapping possibilities, because it performs the mapping dynamically. You don't even have to define label variables in the trigger configuration. The action assigns field values to the label variables of the same name without requiring the variables to be imported from the label. However, this rule applies to the Print Label action alone. If you want to use the field values in any other action, you have to define variables in the trigger, while still keeping the automatic *variable-to-field* mapping.

NOTE

There is no error if the field available in the input data doesn't have a matching label variable. NiceLabel Automation silently ignores the missing variables.



Configuring XML Assignment Area

When you configure the Data Block as assignment area, two placeholders appear under this element's definition. You have to define how the field name and value are defined, so the filter can extract the **`name-value`** pair.

- **Variable name:** Specifies the item that contains field name. The name can be defined by element name, selected attribute value, or element value. To enable automatic mapping, label variable must have the same name.
- **Variable value:** Specifies the item that contains field value. The name can be defined by the element name, selected attribute value, or element value.

WARNING

The XML element containing **`name:value`** pairs cannot be set as a root element – this element must be set as at least a second level element. As shown in the XML example below, the element **`<label>`** is the second level element and can contain the **`name:value`** pairs.

Formatting Options

This section defines string manipulation functions that apply on the selected variables or fields. You can select one or several functions. These functions are applied in the same order as selected in the user interface – from top to bottom.

- **Delete spaces at the beginning:** Deletes all space characters (decimal ASCII code 32) from the beginning of a string.
- **Delete spaces at the end:** Deletes all space characters (decimal ASCII value 32) from the end of a string.
- **Delete opening and closing character:** Deletes the first occurrence of the selected opening, and closing characters that are found in a string.

Example

If you use "{" for the opening character and "}" for the closing character, the input string `{{selection}}` converts to `{selection}`.

- **Search and replace:** Executes standard search and replace function upon the provided values for *find what* and *replace with*. You can also use regular expressions.

  NOTE

  There are several implementations of the regular expressions in use. uses the .NET Framework syntax for the regular expressions. For more information, see the Knowledge Base article.

- **Replace non printable characters with spaces:** Replaces all control characters in the string with space character (decimal ASCII code 32). The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Delete non printable characters:** Deletes all control characters in the string. The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Decode special characters:** Special characters (or control codes) are characters that are not available on the keyboard, such as Carriage Return or Line Feed. uses a notation to encode such characters in human-readable form, such as <CR> for Carriage Return and <LF> for Line Feed. For more information, see section Entering Special Characters (Control Codes).
  This option converts special characters from syntax into actual binary characters.

Example

When you receive the data sequence "<CR><LF>", uses it as a as plain string of 8 characters. Enable this option to interpret and use the received data as two binary characters **CR** (Carriage Return - ASCII code 13) and **LF** (Line Feed - ASCII code 10).

- **Search and delete everything before:** Finds the provided string and deletes all characters from the beginning of the data until the string. The found string itself can also be deleted.

- **Search and delete everything after:** Finds the provided string and deletes all characters from the string until the end of the data. The found string itself can also be deleted.
- **Change case**: Changes all characters in your strings to uppercase or lowercase.

Example

The **<label>** element is defined as data block and assignment area. The **variable name** is defined by value of the attribute name, **the variable value** is defined by element text.

```
<?xml version="1.0" standalone="no"?>
<labels _FORMAT="case.nlbl" _PRINTERNAME="Production01" _QUANTITY="1">
    <label>
        <variable name="CASEID">0000000123</variable>
        <variable name="CARTONTYPE"/>
        <variable name="ORDERKEY">0000000534</variable>
        <variable name="BUYERPO"/>
        <variable name="ROUTE"> </variable>
        <variable name="CONTAINERDETAILID">0000004212</variable>
        <variable name="SERIALREFERENCE">0</variable>
        <variable name="FILTERVALUE">0</variable>
        <variable name="INDICATORDIGIT">0</variable>
        <variable name="DATE">11/19/2012 10:59:03</variable>
    </label>
</labels>
```

For more information, see section Examples.

2.4. Configuring JSON filter

2.4.1. JSON Filter

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

To learn more about filters in general, see section Understanding Filters.

Use the JSON filter if your trigger receives JSON-encoded data. JSON filter allows you to use variables and values from your JSON file. The filter supports data extraction from JSON arrays.

NOTE

Automation allows you to use all JSON data types. Read more about the available JSON data types here.

While you can define the JSON file structure manually, NiceLabel recommends you to import the structure from the received JSON files.

To import the JSON file structure:

1. Go to **Data Filters** and **Edit** your JSON filter.
2. Click **Structure** > **Import Data Structure**. Navigate to your JSON file and click **Open**.
   After you import JSON files, the **Data Preview** section displays the JSON contents. The Data Preview also highlights the elements that you define as output fields.

For JSON data examples, see section JSON Data.

Defining Structure

To use the JSON items, configure them as:

- **Variable value:** Specifies that you want to use the selected item as a field. When building the configuration, you manually map values with respective variables in the Use Data Filter action. For more information, see section Defining JSON Fields.
  - **Optional element**: Specifies that the element is not mandatory. The variable mapped to such field will have an empty value if the element does not appear in the JSON file.
- **Data block:** Specifies that the included sub-items occur multiple times and provide data for your labels. Data block can be defined as repeatable area, assignment area, or both. In terms of JSON, the Data block works as an array.
  - **Repeatable area:** Specifies that you want to extract values from all repeatable data blocks, not just from the first one. You can define fields within each data block. Manually map the defined fields to the respective variables in the Use Data Filter action. For more information, see section Defining Repeatable Elements in JSON Filter.
  - **Assignment area:** Automatically creates the variables and assigns them relevant values. Field names and their values are read simultaneously. Mapping to variables is done automatically. Use this feature to accommodate the filter to changing input data, eliminating the maintenance time. For more information, see section Defining JSON Assignment Area.

The **Data Preview** section simplifies the configuration. The result of a defined filter rule is highlighted in the preview area.

To change the previewed JSON data, click **Open** and browse for a new sample JSON file.

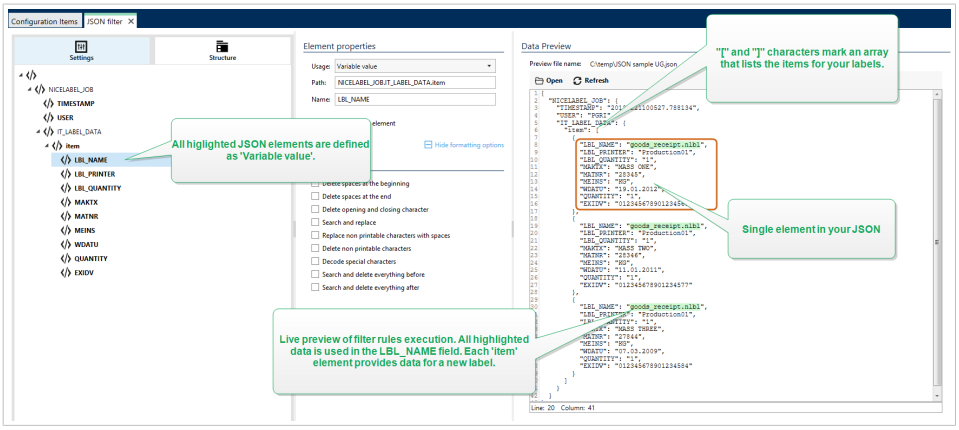
2.4.2. Defining JSON Fields

> **PRODUCT LEVEL INFO**
> This functionality is available in **LMS Enterprise** and **LMS Pro**.

If you define JSON fields, you make values of the selected items automatically available. Your filter definition makes such fields available for mapping to variables in actions. This allows you to extract the element values.

To define JSON fields:

1. Select your element and set its **Usage** to **Variable value**.
2. The element is displayed on the structure list with bold letters, indicating that it is in use.
3. The element is used as output field name.
4. The Data Preview section highlights the values of the selected element.



Formatting Options

This section defines string manipulation functions that apply on the selected variables or fields. You can select one or several functions. These functions are applied in the same order as selected in the user interface – from top to bottom.

- **Delete spaces at the beginning:** Deletes all space characters (decimal ASCII code 32) from the beginning of a string.
- **Delete spaces at the end:** Deletes all space characters (decimal ASCII value 32) from the end of a string.
- **Delete opening and closing character:** Deletes the first occurrence of the selected opening, and closing characters that are found in a string.

Example

If you use "{" for the opening character and "}" for the closing character, the input string `{{selection}}` converts to `{selection}`.

- **Search and replace:** Executes standard search and replace function upon the provided values for *find what* and *replace with*. You can also use regular expressions.

  NOTE

  There are several implementations of the regular expressions in use. uses the .NET Framework syntax for the regular expressions. For more information, see the Knowledge Base article.

- **Replace non printable characters with spaces:** Replaces all control characters in the string with space character (decimal ASCII code 32). The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Delete non printable characters:** Deletes all control characters in the string. The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Decode special characters:** Special characters (or control codes) are characters that are not available on the keyboard, such as Carriage Return or Line Feed. uses a notation to encode such characters in human-readable form, such as <CR> for Carriage Return and <LF> for Line Feed. For more information, see section Entering Special Characters (Control Codes).
  This option converts special characters from syntax into actual binary characters.

Example

When you receive the data sequence "<CR><LF>", uses it as a as plain string of 8 characters. Enable this option to interpret and use the received data as two binary characters **CR** (Carriage Return - ASCII code 13) and **LF** (Line Feed - ASCII code 10).

- **Search and delete everything before:** Finds the provided string and deletes all characters from the beginning of the data until the string. The found string itself can also be deleted.

- **Search and delete everything after:** Finds the provided string and deletes all characters from the string until the end of the data. The found string itself can also be deleted.
- **Change case**: Changes all characters in your strings to uppercase or lowercase.

Data Preview

This section provides preview of the field definition. If the defined item is selected, the preview highlights its placement in the preview data.

- **Preview file name:** Specifies the file that contains sample data that is going to be parsed by the filter. The preview file is copied from filter definition. If you change the preview file name, the new file name is saved.
- **Open:** Selects another file upon which you want to execute the filter rules.
- **Refresh:** Re-runs the filter rules upon the contents of the preview file name. Automation updates the Data Preview section with the result.

2.4.3. Defining Repeatable Elements in JSON Filter

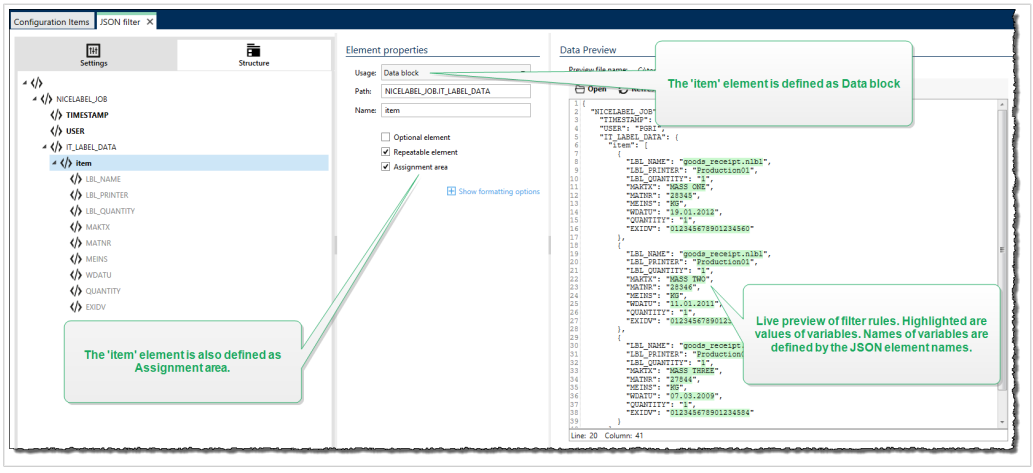
PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

If a JSON element occurs in your JSON data multiple times, it is a repeatable element. Usually, a repeatable element contains data for a single label template. Repeatable items produce multiple labels populated with relevant data.

To indicate that you want to use data from all repeatable elements, and not just from the first one:

1. Select the element and define it as a **Data block**.
2. Enable the **Repeatable element** option.



If the filter contains definition of elements defined as data block / repeatable element, the Use Data Filter action displays repeatable elements with nested placeholders. All actions nested below such a placeholder execute only for data blocks at this level.

Example

The **"item"** element is defined as both – **Data block** and **Repeatable element**. This instructs the filter to extract all occurrences of the array, not just the first one. In this case, the **"item"** should be defined as the sub-level in **Use Data Filter** action. You must nest the actions Open Label and Print Label under this sub-level placeholder, so they are going to be looped for as many times as there are occurrences of the **"item"** element. As shown in the example below, for three times.


```
{
    "NICELABEL_JOB": {
        "TIMESTAMP": "20130221100527.788134",
        "USER": "PGRI",
        "IT_LABEL_DATA": {
            "item": [
                {
                    "LBL_NAME": "goods_receipt.nlbl",
                    "LBL_PRINTER": "Production01",
                    "LBL_QUANTITY": "1",
                    "MAKTX": "MASS ONE",
                    "MATNR": "28345",
                    "MEINS": "KG",
                    "WDATU": "19.01.2012",
                    "QUANTITY": "1",
                    "EXIDV": "012345678901234560"
                },
                {
                    "LBL_NAME": "goods_receipt.nlbl",
                    "LBL_PRINTER": "Production01",
                    "LBL_QUANTITY": "1",
                    "MAKTX": "MASS TWO",
                    "MATNR": "28346",
                    "MEINS": "KG",
                    "WDATU": "11.01.2011",
                    "QUANTITY": "1",
                    "EXIDV": "012345678901234577"
                },
                {
                    "LBL_NAME": "goods_receipt.nlbl",
                    "LBL_PRINTER": "Production01",
                    "LBL_QUANTITY": "1",
                    "MAKTX": "MASS THREE",
                    "MATNR": "27844",
                    "MEINS": "KG",
                    "WDATU": "07.03.2009",
                    "QUANTITY": "1",
                    "EXIDV": "012345678901234584"
                }
```

```
            ]
        }
    }
}
```

2.4.4. Defining JSON Assignment Area

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

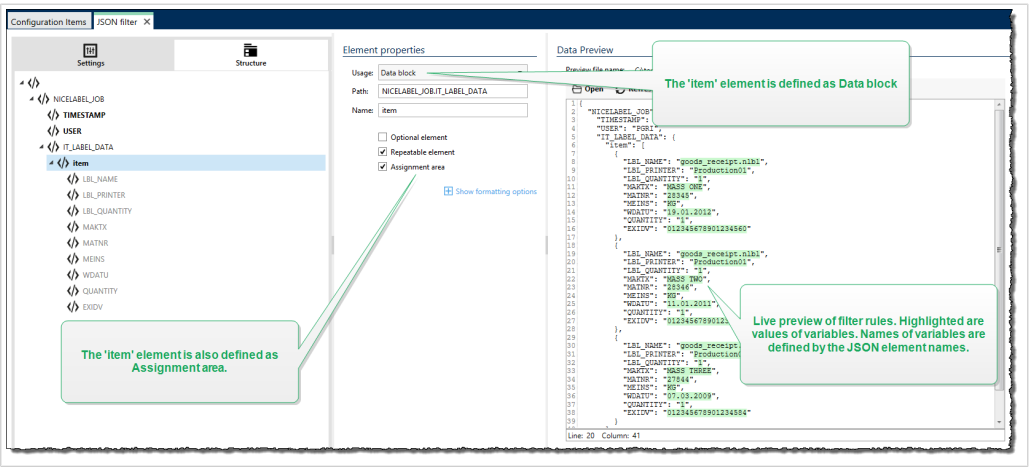
JSON filter automatically identifies fields and their values in the received data. This eliminates the need for manual *variable-to-field* mapping.

Dynamic structure functionality is helpful if the trigger receives data with changing structure. In such cases, main data structure remains unchanged (e.g., fields are delimited by a comma), or retains the same structure, but **the order** and/or **the number** of fields changes. There might be new fields, or some of the old fields could no longer be available. Because of enabled **Dynamic structure**, filter automatically identifies structure of the received file. At the same time, filter reads field names and values (**`name:value`** pairs) from the data. This eliminates the need for manual mapping of fields to variables.

Use Data Filter action does not offer any mapping possibilities, because it performs the mapping dynamically. You don't even have to define label variables in the trigger configuration. The action assigns field values to the label variables of the same name without requiring the variables to be imported from the label. However, this rule applies to the Print Label action alone. If you want to use the field values in any other action, you have to define variables in the trigger, while still keeping the automatic *variable-to-field* mapping.

NOTE

There is no error if the field available in the input data doesn't have a matching label variable. NiceLabel Automation silently ignores the missing variables.



NOTE

Because there are no optional attributes in JSON, Automation defines **Variable names** and **Variable values** automatically.

Formatting Options

This section defines string manipulation functions that apply on the selected variables or fields. You can select one or several functions. These functions are applied in the same order as selected in the user interface – from top to bottom.

- **Delete spaces at the beginning:** Deletes all space characters (decimal ASCII code 32) from the beginning of a string.
- **Delete spaces at the end:** Deletes all space characters (decimal ASCII value 32) from the end of a string.
- **Delete opening and closing character:** Deletes the first occurrence of the selected opening, and closing characters that are found in a string.

Example

If you use "{" for the opening character and "}" for the closing character, the input string `{{selection}}` converts to `{selection}`.

- **Search and replace:** Executes standard search and replace function upon the provided values for *find what* and *replace with*. You can also use regular expressions.

  NOTE

  There are several implementations of the regular expressions in use. uses the .NET Framework syntax for the regular expressions. For more information, see the Knowledge Base article.

- **Replace non printable characters with spaces:** Replaces all control characters in the string with space character (decimal ASCII code 32). The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Delete non printable characters:** Deletes all control characters in the string. The non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Decode special characters:** Special characters (or control codes) are characters that are not available on the keyboard, such as Carriage Return or Line Feed. uses a notation to encode such characters in human-readable form, such as <CR> for Carriage Return and <LF> for Line Feed. For more information, see section Entering Special Characters (Control Codes).
  This option converts special characters from syntax into actual binary characters.

Example

When you receive the data sequence "<CR><LF>", uses it as a as plain string of 8 characters. Enable this option to interpret and use the received data as two binary characters **CR** (Carriage Return - ASCII code 13) and **LF** (Line Feed - ASCII code 10).

- **Search and delete everything before:** Finds the provided string and deletes all characters from the beginning of the data until the string. The found string itself can also be deleted.

- **Search and delete everything after:** Finds the provided string and deletes all characters from the string until the end of the data. The found string itself can also be deleted.
- **Change case**: Changes all characters in your strings to uppercase or lowercase.

Example

The **"LIST_ITEM"** element is defined as data block and assignment area.

```
{
  "DELIVERYNOTE": {
    "LIST_CUSTOMER_INFO": {
      "CUSTOMER_INFO": {
        "CUSTOMER_NAME": "Customer A",
        "CUSTOMER_STREET_ADDRESS": "Test St",
        "CUSTOMER_POST_ADDRESS": "1234, Test City",
        "CUSTOMER_NUMBER": "1234",
        "CURRENCY": "EUR",
        "DELIVERY_METHOD": "Express delivery",
        "EDI_INFORMATION": "EDI",
        "ORDER_TYPE": "CSO",
        "ORDER_NUMBER": "123",
      }
    }
  }
}
```

For more information, see section Examples.

2.5. Setting Label and Printer Names from Input Data

Typically, filters extract values from the received data and send them to label variables for printing. In such cases, label or printer names are hard-coded into the actions. For example, Open Label action hard-codes the label name, and Set Printer action hard-codes the printer name. However, the input data can also provide *meta-data*. These are the values used inside the NiceLabel Automation processing, but not printed on the label, such as label name, printer name, label quantity, etc.

To use the values of meta-data fields in the print process, do the following.

1. **Filter reconfiguration:** Define new fields for the input data to extract the meta-data fields as well.
2. **Variable definition:** Manually define the variables to store the meta-data since they don't exist on the label and cannot be imported. Use intuitive names, such as **`LabelName`**, **`PrinterName`**, and **`Quantity`**. You are free to use any variable name.

3. **Mapping reconfiguration:** Manually configure the Use Data Filter action to map meta-fields to new variables.
4. **Action reconfiguration:** Reconfigure **Open Label** action to open the label specified by variable **`LabelName`**, and Set Printer action to use the printer as specified by the **`PrinterName`** variable.

Example

The CSV file contains label data, but also provides *meta-data*, such as label name, printer name and quantity of labels. The Structured Text filter extracts all fields, sends label-related values to the label variables and uses *meta-data* to configure actions Open Label, Set Printer and Print Label.

```
label_name;label_count;printer_name;art_code;art_name;ean13;weight
label1.nlbl;1;CAB A3 203DPI;00265012;SAC.PESTO 250G;383860026501;1,1 kg
label2.nlbl;1;Zebra R-402;00126502;TAGLIOLINI 250G;383860026002;3,0 kg
```

For more information, see section Examples.

3. Understanding Triggers

PRODUCT LEVEL INFO

This functionality is not entirely available with every NiceLabel Automation product level.

NiceLabel Automation is an event-based application that triggers the execution of actions upon changes in monitored events. You can use any of the available triggers to monitor changes in events, such as file drop into a certain folder, acquired data on a specific TCP/IP socket, HTTP message, and others. Main purpose of a trigger is to detect changes in events, retrieve the data provided by the event, and to execute actions.

The majority of triggers passively listens for the monitored event to occur. There are two exceptions. **Database trigger** is an active trigger that periodically checks for changes in the monitored database. **Serial port trigger** can wait for incoming connection, or can actively poll for data in specified time intervals.

Processing Triggers

In most cases, the trigger receives data that must be printed on labels. Once the trigger receives the data, the actions are executed in the defined order from top to bottom. The received data can contain values for label objects. However, before you can use these values, you must extract them from the received data and save them in variables. The filters define extraction rules. When executed, filters save the extracted data to the mapped variables. Once you have the data stored in variables, you can run actions that use the variables, such as Print Label.

After an event occurs, the provided input data is saved in a temporary file located in the service user's **`%temp%`** folder. The internal variable **`DataFileName`** refers to the temporary file location. The file is deleted when the trigger completes its execution.

Trigger Properties

To configure a trigger, define how to accept the data and the actions that you want to run. Optionally, you can also use variables. There are three sections that form the trigger configuration.

- **Settings:** Defines main parameters of the selected trigger. Select the event that the trigger will monitor for changes, or define the inbound communication channel. Settings tab allows you to select the script programming engine and security options. The available options depend on the trigger type. For more information, see section Trigger Types below.
- **Variables:** This tab defines the variables you need inside the trigger. Usually, you import variables from the label templates, so you can map them with the fields extracted from the inbound data. You can also define variables to be used internally in various actions and won't be sent to the label. For more information, see section Variables.
- **Actions:** This tab defines the actions to be executed whenever the trigger detects a change in the monitored event. Actions execute in order from top to bottom. For more information, see section Actions.

Trigger Types

- **File Trigger:** Monitors the change in the file or in a set of files in the folder. Contents of the file can be parsed in filters and used in actions.
- **Serial Port Trigger:** Monitors inbound communication on the serial RS232 port. Contents of the input stream are parsed by filters and used in actions. The data can be also polled from the external device in defined time intervals.
- **Database Trigger:** Monitors record changes in SQL database tables. Contents of the returned data set can be parsed and used in actions. The database is monitored in defined time intervals. The trigger can also update the database after the actions execute using **`INSERT`**, **`UPDATE`** and **`INSERT SQL`** statements.
- **Scheduler Trigger:** Executes your trigger in scheduled time intervals.
- **TCP/IP Server Trigger:** Monitors the inbound raw data stream arriving on the defined socket. Contents of the input stream is parsed by filters and used in actions. TCP/IP server trigger can be bidirectional and used to provide feedback.
- **TCP/IP Client Trigger:** Turns your Automation into a listening client that connects to TCP/IP servers.
- **HTTP Server Trigger:** Monitors the inbound HTTP-formatted data stream arriving on the defined socket. Contents of the input stream is parsed by filters and used in actions. User authentication can be enabled. Is bidirectional, providing feedback.
- **Web Service Trigger:** Monitors the inbound data stream arriving on the defined Web Service method. Contents of the input stream are parsed by filters and used in actions. Is bidirectional, providing feedback.
- **Cloud Trigger:** Captures data from NiceLabel Cloud.

Error Handling in Triggers

- **Configuration errors:** The trigger is in error state if it is configured improperly or incompletely. For example, you have configured the file trigger, but failed to specify the file name to check for changes. Or, you defined the action to print labels, but you haven't specified the label name. You can save triggers that contain configuration errors, but you cannot run them in Automation Manager until you resolve the issue. The error reported in the lower level of the configuration propagates itself all the way to the higher level, so it is easy to find the error location.

  Example

  If you have one action in error state, all upper-level actions indicate the error state. The error icon is displayed in Actions tab and in trigger name.

- **Overlapping configurations:** While it is perfectly acceptable for the configuration to include triggers monitoring the same event, such as the same file name, or listening on the same TCP/IP port, such triggers cannot run simultaneously. When you start the trigger in Automation Manager, it starts only if no other trigger from the same or other configuration monitors the same event.

Print Job Status Feedback

See section Print Job Status Feedback.

3.1. File Trigger

To learn more about triggers in general, see section Understanding Triggers.

File trigger event occurs if:

- monitored file changes
- set of files in the monitored folder changes
- a new file appears in the monitored folder

Depending on the trigger configuration, Windows operating system alerts the trigger about the changed files, or the trigger itself keeps a list of the file's last-write time stamp and fires after the file receives a newer time-stamp.

NOTE

Typical usage: business system executes a transaction, which in effect generates trigger file in a shared folder. Data content might be structured in CSV, XML and other formats, or it can be structured in a legacy format. In either way, NiceLabel Automation reads the data, parses values using filters and prints them on labels. For more information on how to parse and extract data, see section Understanding Filters.

TIP

To help you build configurations with file trigger see Automation sample files: Compound CSV, CSV Medium, CSV Simple, etc. Find sample files under **Help** > **Sample Files**.

General

This section allows you to configure the most important file trigger settings.

- **Name:** Specifies the unique name of the trigger. The names helps you distinguish between different triggers when you configure them in Automation Builder, and later when you run them in Automation Manager.
- **Description:** Allows you to describe the role of this trigger. Help the users with a short explanation about what the trigger does.
- **Detect the specified file:** Specifies path and name of the file that you monitor for changes.
- **Detect a set of files in the specified folder:** Specifies path to the folder, which you monitor for file changes, and the file names. You can use standard Windows wild cards "*" and "?". Some file types are predefined in the drop-down box, but you can also enter your own types.

NOTE

When monitoring a network folder, make sure you use the UNC notation of `\\server\share\file`. For more information, see section Access to Network Shared Resources.

- **Automatically detect changes:**NiceLabel Automation responds to file changes as soon as the file is created or changed. In this case, Windows operating system informs NiceLabel Automation Service about the change. You can use it when the monitored folder is located on the local drive and also in some network environments.
- **Check for changes in folder in intervals:**NiceLabel Automation scans the folder for file changes in the defined time intervals. In this case, NiceLabel Automation monitors folder for file changes by itself. This polling method tends to be slower than automatic detection. Use it as a fallback, when automatic detection cannot be used in your environment.

Execution

Options in the **File Access** section specify how the application accesses the trigger file.

- **Open file exclusively:** Opens the trigger file in exclusive mode. No other application can access the file at the same time. This is the default selection option.
- **Open file with read only permissions:** Opens the trigger file in read-only mode.
- **Open file with read and write permissions:** Opens the trigger file in read-write mode.
- **File open retry period:** Specifies the time period after which NiceLabel Automation tries to open the trigger file. If the file access is still not possible after this time period, NiceLabel Automation reports an error.

Options in the **Monitoring Options** section specify the file detection possibilities.

- **Check file size:** Enables detection of changes not only for file time-stamp, but also for file length. The changes in file time-stamp might not be detected in some cases. Therefore, Automation also checks for changed file size and triggers the actions.
- **Ignore empty trigger files:** If the trigger file has no contents, it is ignored. The actions do not execute.
- **Delete the trigger file:** After the change in the trigger file has been detected, and the trigger fires, Automation deletes the file. Enabling this option keeps the folder clean of processed files.

NOTE

NiceLabel Automation always creates a backup of the received trigger data. The contents of trigger file is saved using a unique file name. This is important, if you need the contents of the trigger file in some of the actions, such as **Run Command File**. The location of the backup trigger data is referred to by the internal variable *DataFileName*.

- **Empty file contents:** After the actions execute, the trigger file is emptied. This is useful if the third party applications append data to the trigger file. You want to keep the file, so appending can be done, but you don't want to print the old data.
- **Track changes while trigger is inactive:** Fires trigger upon files that change while the trigger is inactive. If your NiceLabel Automation is not deployed in a high-availability environment with backup servers, the incoming trigger files might become lost if the server is down. After the NiceLabel Automation is back online, the existent trigger files are processed.
- **Number of concurrent action executions**: Specify the number of your concurrent action executions. You don't have to wait for the execution to finish in order for the next one to start. Your processing action order stays the same while simultaneously the same action from another thread can start executing.
  The maximum number of your concurrent action executions also depends on your hardware performance. Learn more about Section 7.1, “Parallel Processing”.

Other

Options in the **Feedback from the Print Engine** section specify communication parameters that allow you to receive print engine feedback.

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

- **Supervised printing:** Activates synchronous printing mode. Use it whenever you want to send the print job status back to the third party application. For more information, see section Synchronous Print Mode.

Options in the **Data Processing** section specify if you want to trim the data so that it fits into a variable, or ignore the missing label variables. By default, reports errors and breaks the printing process if you try to save values that are too long in label variables, or try to set values for non-existing label variables.

- **Ignore excessive variable contents:** truncates data values that exceed the length of the variable as defined in the label designer to make them fit. This option is in effect if you are setting variable values in filters, from command files, and when you are setting values of trigger variables to label variables of the same name.

  Example

  Label variable accepts 5 characters at maximum. With this option enabled, any value longer than 5 characters is truncated to the first 5 characters. If the value is 1234567 ignores digits 6 and 7.

- **Ignore missing label variables:** When printing with command files (such as JOB file), the printing process ignores all variables that are:
  - specified in the command file (using the SET command)
  - not defined on the label

  Similar happens if you define assignment area in a filter to extract all name-value pairs, but your label contains fewer variables.
  When setting values of non-existing label variables, reports an error. If this option is enabled, the printing continues.

Options in **Scripting** section specify scripting possibilities.

- **Scripting language:** Selects scripting language for the trigger. All **Execute script** actions that you use within a single trigger use the selected scripting language.

Options in the **Save Received Data** section specify the available commands for data that the trigger receives.

- **Save data received by the trigger to file:** Enable this option to save the data received by the trigger. The option **Variable** enables variable file name. Select a variable that contains path and file name.
- **On error save data received by the trigger to file:** Enable this option to save the data into the trigger only if an error occurs during the action execution. You might want to enable this option to keep the data that caused the issue ready for troubleshooting.

NOTE

Make sure you enable the Supervised printing support. If not, cannot detect errors during the execution. For more information, see section Synchronous Print Mode.

NOTE

saves the received data into a temporary file. This temporary file is deleted right after the trigger execution completes. The internal variable **`DataFileName`** points to that file name. For more information, see Internal Variables.

Security

- **Lock and encrypt trigger:** Enables trigger protection. If you enable it, the trigger becomes locked and you cannot edit it any longer. This encrypts the actions. Only users with password can unlock the trigger and modify it.

3.2. Serial Port Trigger

To learn more about triggers in general, see section Understanding Triggers.

Serial port trigger event occurs when data is received on the monitored RS232 serial port.

Typical usage: **(1) Printer replacement.** You are retiring the existing serial port-connected label printer. In its place NiceLabel Automation will accept the data, extract the values for label objects from the received print stream, and create a print job for the new printer model. **(2) Weight scales.** Weight scales provides data about the weighted object. NiceLabel Automation extracts the required data from the received data stream, and prints a label. For more information on how to parse and extract data, see section Understanding Filters.

TIP

To help you build configurations with serial port trigger see Automation sample file Scan & Print from Excel. Find sample files under **Help** > **Sample Files**.

General

This section allows you to configure the most important file trigger settings.

- **Name:** Specifies the unique name of the trigger. The names helps you distinguish between different triggers when you configure them in Automation Builder, and later when you run them in Automation Manager.
- **Description:** Allows you to describe the role of this trigger. Help the users with a short explanation about what the trigger does.

- **Port:** Specifies the serial port (COM) number on which the incoming data is received. Use a port that is not in use by any other application or device, such as printer driver. If the selected port is in use, you won't be able to start the trigger in Automation Manager.

The options in the **Port Settings** section specify communication parameters that must match the serial port device parameters.

- **Disable port initialization:** Port initialization is not executed after you start the trigger in Automation Manager. This option is sometimes required for virtual COM ports.

Execution

- **Use initialization data:** Specifies that you want to send the initialization string to the serial device each time the trigger is started. Some serial devices require to be awaken or put into standby mode before they can provide the data. For more information about the initialization string and if you need it at all, see your device's user guide. You can include binary characters. For more information, see section Entering Special Characters.
- **Use data polling:** Specifies that the trigger actively asks the device for data. Within the specified time intervals, the trigger sends the commands provided in the **Contents** field. This field can include binary characters. For more information, see section Entering Special Characters.

Other

Options in the **Feedback from the Print Engine** section specify communication parameters that allow you to receive print engine feedback.

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

- **Supervised printing:** Activates synchronous printing mode. Use it whenever you want to send the print job status back to the third party application. For more information, see section Synchronous Print Mode.

Options in the **Data Processing** section specify if you want to trim the data so that it fits into a variable, or ignore the missing label variables. By default, reports errors and breaks the printing process if you try to save values that are too long in label variables, or try to set values for non-existing label variables.

- **Ignore excessive variable contents:** truncates data values that exceed the length of the variable as defined in the label designer to make them fit. This option is in effect if you are setting variable values in filters, from command files, and when you are setting values of trigger variables to label variables of the same name.

Example

Label variable accepts 5 characters at maximum. With this option enabled, any value longer than 5 characters is truncated to the first 5 characters. If the value is 1234567 ignores digits 6 and 7.

- **Ignore missing label variables:** When printing with command files (such as JOB file), the printing process ignores all variables that are:
  - specified in the command file (using the SET command)
  - not defined on the label

  Similar happens if you define assignment area in a filter to extract all name-value pairs, but your label contains fewer variables.
  When setting values of non-existing label variables, reports an error. If this option is enabled, the printing continues.

Options in **Scripting** section specify scripting possibilities.

- **Scripting language:** Selects scripting language for the trigger. All **Execute script** actions that you use within a single trigger use the selected scripting language.

Options in the **Save Received Data** section specify the available commands for data that the trigger receives.

- **Save data received by the trigger to file:** Enable this option to save the data received by the trigger. The option **Variable** enables variable file name. Select a variable that contains path and file name.
- **On error save data received by the trigger to file:** Enable this option to save the data into the trigger only if an error occurs during the action execution. You might want to enable this option to keep the data that caused the issue ready for troubleshooting.

NOTE

Make sure you enable the Supervised printing support. If not, cannot detect errors during the execution. For more information, see section Synchronous Print Mode.

NOTE

saves the received data into a temporary file. This temporary file is deleted right after the trigger execution completes. The internal variable `DataFileName` points to that file name. For more information, see Internal Variables.

Security

- **Lock and encrypt trigger:** Enables trigger protection. If you enable it, the trigger becomes locked and you cannot edit it any longer. This encrypts the actions. Only users with password can unlock the trigger and modify it.

3.3. Database Trigger

To learn more about triggers in general, see section Understanding Triggers.

Database trigger event occurs when a change in the monitored database table is detected. There might be new records, or existing records that have been updated. Database trigger doesn't wait for any event change, such as data delivery. Instead, it pulls data from the database in defined time intervals.

Typical usage: An existing business system executes a transaction, which in effect updates data in a database table. NiceLabel Automation detects the updated and new records, and prints their contents on labels.

General

This section allows you to configure the most important file trigger settings.

- **Name:** Specifies the unique name of the trigger. The names helps you distinguish between different triggers when you configure them in Automation Builder, and later when you run them in Automation Manager.
- **Description:** Allows you to describe the role of this trigger. Help the users with a short explanation about what the trigger does.
- **Database connection:** Specifies connection string to the database. Click **Define** to open the Database dialog box. Use it to configure the database connection, including database type, table name, and user credentials. You have to connect to a database that supports access with SQL commands. For this reason, you cannot use the database trigger to automatically detect data changes in CSV text files or Microsoft Excel spreadsheets.

  NOTE

  Configuration details depend on the type of selected database. The options in the dialog box depend on the database driver that you use. For configuration details, see user guide for your database driver. For more information about database connectivity, see section Accessing Databases.

- **Check database in the time intervals:** Specifies the time interval in which the database is polled for changes in records.
- **Detection Options and Advanced:** These options allow you to fine-tune the record detection mechanism. After the records are acquired from the database, Actions tab automatically displays the For Each Record action, using which you map table fields to label variables.

Get records based on unique incremental field value

With this option enabled, the trigger monitors the specified auto-incremental numeric field in the table. NiceLabel Automation remembers the field's value for the last processed record. At the next polling interval, only records with values greater than the remembered value are acquired.

To configure this option, select the table name in which the records reside **`(table name)`**, the auto-incremental field **`(key field)`** and the starting value for the field **`(key field default value)`**. Internally, the variable **`KeyField`** is used to refer to the last remembered value of key field.

NOTE

The last value of the key field is remembered internally, but is not updated back into configuration, so the value for **`key field default value`** does not change in this dialog box. You can safely reload configuration and/or stop/start this trigger in the Automation Manager and still keep the last remembered value. However, if you remove the configuration from Automation Manager and add it back, the value of last remembered key field is reset to what you have defined in **`key field default value`**.

Get records and delete them

With this option selected, all records are acquired from the table and deleted. To configure this option, select the table name in which the records reside (**`table name`**) and specify the primary key in the table (**`key fields`**). While Automation allows you to have a table without a primary key, it is strongly recommended that you define the primary key. If the primary key exists, the records are deleted one by one if a particular record is processed in the actions.

WARNING

If the primary key does not exist, all records obtained by the current trigger are deleted at once. That's fine if no error occurs during record processing. However, if there is a processing error with some record, Automation stops processing further records. Because all records captured in this polling interval have already been deleted without being processed, you can lose data. Therefore, having a primary key in a table is advisable.

SQL Code Examples

NOTE

These SQL statements are read-only and provided for reference only. To provide custom SQL statements, select the **Get and manage records with custom SQL** detection method.

Example table:

ID	ProductID	CodeEAN	ProductDesc	AlreadyPrinted
1	CAS0006	8021228110014	CASONCELLI ALLA CARNE 250G	Y
2	PAS501	8021228310001	BIGOLI 250G	
3	PAS502GI	8021228310018	TAGLIATELLE 250G	

Example of Update SQL statement in case the table contains primary index:

```
DELETE FROM [Table]
WHERE [ID] = :ID
```

`ID` field in the table is defined as primary index. The **`:ID`** construct in WHERE clause contains value of field ID in each iteration. For the first record, the value of **`ID`** is 1, for the second record 2, etc. Specifying the colon in front of the field name in SQL statement specifies how the variable is used.

Example of Update SQL statement in case the table does has no primary index defined:

```
DELETE FROM [Table]
```

If no primary index is defined in the table, all records are deleted from the table after the first record gets processed.

Get records and update them

In this case, all records are acquired from the table and updated. You can write a custom value into field in the table as indication 'this records has already been printed'. To configure this option, you have to select the table name, in which the records reside **`(table name)`**, select the field that you want to update **`(update field)`**, and enter the value that will be stored in the field **`(update value)`**. Internally, the variable **`UpdateValue`** is used in the SQL statement to refer to the current value of field (**`update value`**).

While Automation allows you to have a table without a primary key, it is strongly recommended that you define a primary key. If the primary key exists, the records will be updated one by one , when the particular record is processed in the actions.

WARNING

If the primary key does not exists, all records obtained in the trigger are updated at once. That's fine if there is no error processing the records. But if there is an error processing some record, the Automation stops processing further records. Because all records captured in this polling interval have already been updated without being processed in actions, you can lose data. Therefore, having a primary key in a table is a good idea.

SQL Code Examples

NOTE

These SQL statements are read-only and provided as a reference only. To provide custom SQL statements, select the **Get and manage records with custom SQL** detection method.

Example table:

ID	ProductID	CodeEAN	ProductDesc	AlreadyPrinted
1	CAS0006	8021228110014	CASONCELLI ALLA CARNE 250G	Y
2	PAS501	8021228310001	BIGOLI 250G	
3	PAS502GI	8021228310018	TAGLIATELLE 250G	

Example of Update SQL statement, if table contains primary index:

```
UPDATE [Table]
SET [AlreadyPrinted] = :UpdateValue
WHERE [ID] = :ID
```

`ID` field in the table is defined as primary index. The construct **`:ID`** in the WHERE clause contains the value of field ID in each iteration. For first record, the value of **`ID`** is 1, for second record 2, etc. Adding a colon in front of the field name in SQL statement specifies usage of a variable. The field **`UpdateValue`** is defined in the trigger configuration using the **Update value** edit field.

Example of Update SQL statement, when table does not have primary index defined:

```
UPDATE [Table]
SET [AlreadyPrinted] = :UpdateValue
```

If no primary index is defined in the table, all records from the table are updated, after the first record gets processed.

Get and manage records with custom SQL

In this case, the creation of SQL statements for record extraction and field updates is entirely up to you. To configure this option, you have to provide a custom SQL statement to acquire records **`(search SQL statement)`** and a custom SQL statement to update the records after processing **`(update SQL statement)`**. Click the **Test** button to test-execute your SQL statements and see the result on-screen.

You can use table field values or values of trigger variables as parameters in the WHERE clause in the SQL statement. You would precede the field or variable name using the colon character (:). This instructs NiceLabel Automation to use the current value of that field or variable.

SQL Code Examples

Example table:

ID	ProductID	CodeEAN	ProductDesc	AlreadyPrinted
1	CAS0006	8021228110014	CASONCELLI ALLA CARNE 250G	Y
2	PAS501	8021228310001	BIGOLI 250G	
3	PAS502GI	8021228310018	TAGLIATELLE 250G	

Example of Search SQL statement:

To get the records that haven't already been printed, do the following. The field **`AlreadyPrinted`** neither must contain value **`Y`**, nor have blank or NULL value.

```
SELECT * FROM Table
WHERE AlreadyPrinted <> 'Y' or AlreadyPrinted is NULL
```

From the sample table above, two records with ID values 2 and 3 will be extracted. The first record has already been printed and will be ignored.

Example of Update SQL statement:

To mark the already printed records with value **`Y`** in the **`AlreadyPrinted`** field, do the following:

```
UPDATE [Table]
SET [AlreadyPrinted] = 'Y'
WHERE [ID] = :ID
```

Put colon (**`:`**) in front of the variable name in your SQL statement to identify it as a variable. You can use any field from the table for parameters in the WHERE clause. In the example, we are updating the **`AlreadyPrinted`** field only for the currently processed record (value of field **`ID`** must be the same as the value from the current record). In the similar way, you would refer to other fields in the record as **`:ProductID`** or **`:CodeEAN`**, or even refer to variables defined inside this database trigger.

To delete the current record from the table, do the following:

```
DELETE FROM [Table]
WHERE [ID] = :ID
```

Show SQL statement: Expand this section to see the generated SQL statement and to write your own statement if you have selected the option **Get and manage records with custom SQL**.

Previewing SQL Execution

To test the execution of SQL sentences and to see what their effect is, click Test in the toolbar of the SQL edit area. The Data Preview section opens in the right-hand pane. Click the **Execute** button to start the SQL code. If you use values of table field in the SQL statement (with colon (:) in front of the field name), you have to provide test values for them.

NOTE

If you have Data Preview open and you have just added some variables to the script, click **Test** button twice. This closes and opens Data Preview section and updates the list of variables in the preview.

- **Simulate execution:** Specifies that all changes made to the database are ignored. The database transaction is reverted so no updates are written to the database.

Execution

The options in Execution specify when does the database updating take place. The update type depends on the Detection Options for the trigger.

- **Before processing actions:** Specifies that records are updated before the actions defined for this trigger have started to execute.
- **After processing actions:** Specifies that records are updated after the actions defined for this trigger have been executed. Usually you want to update the records after they have been successfully processed.

NOTE

If necessary, you can update the records while the actions are still executing. For more information, see section Execute SQL Statement.

Other

Options in the **Feedback from the Print Engine** section specify communication parameters that allow you to receive print engine feedback.

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

- **Supervised printing:** Activates synchronous printing mode. Use it whenever you want to send the print job status back to the third party application. For more information, see section Synchronous Print Mode.

Options in the **Data Processing** section specify if you want to trim the data so that it fits into a variable, or ignore the missing label variables. By default, reports errors and breaks the printing process if you try to save values that are too long in label variables, or try to set values for non-existing label variables.

- **Ignore excessive variable contents:** truncates data values that exceed the length of the variable as defined in the label designer to make them fit. This option is in effect if you are setting variable values in filters, from command files, and when you are setting values of trigger variables to label variables of the same name.

Example

Label variable accepts 5 characters at maximum. With this option enabled, any value longer than 5 characters is truncated to the first 5 characters. If the value is 1234567 ignores digits 6 and 7.

- **Ignore missing label variables:** When printing with command files (such as JOB file), the printing process ignores all variables that are:
  - specified in the command file (using the SET command)
  - not defined on the label

  Similar happens if you define assignment area in a filter to extract all name-value pairs, but your label contains fewer variables.
  When setting values of non-existing label variables, reports an error. If this option is enabled, the printing continues.

Options in **Scripting** section specify scripting possibilities.

- **Scripting language:** Selects scripting language for the trigger. All **Execute script** actions that you use within a single trigger use the selected scripting language.

Options in the **Save Received Data** section specify the available commands for data that the trigger receives.

- **Save data received by the trigger to file:** Enable this option to save the data received by the trigger. The option **Variable** enables variable file name. Select a variable that contains path and file name.
- **On error save data received by the trigger to file:** Enable this option to save the data into the trigger only if an error occurs during the action execution. You might want to enable this option to keep the data that caused the issue ready for troubleshooting.

NOTE

Make sure you enable the Supervised printing support. If not, cannot detect errors during the execution. For more information, see section Synchronous Print Mode.

NOTE

saves the received data into a temporary file. This temporary file is deleted right after the trigger execution completes. The internal variable **`DataFileName`** points to that file name. For more information, see Internal Variables.

Security

- **Lock and encrypt trigger:** Enables trigger protection. If you enable it, the trigger becomes locked and you cannot edit it any longer. This encrypts the actions. Only users with password can unlock the trigger and modify it.

3.4. TCP/IP Server Trigger

To learn more about triggers in general, see section Understanding Triggers.

TCP/IP server trigger event occurs after the monitored socket (IP address and port number) receives data.

Typical usage: An existing business system executes a transaction, which in effect sends the data to NiceLabel Automation server on a specific socket. Data content might be structured using CSV, XML and other formats, or it might use a legacy format. In either way, NiceLabel Automation reads the data, parses values using filters and prints them on labels. For more information on how to parse and extract the data, see section Understanding Filters.

General

NOTE

This trigger supports Internet Protocol version 6 (IPv6).

This section allows you to configure the most important file trigger settings.

- **Name:** Specifies the unique name of the trigger. The names helps you distinguish between different triggers when you configure them in Automation Builder, and later when you run them in Automation Manager.
- **Description:** Allows you to describe the role of this trigger. Help the users with a short explanation about what the trigger does.
- **Port:** Specifies the port number where incoming data will be accepted on. Use the port number that is not in use by any other application. If the selected port is in use, you won't be able to start the trigger in Automation Manager. For more information about security concerns, see section Securing Access to your Triggers.

NOTE

If your server has multi-homing enabled (more IP addresses on one or more network cards), NiceLabel Automation responds on the defined port on all IP addresses.

- **Maximum number of concurrent connections:** Specifies the maximum number of accepted connections. That many concurrent clients can send data to the trigger simultaneously.

The options in the **Execution Event** section specify when the trigger should fire and start executing actions.

- **On client disconnect:** Specifies that the trigger fires after the client sends data and closes the connection. This is a default setting.

NOTE

If you want to send the print job status back to the third party application as a feedback, don't use this option. If the connection is left open, you can send feedback using the action **Send data to TCP/IP port** with the parameter *Reply to sender*.

- **On number of characters received:** Specifies that trigger fires when the required number of characters is received. In this case, the third party application can keep a connection open and continuously sends data. Each chunk of data must be of the same size.
- **On sequence of characters received:** Specifies that the trigger fires each time the required sequence of characters is received. You would use this option if you know that the 'end of data' is always identified by a unique set of characters. You can insert special (binary) characters using the button next to the edit field.
    - **Include in trigger data:** The sequence of characters that determines the trigger event is not stripped of the data, but is included along with the data. The trigger receives the complete received data stream.
- **When nothing is received after the specified time interval:** Specifies that the trigger fires after a required time interval passes since the last character is received.

Execution

- **Allow connections from the following hosts:** Specifies the list of IP addresses or host names of the computers that are allowed to connect to the trigger. Place each entry in a new line.
- **Deny connections from the following hosts:** Specifies the list of IP addresses or host names of the computers that are not allowed to connect to the trigger. Place each entry in a new line.
- **Welcome message:** Specifies a text message that returns to the client each time it connects to the TCP/IP trigger.
- **Answer message:** Specifies the text message that returns to the client each time the actions execute. Use this option when the client doesn't disconnect upon data send and expects the answer

about when the action execution ends. The answer message is hard-coded and thus always the same.

- **Message encoding:** Specifies the data encoding scheme, so the special characters can be correctly processed. NiceLabel Automation can automatically detect the data encoding, based on BOM header (text files), or encoding attribute (XML files).

Other

Options in the **Feedback from the Print Engine** section specify communication parameters that allow you to receive print engine feedback.

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

- **Supervised printing:** Activates synchronous printing mode. Use it whenever you want to send the print job status back to the third party application. For more information, see section Synchronous Print Mode.

Options in the **Data Processing** section specify if you want to trim the data so that it fits into a variable, or ignore the missing label variables. By default, reports errors and breaks the printing process if you try to save values that are too long in label variables, or try to set values for non-existing label variables.

- **Ignore excessive variable contents:** truncates data values that exceed the length of the variable as defined in the label designer to make them fit. This option is in effect if you are setting variable values in filters, from command files, and when you are setting values of trigger variables to label variables of the same name.

  Example

  Label variable accepts 5 characters at maximum. With this option enabled, any value longer than 5 characters is truncated to the first 5 characters. If the value is 1234567 ignores digits 6 and 7.

- **Ignore missing label variables:** When printing with command files (such as JOB file), the printing process ignores all variables that are:

  - specified in the command file (using the SET command)
  - not defined on the label

  Similar happens if you define assignment area in a filter to extract all name-value pairs, but your label contains fewer variables.
  When setting values of non-existing label variables, reports an error. If this option is enabled, the printing continues.

Options in **Scripting** section specify scripting possibilities.

- **Scripting language:** Selects scripting language for the trigger. All **Execute script** actions that you use within a single trigger use the selected scripting language.

Options in the **Save Received Data** section specify the available commands for data that the trigger receives.

- **Save data received by the trigger to file:** Enable this option to save the data received by the trigger. The option **Variable** enables variable file name. Select a variable that contains path and file name.
- **On error save data received by the trigger to file:** Enable this option to save the data into the trigger only if an error occurs during the action execution. You might want to enable this option to keep the data that caused the issue ready for troubleshooting.

> **NOTE**
> Make sure you enable the Supervised printing support. If not, cannot detect errors during the execution. For more information, see section Synchronous Print Mode.

> **NOTE**
> saves the received data into a temporary file. This temporary file is deleted right after the trigger execution completes. The internal variable **`DataFileName`** points to that file name. For more information, see Internal Variables.

Security

- **Lock and encrypt trigger:** Enables trigger protection. If you enable it, the trigger becomes locked and you cannot edit it any longer. This encrypts the actions. Only users with password can unlock the trigger and modify it.

Sending keepalive signals

Your network routing system can silently drop your connection when there is no TCP/IP traffic for a few minutes. To avoid being disconnected, you can choose to periodically send keepalive signals in NiceLabel Automation.

Enable sending keepalive signals in your `product.config` file:

1. Navigate to the System folder.
   %PROGRAMDATA%\NiceLabel\NiceLabel 10
2. Make a backup copy of the **`product.config`** file.
3. Open **`product.config`** in a text editor. The file has an XML structure.
4. Add the following lines:

```
<configuration>
   <IntegrationService>
     <KeepAliveTime>60000</KeepAliveTime>
     <KeepAliveInterval>10000</KeepAliveInterval>
   </IntegrationService>
</configuration>
```

NOTE

`KeepAliveTime` (in milliseconds): Specifies how long the TCP socket connection must be inactive before Automation sends a keepalive signal and waits for the acknowledge packets to be returned.

`KeepAliveInterval` (in milliseconds): Specifies at what intervals to send anotherkeepalive packet when the host is not returning any acknowledge packets.

When the TCP/IP trigger receives the keep alive packet and returns the acknowledge packet, the `KeepAliveTime` timer starts again.

Both values (`KeepAliveTime` and `KeepAliveInterval`) are mandatory to enable sending keepalive signals and must have values larger than 0.

5. Save the `product.config` file.
6. Restart your Automation service.

Enabled keepalive signals now keep your connection active.

3.5. TCP/IP Client Trigger

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

To learn more about triggers in general, see section Understanding Triggers.

The TCP/IP client trigger turns your Automation into a listening client that connects to TCP/IP servers. There are multiple devices and systems that take the role of a TCP/IP server: vision inspection systems, printers, PLCs, scanners, weighing scales, etc. Automation can connect to them and wait for the incoming data. After receiving a certain number of characters, a character sequence, or a timeout, the TCP/IP client trigger fires, and starts executing your actions. If the connection fails, the trigger allows you to automatically reconnect.

Typical use: You are automatically printing multiple types of packaging labels using a network printer. It is crucial for you to know when your printer is done printing one label type before you start printing the second label type. The TCP/IP client trigger allows you to set permanent automated checking of printer availability. While printing the first label type, the TCP/IP client trigger checks for the printing status, and when the printer is done, it sends a message to Automation to send the second label type for printing.

General

- **Name:** Specifies the unique name of the trigger. The names helps you distinguish between different triggers when you configure them in Automation Builder, and later when you run them in Automation Manager.

- **Description:** Allows you to describe the role of this trigger. Help the users with a short explanation about what the trigger does.
- **Destination server**: Type the location (IP address or hostname) of the TCP/IP server you wish to connect to.
- **Port:** Specifies the host port number from which you are going to receive the incoming data. Make sure firewall ports are open on the server side.
- **Reconnect to server interval**: Define the time in milliseconds after which Automation tries to reconnect to your TCP/IP server.
- **On number of characters received:** Specifies that the trigger fires when the required number of characters is received. In this case, server can keep a connection open and continuously sends data. Each chunk of data must be of the same size.
- **On sequence of characters received:** Specifies that the trigger fires each time the required sequence of characters is received. You would use this option if you know that the 'end of data' is always identified by a unique set of characters. You can insert special (binary) characters using the button next to the edit field.
    - **Include in trigger data:** The sequence of characters that determines the trigger event is not stripped of the data, but is included along with the data. The trigger receives the complete received data stream.
- **If nothing is received after the specified time interval:** Specifies that the trigger fires after a required time interval passes since the last character is received.

Execution

- **Initialization message**: Text message that goes to the server each time Automation establishes connection.
- **Answer message:** Specifies the text message that returns to the server upon firing the trigger (before the actions start to execute).
- **Message encoding:** Specifies the data encoding scheme, so the special characters can be correctly processed. NiceLabel Automation can automatically detect the data encoding, based on BOM header (text files), or encoding attribute (XML files).

Other

Options in the **Feedback from the Print Engine** section specify communication parameters that allow you to receive print engine feedback.

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

- **Supervised printing:** Activates synchronous printing mode. Use it whenever you want to send the print job status back to the third party application. For more information, see section Synchronous Print Mode.

Options in the **Data Processing** section specify if you want to trim the data so that it fits into a variable, or ignore the missing label variables. By default, reports errors and breaks the printing process if you try to save values that are too long in label variables, or try to set values for non-existing label variables.

- **Ignore excessive variable contents:** truncates data values that exceed the length of the variable as defined in the label designer to make them fit. This option is in effect if you are setting variable values in filters, from command files, and when you are setting values of trigger variables to label variables of the same name.

Example

Label variable accepts 5 characters at maximum. With this option enabled, any value longer than 5 characters is truncated to the first 5 characters. If the value is 1234567 ignores digits 6 and 7.

- **Ignore missing label variables:** When printing with command files (such as JOB file), the printing process ignores all variables that are:
  - specified in the command file (using the SET command)
  - not defined on the label

  Similar happens if you define assignment area in a filter to extract all name-value pairs, but your label contains fewer variables.
  When setting values of non-existing label variables, reports an error. If this option is enabled, the printing continues.

Options in **Scripting** section specify scripting possibilities.

- **Scripting language:** Selects scripting language for the trigger. All **Execute script** actions that you use within a single trigger use the selected scripting language.

Options in the **Save Received Data** section specify the available commands for data that the trigger receives.

- **Save data received by the trigger to file:** Enable this option to save the data received by the trigger. The option **Variable** enables variable file name. Select a variable that contains path and file name.
- **On error save data received by the trigger to file:** Enable this option to save the data into the trigger only if an error occurs during the action execution. You might want to enable this option to keep the data that caused the issue ready for troubleshooting.

NOTE

Make sure you enable the Supervised printing support. If not, cannot detect errors during the execution. For more information, see section Synchronous Print Mode.

> **NOTE**
> saves the received data into a temporary file. This temporary file is deleted right after the trigger execution completes. The internal variable **`DataFileName`** points to that file name. For more information, see Internal Variables.

Security

- **Lock and encrypt trigger:** Enables trigger protection. If you enable it, the trigger becomes locked and you cannot edit it any longer. This encrypts the actions. Only users with password can unlock the trigger and modify it.

Sending keepalive signals

Your network routing system can silently drop your connection when there is no TCP/IP traffic for a few minutes. To avoid being disconnected, you can choose to periodically send keepalive signals in NiceLabel Automation.

Enable sending keepalive signals in your `product.config` file:

1. Navigate to the System folder.
   %PROGRAMDATA%\NiceLabel\NiceLabel 10
2. Make a backup copy of the **`product.config`** file.
3. Open **`product.config`** in a text editor. The file has an XML structure.
4. Add the following lines:

```
<configuration>
   <IntegrationService>
     <KeepAliveTime>60000</KeepAliveTime>
     <KeepAliveInterval>10000</KeepAliveInterval>
   </IntegrationService>
</configuration>
```

> **NOTE**
> `KeepAliveTime` (in milliseconds): Specifies how long the TCP socket connection must be inactive before Automation sends a keepalive signal and waits for the acknowledge packets to be returned.
>
> `KeepAliveInterval` (in milliseconds): Specifies at what intervals to send anotherkeepalive packet when the host is not returning any acknowledge packets.
>
> When the TCP/IP trigger receives the keep alive packet and returns the acknowledge packet, the `KeepAliveTime` timer starts again.
>
> **Both values (`KeepAliveTime` and `KeepAliveInterval`) are mandatory to enable sending keepalive signals and must have values larger than 0.**

5. Save the `product.config` file.
6. Restart your Automation service.

Enabled keepalive signals now keep your connection active.

3.6. HTTP Server Trigger

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

To learn more about triggers in general, see section Understanding Triggers.

HTTP server trigger fires if data is received on the monitored socket (IP address and port number).

Contrary to TCP/IP trigger, the received data is not a raw data stream, but includes standard HTTP header. Third party applications must use POST or GET request methods. They must provide data in message body or in query string. You can use both Internet media types in the message body – MIME Type, or Content-Type. NiceLabel Automation receives the message and extracts relevant data from the message content using a filter.

Typical use: Existing business system executes a transaction which sends data to NiceLabel Automation server formatted as HTTP POST message to a specific socket. The sent data might be structured in CSV, XML and other formats, or it can be structured using a proprietary legacy format. Either way, NiceLabel Automation reads the data, parses values using filters and prints the extracted data on labels. For more information on how to parse and extract data, see section Understanding Filters.

TIP

To help you build configurations with HTTP server trigger see Automation sample file Label Preview as HTTP Response. Find sample files under **Help** > **Sample Files**.

Providing data

Provide the HTTP trigger data using any of the following methods. You can also combine the methods if needed, and use both in the same HTTP request.

Data in the query string

Query string is a part of the uniform resource locator (URL) that contains data to be passed to the HTTP trigger.

Example of a typical URL that contains query string:

```
http://server/path/?query_string
```

Question mark is used as a separator and is not part of the query string.

Query string is usually composed of a series of **`name:value`** pairs. Within each pair, field name and value are separated by equals sign (=). Series of pairs are separated by ampersand (&). A typical query string provide values for fields (variables) in the following format:

```
field1=value1&field2=value2&field3=value3
```

HTTP trigger offers built-in support for extracting values from all fields, and for storing them in variables that carry the same name. As a result, you don't have to define any filters to extract values from the query string.

- You don't have to define variables inside a trigger to populate them with values from the query string. NiceLabel Automation extracts all variables in the query string and sends their values to the currently active label. If variables of the same name exist in the label, Automation populates them with values. If the variables do not exist in the label, Automation ignores their values without reporting any errors.
- If an action requires variable values, define matching variables in the trigger. To get and store all values from the query string, create matching variables using the same names as fields in the query string. For the example above, define trigger variables with names **`field1`**, **`field2`** and **`field3`**.

You would usually use GET HTTP request method to provide the query string.

Data in body of HTTP request

Use POST request method to provide message in the body of an HTTP request.

You are free to send any type of data or use any data structure that you want in the body. You only need to ensure that you can handle the data using NiceLabel Automation filters. The content can be formatted as XML, CSV, or plain text. Sent content can even be binary data (Base64-encoded). Bare in mind that you have to parse the data with filters.

If you can affect the structure of the incoming message, use standardized structures, such as XML or CSV to simplify the filter configuration.

Use POST HTTP request method to provide the data in message body.

General

This section allows you to configure the most important file trigger settings.

- **Name:** Specifies the unique name of the trigger. The names helps you distinguish between different triggers when you configure them in Automation Builder, and later when you run them in Automation Manager.
- **Description:** Allows you to describe the role of this trigger. Help the users with a short explanation about what the trigger does.

Communication

> **NOTE**
> This trigger supports Internet Protocol version 6 (IPv6).

This section allows you to configure the mandatory port number and optional feedback options. You can use standard HTTP Response Codes to indicate a successfully executed action. For more advanced purposes, you can also send the custom content back to the data-providing application. Such content may be a simple feedback string, or binary data, such as label preview or print stream.

Typical URL to connect to an HTTP trigger is as follows:

```
http://server:port/path/?query_string
```

- **Server:** This is the FQDN or IP address of the machine where NiceLabel Automation is installed.
- **Port:** Number of the port on which incoming data is received. Use a port number that is not in use by any other application. If the selected port is in use, you won't be able to start the trigger in Automation Manager. For more information about security concerns, see section Securing Access to your Triggers.

  > **NOTE**
  > If multi-homing is enabled on your server (multiple IP addresses on one or more network cards), NiceLabel Automation responds on the defined port on all IP addresses.

- **Path:** Specifies optional path in the URL. This functionality enables NiceLabel Automation to expose multiple HTTP triggers on the same port. The client uses the triggers through a single port in a REST like syntax causing different triggers to be fired by different URLs. If you are not sure what to use, leave the default path (\).

  > **PRODUCT LEVEL INFO**
  > The functionality from this section is available in LMS Enterprise.

- **Secure connection (HTTPS):** Enables secure transport layer for your HTTP messages and prevents eavesdropping. For more information on how to set it up, see section Using Secure Transport Layer (HTTPS).
- **Query string:** Specifies name-value pairs in the URL. This an optional parameter. The data is usually provided in the body of the HTTP request.

- **Wait for trigger execution to finish:** HTTP protocol requires a receiver (in this case NiceLabel Automation) to send a numeric response back to the sender indicating the status of the received message. By default, NiceLabel Automation responds with code 200. This indicates that Automation successfully received the data, but gives no information about the success of trigger actions.
  This option specifies that a trigger doesn't send a response immediately after receiving the data, but waits until all actions are executed. After that, it sends the response code indicating a successful action execution. With this option enabled, you can send back a custom response type and data (e.g. the response to a HTTP request is label preview in PDF format).
  The available built-in HTTP response codes are:

HTTP Response Code	Description
200	All actions executed successfully.
401	Unauthorized, wrong user name and password were specified.
500	There were errors during action execution.

NOTE

To send feedback about the print process, enable the **synchronous** print mode. For more information, see Synchronous Print Mode.

- **Maximum number of concurrent requests:** Specifies the maximum number of concurrent inbound connections. That many concurrent clients can send data to the trigger simultaneously. This number also depends on the hardware performance of your server.
  Learn more about Section 7.1, "Parallel Processing".

- **Response type:** Specifies the type of your response message. Frequently used Internet media types (also known as MIME types, or Content-types) are predefined in the drop down box. If your media type is not available in the list, enter it by yourself. Automation sends the response data outbound as feedback, formatted in the defined media type. **Variable** enables variable media type. If enabled, select or create a variable that contains media type.

NOTE

If you don't specify the Content-Type, NiceLabel Automation uses **`application/octet-stream`** as the default one.

- **Response data:** Defines content of the response message. Examples of what you can send back as an HTTP response are custom error messages, label preview, generated PDF files, print stream (spool) file, XML file with details from the print engine plus the label preview (encoded as Base64 string). The possibilities are countless.
  If your output will consist of binary-only content (such as label preview or print stream), make sure you select a proper media type, e.g. **`image/jpeg`** or **`application/octet-stream`**.

- **Additional headers**: Allow you to define custom MIME headers for the HTTP response message. Response header syntax and example are available in the HTTP Request action section.

> **TIP**
>
> With Response data and Additional headers, you can use fixed content, mix of fixed and variable content, or variable content alone. To insert variable content, click the button with arrow to the right of data area and insert variable from the list (or create a new one) that contains the data you want to use. For more information, see section Using Compound Values.

Authentication

- **None**: No authentication method is in use.
- **User:** Specifies that incoming messages include user name and password. When in use, the trigger only accepts HTTP messages with matching credentials. For more information about security concerns, see section Securing Access to your Triggers.
- **Application Group (defined in NiceLabel Control Center):** As in case with **User** authentication type, this option also specifies that the incoming messages include user name and password. When in use, the trigger only accepts HTTP messages with adequate credentials for NiceLabel Control Center users who belong to a specific application group.
  - **Group**: Multiple application groups can be defined on the NiceLabel Control Center. To select which group should be allowed to access the HTTP Server Trigger, use the **Group** drop-down list. The selected group and its users must be set as active when the trigger is running.

    > **NOTE**
    >
    > Group with a specified name must exist on NiceLabel Control Center when the trigger is running. While working on the configuration in Automation Builder, you can use any group name. Make sure you eventually define a final name on NiceLabel Control Center and match it in the configuration before deploying it.

    > **TIP**
    >
    > The users authenticate themselves using their credentials as defined in **NiceLabel Control Center** , **Users** tab. Refer to NiceLabel Control Center User Guide for details on user management (section Users and Groups).

Other

Options in the **Feedback from the Print Engine** section specify communication parameters that allow you to receive print engine feedback.

> **PRODUCT LEVEL INFO**
>
> This functionality is available in **LMS Enterprise** and **LMS Pro**.

- **Supervised printing:** Activates synchronous printing mode. Use it whenever you want to send the print job status back to the third party application. For more information, see section Synchronous Print Mode.

Options in the **Data Processing** section specify if you want to trim the data so that it fits into a variable, or ignore the missing label variables. By default, reports errors and breaks the printing process if you try to save values that are too long in label variables, or try to set values for non-existing label variables.

- **Ignore excessive variable contents:** truncates data values that exceed the length of the variable as defined in the label designer to make them fit. This option is in effect if you are setting variable values in filters, from command files, and when you are setting values of trigger variables to label variables of the same name.

Example

Label variable accepts 5 characters at maximum. With this option enabled, any value longer than 5 characters is truncated to the first 5 characters. If the value is 1234567 ignores digits 6 and 7.

- **Ignore missing label variables:** When printing with command files (such as JOB file), the printing process ignores all variables that are:
  - specified in the command file (using the SET command)
  - not defined on the label

  Similar happens if you define assignment area in a filter to extract all name-value pairs, but your label contains fewer variables.
  When setting values of non-existing label variables, reports an error. If this option is enabled, the printing continues.

Options in **Scripting** section specify scripting possibilities.

- **Scripting language:** Selects scripting language for the trigger. All **Execute script** actions that you use within a single trigger use the selected scripting language.

Options in the **Save Received Data** section specify the available commands for data that the trigger receives.

- **Save data received by the trigger to file:** Enable this option to save the data received by the trigger. The option **Variable** enables variable file name. Select a variable that contains path and file name.
- **On error save data received by the trigger to file:** Enable this option to save the data into the trigger only if an error occurs during the action execution. You might want to enable this option to keep the data that caused the issue ready for troubleshooting.

NOTE

Make sure you enable the Supervised printing support. If not, cannot detect errors during the execution. For more information, see section Synchronous Print Mode.

NOTE

saves the received data into a temporary file. This temporary file is deleted right after the trigger execution completes. The internal variable **`DataFileName`** points to that file name. For more information, see Internal Variables.

Security

- **Lock and encrypt trigger:** Enables trigger protection. If you enable it, the trigger becomes locked and you cannot edit it any longer. This encrypts the actions. Only users with password can unlock the trigger and modify it.

3.7. Web Service Trigger

PRODUCT LEVEL INFO

The functionality from this section is available in LMS Enterprise.

To learn more about triggers in general, see section Understanding Triggers.

Web Service trigger event occurs if a monitored socket (IP address and port number) receives data. The data must follow SOAP notation – it encodes XML data into an HTTP message. The Web Service interface is described with WSDL document. Such document is available with each defined Web Service trigger.

Web Service trigger provides feedback about print job status, but you have to enable the **synchronous** processing mode. For more information, see section Print Job Status Feedback.

Typically, programmers use Web Service to integrate label printing into their own applications. An existing business system executes a transaction, which sends the data to NiceLabel Automation server using a specific socket. The sent data is formatted as a SOAP message. The data can be provided in CSV, XML and other structured formats, or it can be provided using one of the legacy formats. In both cases, NiceLabel Automation reads the data, parses values using filters and prints them on labels. For more information on how to parse and extract data, see section Understanding Filters.

TIP

To help you build configurations with Web server trigger see Automation sample file Web Service. Find sample files under **Help** > **Sample Files**.

General

This section allows you to configure general file trigger settings.

- **Name:** Specifies the unique name of the trigger. The names helps you distinguish between different triggers when you configure them in Automation Builder, and later when you run them in Automation Manager.

- **Description:** Allows you to describe the role of this trigger. Help the users with a short explanation about what the trigger does.

Communication

> **NOTE**
> This trigger supports Internet Protocol version 6 (IPv6).

This section allows you to configure mandatory port number and optional feedback settings.

- **Port:** Specifies number of the port that accepts incoming data. Use a port number that is not in use by any other application. If the selected port is in use, you won't be able to start the trigger in Automation Manager. For more information about security concerns, see section Securing Access to your Triggers.

  > **NOTE**
  > If your server has multi-homing enabled (multiple IP addresses on one or more network cards), NiceLabel Automation responds on the defined port on all IP addresses.

- **Secure connection (HTTPS):** Enables secure transport layer for your HTTP message and prevents eavesdropping. For more information on how to set up a secure conection, see section Using Secure Transport Layer (HTTPS).
- **Maximum number of concurrent calls:** Specifies the maximum number of accepted connections. That many concurrent clients can send data to the trigger simultaneously.
- **Response data:** Defines the custom response that can be used with `ExecuteTriggerWithResponse` and `ExecuteTriggerAndSetVariablesWithResponse` methods. The response data contains the content as provided in the text area. You can combine fixed values, variable values and special characters. To insert (or create) variables and special characters, click the arrow button to the right of the text area. The response can contain binary data, such as image of label preview and print file (*.PRN).

Status Feedback

By design, Web Service trigger provides feedback about the status of the created print job. The trigger accepts the provided data and uses it to execute the defined actions. You can supervise the execution of actions – the trigger reports success status for every event that happens during execution. To enable status reporting during the printing process, activate Synchronous Print Mode.

Web Service trigger exposes the following methods (functions):

- **ExecuteTrigger:** This method accepts data into processing and provides optional status feedback. One of the input parameters enables or disables feedback. If you enable status reporting, feedback contains error ID and detailed description of the error. If error ID equals 0, this makes no issue while creating the print file. If error ID is greater than 0, some error occurred during the print process. Web

Service response in this method is not configurable – the response always contains error ID and error description.

- **ExecuteTriggerWithResponse:** This method accepts data into processing and provides custom status feedback. Web Service response is configurable. You can respond using any type of data organized in any available structure. You can use binary data in the response.
- **ExecuteTriggerAndSetVariables:** Similar to the above described **ExecuteTrigger**, this web service exposes an additional inbound parameter that accepts the formatted list of *name-value* pairs. The trigger automatically parses the list, extracts values and stores values in variables of the same name, so you don't have to create any data-extraction filter yourself.
- **ExecuteTriggerAndSetVariablesWithResponse:** Similar to the **ExecuteTriggerWithResponse** above, this web service exposes an additional inbound parameter that accepts the formatted list of *name-value* pairs. The trigger automatically parses the list, extracts values and stores the values in variables of the same name, so you don't have to create any data-extraction filter by yourself.

For more information about the structure of messages that you can send using one of the above listed methods, see section WSDL below.

WSDL

Web Service Description Language (WSDL) specifies style of SOAP messages. It can either be **Remote Procedure Call (RPC)** style or **Document** style. Choose the style that your data providing application supports.

WSDL document defines input and output parameters of the Web Service.

After you define the Web Service trigger on port 12345, deploy it in Automation Manager and start it. WSDL becomes available at:

```
http://localhost:12345
```

WSDL exposes several methods that all provide data into the trigger. Choose the most appropriate method for what you are trying to achieve.

- Methods that have *WithResponse* in their names allow you to send customized responses, such as custom error messages, label previews, PDF files, print files (*.PRN), and similar. Methods without *WithResponse* in their name also provide feedback, but you cannot customize the response. The feedback contains default error messages.
- Methods that have *SetVariables* in their names allow you to provide a list of variables in two predefined formats. Automation automatically extracts values and maps them to the appropriate variables. This saves you time because you don't have to set up any filter to do the extraction and mapping. For the methods without *SetVariables* in their names, you have to define the filter by yourself.

Web Service interface defines the following methods:

Method ExecuteTrigger

Main part of the definition is:

```
<wsdl:message name="WebSrviTrg_ExecuteTrigger_InputMessage">
    <wsdl:part name="text" type="xsd:string"/>
    <wsdl:part name="wait" type="xsd:boolean"/>
</wsdl:message>
<wsdl:message name="WebSrviTrg_ExecuteTrigger_OutputMessage"
    <wsdl:part name="ExecuteTriggerResult" type="xsd:int"/
    <wsdl:part name="errorText" type="xsd:string"/>
</wsdl:message>
```

The definition includes two input variables (you provide their values):

- **text:** Filter in configuration parses this input string. Usually, the input string is structured as CSV or XML file which makes it easy to parse. You can also use any other textual file format.
- **wait:** This is a Boolean field that specifies two things:
  - If you wish to wait for the print job status response or not.
  - If the Web Service should provide feedback or not.

  In case of *True*, use **1**. In case of *False*, use **0**. Depending on the selected method type, there is either a predefined response, or you can send a customized response.

These are the optional output variables (you receive their values, if you request them by setting **wait** to **1**):

- **ExecuteTriggerResult:** The integer response contains value 0 in case of reported data processing error(s). It contains an integer greater than 0 if there are errors. The application executing the Web Service call to NiceLabel Automation can use the response as error indicator.
- **errorText:** This string value contains print job status response if an error occurs during trigger processing.

  > **NOTE**
  > If there is an error during trigger processing, this element is included in the XML response message and its value contains the error description. However, if there is no error, this element is not included in the XML response.

Method ExecuteTriggerWithResponse

You would use this method if a trigger send a custom response after it completes the execution.

Some examples of what you can send back as custom response: custom error messages, label preview, generated PDF files, print stream file (spool file), XML file with details from the print engine plus the label preview (encoded as Base64 string), the possibilities are endless.

Main part of the definition is:

```
<wsdl:message name="WebSrviTrg_ExecuteTriggerWithResponse_InputMessage">
    <wsdl:part name="text" type="xsd:string"/>
    <wsdl:part name="wait" type="xsd:boolean"/>
</wsdl:message>
```

```
<wsdl:message name="WebSrviTrg_ExecuteTriggerWithResponse_OutputMessage">
    <wsdl:part name="ExecuteTriggerWithResponseResult" type="xsd:int"/>
    <wsdl:part name="responseData" type="xsd:base64Binary"/>
    <wsdl:part name="errorText" type="xsd:string"/>
</wsdl:message>
```

In the example above, there are two input variables (you provide their values):

- **text:** Filter in configuration parses this input string. Usually, the input string is structured as CSV or XML file which makes it easy to parse. You can also use any other textual file format.
- **wait:** This is a Boolean field that specifies two things:
  - If you wish to wait for the print job status response or not.
  - If the Web Service should provide feedback or not.

  In case of *True*, use **1**. In case of *False*, use **0**. Depending on the selected method type, there is either a predefined response, or you can send a customized response.

Furthermore, the following optional output variables are included in the example above.

NOTE

You receive values of optional output variables if you request them by setting the **wait** field value to **1**.

- **ExecuteTriggerWithResponseResult:** Integer response contains value 0 if there are no issues during data processing. The response contains an integer greater than 0, if there are errors. The application that executes the Web Service call to NiceLabel Automation can use this response as error indicator.
- **responseData:** Custom response that you can define in the Web Service trigger configuration. The response is base64-encoded data.
- **errorText:** If an error raises during trigger processing, this string contains the print job status response value.

  NOTE

  If there is an error reported during trigger processing, the XML response message includes the **errorText** element. Value of this element contains error description. However, if there is no error, this element is not included in the XML response.

Method ExecuteTriggerAndSetVariables

Main part of the definition is:

```
<wsdl:message name="WebSrviTrg_ExecuteTriggerAndSetVariables_InputMessage">
    <wsdl:part name="text" type="xsd:string"/>
    <wsdl:part name="variableData" type="xsd:string"/>
    <wsdl:part name="wait" type="xsd:boolean"/>
```

```
</wsdl:message>
    <wsdl:message
name="WebSrviTrg_ExecuteTriggerAndSetVariables_OutputMessage">
    <wsdl:part name="ExecuteTriggerAndSetVariablesResult" type="xsd:int"/>
    <wsdl:part name="errorText" type="xsd:string"/>
</wsdl:message>
```

In the example above, there are three input variables (you provide their values):

- **text:** Filter in configuration parses this input string. Usually, the input string is structured as CSV or XML file which makes it easy to parse. You can also use any other textual file format.
- **wait:** This is a Boolean field that specifies two things:
  - If you wish to wait for the print job status response or not.
  - If the Web Service should provide feedback or not.

  In case of *True*, use **1**. In case of *False*, use **0**. Depending on the selected method type, there is either a predefined response, or you can send a customized response.

- **variableData:** This is the string that contains *name:value* pairs. The trigger reads all pairs and assigns provided values to the trigger variables of the same name. If the variable doesn't exist in the trigger, the trigger discards that *name:value* pair. If you provide the list of variables and their values using this method, you don't have to define any data extraction with filters. The trigger does all the parsing for you.
  There are two available structures for the variableData content.

  XML structure
  The trigger provides variables within the **<Variables />** root element of the XML file. Variable name includes attribute name, while the variable value includes element value.

```
<?xml version="1.0" encoding="utf-8"?>
<Variables>
    <variable name="Variable1">Value 1</variable>
    <variable name="Variable2">Value 2</variable>
    <variable name="Variable3">Value 3</variable>
</Variables>
```

NOTE

Embed your XML data inside the CDATA section. **CDATA**, meaning **character data**, is a section of element content that is marked for the parser to help it interpret the XML data as character-only data, and not as markup. As a result, the trigger handles the entire content as character data. For example, **<element>ABC</element>** gets interpreted as **<element>ABC</element>**. Each CDATA section starts with **<![CDATA[** sequence and ends with the **]]>** sequence. To sum up, simply insert your XML data inside these two sequences.

Name-value pairs

Trigger provides variables using a text stream. Every *name:value* pair comes in its own line. Variable name is to the left of the equals character (=), variable value is to the right.

```
Variable1="Value 1"
Variable2="Value 2"
Variable3="Value 3"
```

These are the optional output variables:

NOTE

You receive values for optional variables if you request them by setting **wait** to **1**:

- **ExecuteTriggerAndSetVariablesResult:** Integer response contains value 0 if there are no issues during data processing. It contains an integer greater than 0, if data processing reports error(s). The application that executes the Web Service call to NiceLabel Automation can use the response as error indicator.
- **errorText:** This string value contains print job status response in case it reports a trigger processing error.

  NOTE

  In case of trigger processing error, this element is included in the XML response message. Its value contains error description. However, if there is no error, this element is not included in the XML response.

Method ExecuteTriggerAndSetVariablesWithResponse

You would use this method if the trigger should send a custom response after it completes the execution.

Some examples of what you can send back as custom response: custom error messages, label preview, generated PDF files, print stream file (spool file), XML file with details from the print engine plus the label preview (encoded as Base64 string), the possibilities are endless.

Main part of the definition is:

```
<wsdl:message name="WebSrviTrg_ExecuteTriggerAndSetVariablesWithResponse_
InputMessage">
    <wsdl:part name="text" type="xsd:string"/>
    <wsdl:part name="variableData" type="xsd:string"/>
    <wsdl:part name="wait" type="xsd:boolean"/>
</wsdl:message>
<wsdl:message name="WebSrviTrg_ExecuteTriggerAndSetVariablesWithResponse_
OutputMessage">
    <wsdl:part name="ExecuteTriggerAndSetVariablesWithResponseResult"
type="xsd:int"/>
    <wsdl:part name="responseData" type="xsd:base64Binary"/>
```

```
    <wsdl:part name="errorText" type="xsd:string"/>
</wsdl:message>
```

There are three input variables (you provide their values):

- **text:** Filter in configuration parses this input string. Usually, the input string is structured as CSV or XML file which makes it easy to parse. You can also use any other textual file format.
- **wait:** This is a Boolean field that specifies two things:
  - If you wish to wait for the print job status response or not.
  - If the Web Service should provide feedback or not.

  In case of *True*, use **1**. In case of *False*, use **0**. Depending on the selected method type, there is either a predefined response, or you can send a customized response.

- **variableData:** This is the string that contains *name:value* pairs. The trigger reads all pairs and assigns provided values to the trigger variables of the same name. If the variable doesn't exist in the trigger, the trigger discards that *name:value* pair. If you provide the list of variables and their values using this method, you don't have to define any data extraction with filters. The trigger does all the parsing for you.
  There are two available structures for the variableData content.

  XML structure
  The trigger provides variables within the **<Variables />** root element of the XML file. Variable name includes attribute name, while the variable value includes element value.

```
<?xml version="1.0" encoding="utf-8"?>
<Variables>
    <variable name="Variable1">Value 1</variable>
    <variable name="Variable2">Value 2</variable>
    <variable name="Variable3">Value 3</variable>
</Variables>
```

NOTE
Embed your XML data inside the CDATA section. **CDATA**, meaning **character data**, is a section of element content that is marked for the parser to help it interpret the XML data as character-only data, and not as markup. As a result, the trigger handles the entire content as character data. For example, **<element>ABC</element>** gets interpreted as **<element>ABC</element>**. Each CDATA section starts with **<![CDATA[** sequence and ends with the **]]>** sequence. To sum up, simply insert your XML data inside these two sequences.

  Name-value pairs
  Trigger provides variables using a text stream. Every *name:value* pair comes in its own line. Variable name is to the left of the equals character (=), variable value is to the right.

```
Variable1="Value 1"
Variable2="Value 2"
Variable3="Value 3"
```

These are the optional output variables:

> **NOTE**
> You receive their values, if you request them by setting **wait** to **1**:

- **ExecuteTriggerAndSetVariablesWithResponseResult:** Integer response contains value 0 if there are no problems during data processing. It contains an integer greater than 0 if the response reports error(s). The application executing the Web Service call to NiceLabel Automation can use the response as error indicator.
- **responseData:** Custom response that you can define in the Web Service trigger configuration. The response is base64-encoded data.
- **errorText:** This string value contains print job status response in case it reports a trigger processing error.

  > **NOTE**
  > In case of trigger processing error, this element is included in the XML response message. Its value contains error description. However, if there is no error, this element is not included in the XML response.

Other

Options in the **Feedback from the Print Engine** section specify communication parameters that allow you to receive print engine feedback.

> **PRODUCT LEVEL INFO**
> This functionality is available in **LMS Enterprise** and **LMS Pro**.

- **Supervised printing:** Activates synchronous printing mode. Use it whenever you want to send the print job status back to the third party application. For more information, see section Synchronous Print Mode.

Options in the **Data Processing** section specify if you want to trim the data so that it fits into a variable, or ignore the missing label variables. By default, reports errors and breaks the printing process if you try to save values that are too long in label variables, or try to set values for non-existing label variables.

- **Ignore excessive variable contents:** truncates data values that exceed the length of the variable as defined in the label designer to make them fit. This option is in effect if you are setting variable values in filters, from command files, and when you are setting values of trigger variables to label variables of the same name.

Example

Label variable accepts 5 characters at maximum. With this option enabled, any value longer than 5 characters is truncated to the first 5 characters. If the value is 1234567 ignores digits 6 and 7.

- **Ignore missing label variables:** When printing with command files (such as JOB file), the printing process ignores all variables that are:
  - specified in the command file (using the SET command)
  - not defined on the label

  Similar happens if you define assignment area in a filter to extract all name-value pairs, but your label contains fewer variables.
  When setting values of non-existing label variables, reports an error. If this option is enabled, the printing continues.

Options in **Scripting** section specify scripting possibilities.

- **Scripting language:** Selects scripting language for the trigger. All **Execute script** actions that you use within a single trigger use the selected scripting language.

Options in the **Save Received Data** section specify the available commands for data that the trigger receives.

- **Save data received by the trigger to file:** Enable this option to save the data received by the trigger. The option **Variable** enables variable file name. Select a variable that contains path and file name.
- **On error save data received by the trigger to file:** Enable this option to save the data into the trigger only if an error occurs during the action execution. You might want to enable this option to keep the data that caused the issue ready for troubleshooting.

NOTE

Make sure you enable the Supervised printing support. If not, cannot detect errors during the execution. For more information, see section Synchronous Print Mode.

NOTE

saves the received data into a temporary file. This temporary file is deleted right after the trigger execution completes. The internal variable **`DataFileName`** points to that file name. For more information, see Internal Variables.

Security

- **Lock and encrypt trigger:** Enables trigger protection. If you enable it, the trigger becomes locked and you cannot edit it any longer. This encrypts the actions. Only users with password can unlock the trigger and modify it.

3.8. Cloud Trigger

PRODUCT LEVEL INFO

Requires **Label Cloud** subscription.

To learn more about triggers in general, see section Understanding Triggers.

Read more about NiceLabel Label Cloud here.

Cloud trigger allows you to integrate your NiceLabel Cloud or on-premise Control Center with existing business systems that run in private clouds or in dedicated data centers. If an existing business system (e.g., SAP S/4HANA or Oracle NetSuite) produces an output, a cloud-hosted API enables you to send HTTP requests to the cloud trigger.

The cloud trigger allows you to locally print labels whose content originates from the cloud information systems. Because the cloud trigger that runs on the local Automation server uses standard methods for accessing the cloud-based services, you can deploy local printing in a secure and time efficient way.

The cloud trigger enables a secure and transparent way to integrate your local label printing using applications that communicate over the open Internet.

Compared to the HTTP server trigger, the cloud trigger does not require you to open any inbound ports on your firewall. The cloud trigger uses a dedicated NiceLabel API that runs in the cloud. This is why the trigger only requires open outbound port 443, or ports 9350-9354. In most cases, these ports are already open.

When deploying the cloud trigger, you have two options:

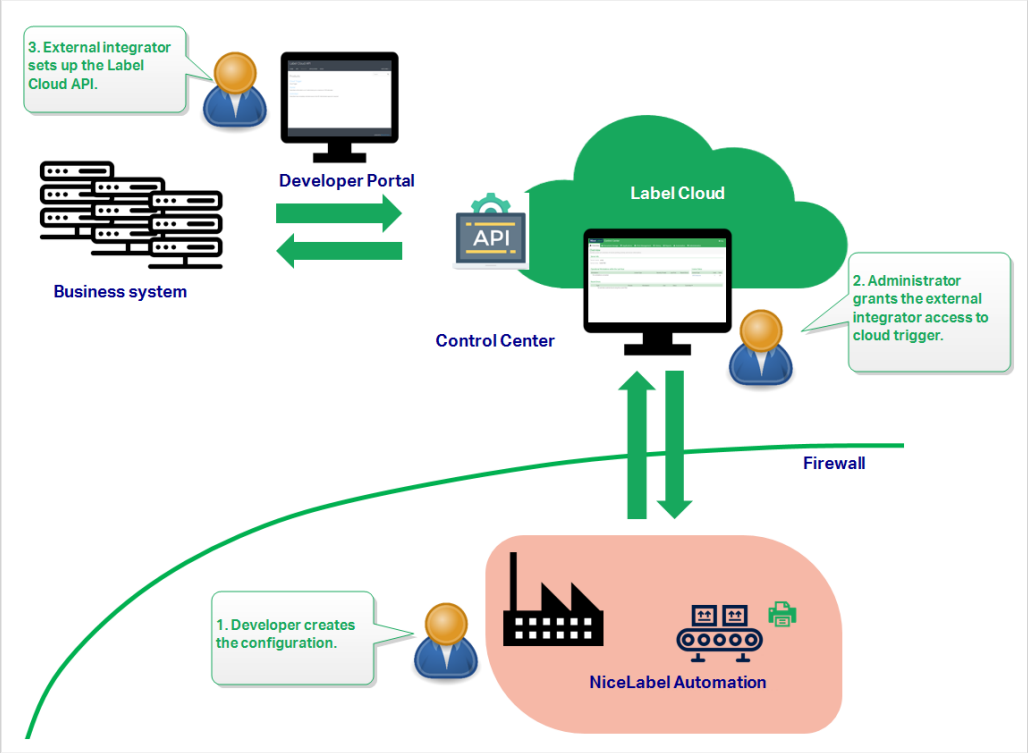
- You can deploy the trigger in your NiceLabel Cloud.
- You can deploy the trigger in your on-premise Control Center that either runs locally on your servers on in private cloud infrastructure.

Both options are equal in terms of offered functionalities. Chose your preferred option based on your available infrastructure.

3.8.1. Deploying Cloud Trigger with NiceLabel Cloud

3.8.1.1. Deployment Stages for NiceLabel Cloud

To enable local label printing using the cloud trigger deployed in NiceLabel Cloud, you must establish cooperation between users with three roles: a user that configures the cloud trigger on the local Automation server (developer), a user that sets up the cloud trigger in the NiceLabelNiceLabel Cloud, and a user that makes the subscription on the Developer Portal.



1. The **developer** configures and deploys the cloud trigger configuration on the local Automation server using Automation Builder and Automation Manager.

   > **NOTE**
   > NiceLabelAutomation must be signed in to the NiceLabel Cloud.

   See section Configuring cloud trigger in Automation Builder for details.

2. The **NiceLabel Cloud administrator** gives the external integrator access to the cloud trigger in Control Center. When done, the NiceLabel Cloud administrator sends the external integrator the corresponding integrator key.
   See section Setting up Cloud Trigger Access for the External integrator for details.

3. The **external integrator** joins the Developer Portal to bring the customer's business system and the NiceLabel Cloud together.

   > **NOTE**
   > The Developer Portal hosts the dedicated API called **Cloud Trigger**. This API serves as an interconnection point between the events that take place in the customer's business system, and the Automation configuration that runs locally.

> **NOTE**
> The word "external" means that this user's role is to make the subscription on the Developer Portal. The created subscription authenticates the integrator. External integrators are not necessarily outside collaborators. They can be in-house integrators who belong to the company's own development team.

The external integrator performs the following actions on the Developer Portal:

a. Signing in to the Developer Portal. Before the first sign-in, the integrator must also finish the sign up procedure.

b. Creating a subscription for the Cloud Trigger API.

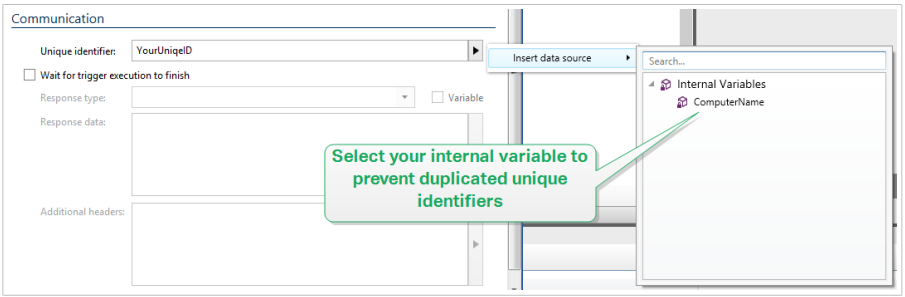
c. Connecting the subscription with the integrator key. This is how the subscription gains access to the customer's cloud triggers.
For details, see section Setting up the NiceLabel Cloud API on the Developer Portal for details.

3.8.1.2. Configuring cloud trigger in Automation Builder

This section describes how to configure the cloud trigger in Automation that runs on your local server.

1. Open your Automation Builder. Make sure the Automation Builder is signed in to NiceLabel Cloud. Go to **File** > **About** > **NiceLabel Cloud** account to confirm that you are signed in.
2. The **Configuration Items** tab opens. Click **Cloud Trigger** to create a new configuration for the cloud trigger.
3. Set **Name** and **Description** to make your cloud trigger easy to find among other triggers.
4. Set the trigger **Communication** settings:
   - Define the **Unique identifier**. After you deploy the trigger, this unique identifier is necessary for calling the trigger.
     If you are running the cloud trigger configuration on multiple computers, you must make sure each computer automatically uses its own unique identifier. To prevent unwanted duplications, insert internal variables as part of the **Unique identifier**. You can use two internal variables for this purpose:
     - **ComputerName**: The name of the computer on which the configuration is running.
     - **SystemUserName**: The Windows user name of the currently logged-in user.

     To insert internal variables to the **Unique identifier**, click Insert data source and select your internal variables.



- **Wait for trigger execution to finish:** The HTTP protocol requires a receiver (in this case NiceLabel Automation) to send a numeric response back to the sender indicating the status of the received message. By default, NiceLabel Automation responds with code 200. This indicates Automation successfully received the data, but gives no information about the success of trigger actions.
  This option specifies that a trigger doesn't send a response immediately after receiving the data, but waits until all actions execute. Then, it sends the response code indicating a successful action execution. With this option enabled, you can send back a custom response type and data (e.g., the response to a HTTP request is label preview in PDF format).
  With cloud trigger, the relevant built-in Automation standard HTTP response codes are:

HTTP Response Code	Description
200	All actions executed successfully.
500	There were errors during action execution.

NOTE

To send feedback to Automation about the print process, enable **synchronous** print mode. For more information, see section Synchronous Print Mode.

- **Response type:** Specifies the type of your response message. Frequently used Internet media types (also known as MIME types, or Content-types) are predefined in the drop down box. If your media type is not available in the list, type it yourself. Automation sends the response data outbound as feedback, formatted in the defined media type. **Variable** enables variable media types. If enabled, select or create a variable that contains media type.

NOTE

If you don't specify the Content-Type, NiceLabel Automation uses **`application/octet-stream`** as a default.

- **Response data:** Defines the content of your response message. Examples of what you can send back as an HTTP response are custom error messages, label previews, generated PDF files, print stream (spool) files, XML files with details from the print engine plus the label preview (encoded as a Base64 string).
  If your output consists of binary-only content (such as label preview or print stream), make sure you select a supported media type, e.g. **`image/jpeg`** or **`application/octet-stream`**.

- **Additional headers**: Allow you to define custom MIME headers for the HTTP response message.
  Response header syntax and examples are available in the HTTP Request action section.

TIP

With **Response data** and **Additional headers**, you can use fixed content, mix of fixed and variable content, or variable content alone. To insert variable content, click the button with an arrow to the right of the data area and insert your variable from the list. You can also create a new variable that contains the data you want to use. For more information, see section Using Compound Values.

5. Deploy and start the trigger in Automation Manager. The cloud trigger now monitors incoming requests.

NOTE

If your configuration requires increased availability and scalability, you can deploy multiple identical cloud triggers. To do this, install multiple instances of Automation, and deploy the cloud triggers on them. If the deployed cloud triggers share the same **Unique identifier**, the built-in load balancer in NiceLabel Cloud automatically distributes the traffic load among them.

3.8.1.3. Setting up Cloud Trigger Access for the External integrator

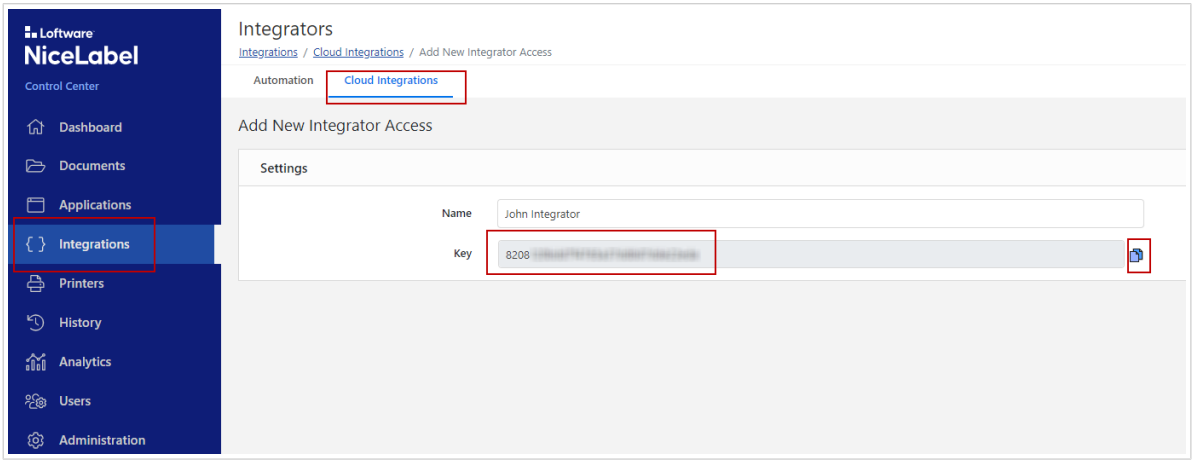
NOTE

To set up the integrator's access to the cloud trigger, you must have the **Manage cloud integration** privilege on the cloud Control Center. See the Control Center user guide for details about managing the user privileges.

1. Go to your cloud Control Center. Open the web browser and type:

   ```
   https://<yourlabelcloudname>.onnicelabel.com/dashboard.
   ```

2. Go to **Integrations** > **Cloud Integrations** .
3. Click **+Add**. This opens the **Add New Integrator Access** page.
4. Type the **Name** of the integrator you are currently adding.



5. Copy the **Key**.
6. Click **Save**.
7. Direct the external integrator to the Developer Portal. Send the external integrator the following information:
   - Link to the NiceLabel Cloud API: https://developerportal.onnicelabel.com/
   - The integrator key (see step 5).
   - The trigger unique identifier. You can find this **Unique identifier** in the cloud trigger Automation configuration settings (see step 4 in section Configuring cloud trigger in Automation Builder.

> **NOTE**
> External integrator needs the key to authenticate themselves for calling the customer's cloud trigger.

> **NOTE**
> For more details, read the Cloud Triggers section of your Control Center user guide.

3.8.1.4. Creating subscription on the Developer Portal

After receiving the required information from the NiceLabel Cloud administrator, the external integrator must first sign up to the Developer Portal, and create subscriptions (per each customer) for calling the triggers. These trigger calls originate from the customers' cloud-based information systems.

> **NOTE**
> If you do not see the Developer Portal email in your Inbox, check your junk folder.

NOTE

Customers are companies that run cloud trigger configurations that receive the data from external information systems.

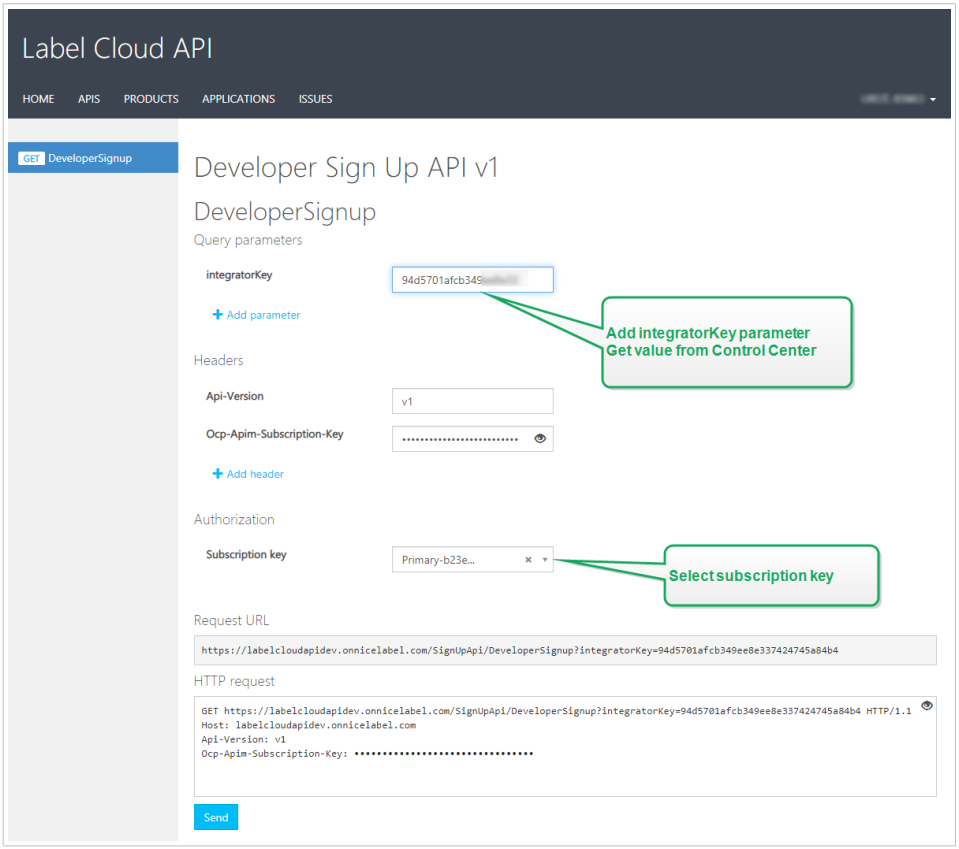
NOTE

Each integrator may call multiple cloud triggers using a single subscription.

1. Open your browser and go to https://developerportal.onnicelabel.com/
2. To complete the sign up procedure, follow the on-screen instructions. After you click **Sign up**, you receive a confirmation email. Click the confirmation link to activate your Developer Portal account.
3. Open the **Products** tab and click **Label Cloud**. The page that you land on contains your APIs and existing subscriptions.
4. Click **Add subscription**. The **Subscribe to product page** opens.

   NOTE

   You can create multiple subscriptions. However, a single subscription can be used for a single customer. This is why NiceLabel recommends you to include the name of the customer to the **Subscription name**, such as `Cloud Trigger Example Customer`.
5. Type the **Subscription name**.
6. Click **Confirm**. The newly created subscription becomes available on **Products** > **Label Cloud**.
7. When again on the **Label Cloud** page, click the **Developer Sign Up API v1**.
8. Click **Try it**. The API page opens.



9. Insert the **Integrator Key** from the customer's cloud Control Center.
10. Click **Send**.
    - The response is: `Subscription <your subscription key> successfully associated with integrator key <integrator key value>`.

NOTE

You received the integrator key from the NiceLabel Cloud administrator. The key looks like this: `979d7be5df2b473193ac5519f94cd901`

Example

When passing the integration key as a query parameter, the URL looks like this: `https://labelcloudapi.onnicelabel.com/SignUpApi/DeveloperSignup?integratorKey=979d7be5df2b473193ac5519f94cd901.`

Once you make this call using the URL as shown in the example, the **DeveloperSignup** operation matches the subscription with the assigned customer. This is how the integrator authenticates themselves when calling the cloud trigger that runs in the customer's Automation.

The connected subscriptions are also visible in the cloud Control Center. Check if the external integrator has connected their subscription under **Integrations** > **Cloud integrations**. The cloud integration should have the status **Developer [name, email address]** subscribed.

3.8.1.5. Calling your cloud trigger (NiceLabel Cloud deployment)

This step makes sure that the outputs of the external business systems successfully execute cloud triggers that run locally. This is the purpose of the **CloudTrigger** operation. In the URL of the call, specify the trigger name you are calling.

To call a trigger with the unique identifier `MyCloudTrigger`, use this URL:

`https://labelcloudapi.onnicelabel.com/TriggerApi/CloudTrigger/MyCloudTrigger`

For each event (output) in the external business system, call the URL as shown in the example. Each call executes the cloud trigger that runs on the local Automation server.

All API calls must include these two headers:

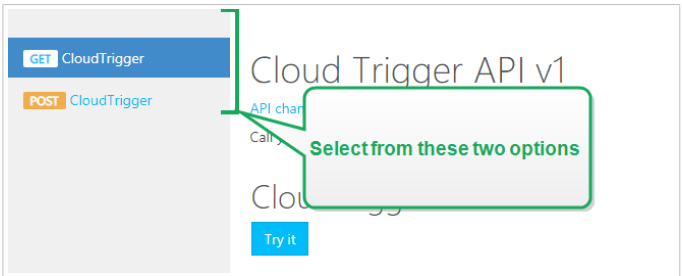
- **Api-Version** is the version of the API you are using. Currently, the only available API version is v1.
- **Ocp-Apim-Subscription-Key** is the key that identifies your subscription.

Testing CloudTrigger calls

To become familiar with how the **CloudTrigger** calls work, the Developer Portal allows you to test such calls.

Before you can test this call, you must set up a running Automation configuration.

1. Open the **Development Portal**, open the **Products** tab, and click **Label Cloud**.
2. Select **Cloud Trigger API v1**.
3. Create a sample for GET or PUT methods. Click the appropriate link.
    - After you make the method selection, click **Try it**. A new page opens. The **triggerID** is already added under the **Query parameters**.
    - In the **Value** field, copy and paste the **triggerID** that you receive from the developer of the Automation configuration. This is the trigger's **Unique identifier**. The **Unique identifier** is available in**Automation Builder**> **Trigger Settings** > **General**.



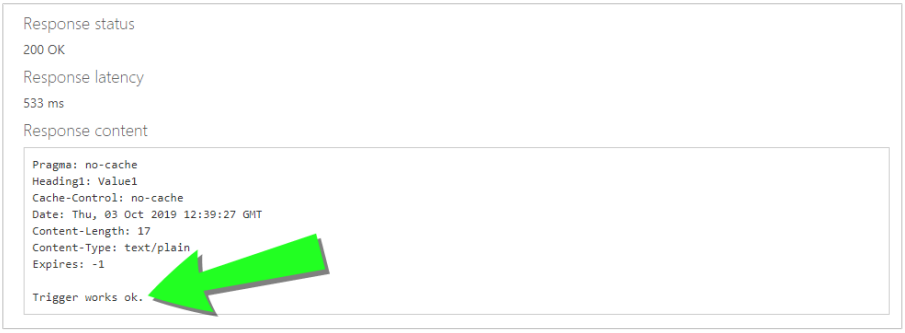
4. Under **Authorization**, select the subscription key. Because you already have at least one subscription defined, the drop-down list already contains the key for the defined subscription. Select this key – either primary or secondary.
5. Click **Send**.

- **Response status** is 200 OK.

3.8.1.6. Quick Check if your Cloud trigger works

Once you have set up your NiceLabel Cloud API on the Developer Portal, you can make a simple configuration in Automation Builder to check if the cloud trigger works. If the trigger works, you receive the "Trigger works ok." message on the Cloud Trigger API page after you click **Try it**.

1. Open Automation Builder and create a new configuration. Make sure your Automation Manager is connected to the NiceLabel Cloud.
2. Add a new **Cloud trigger**.
3. Define **Name** and **Description**, and set the **Unique identifier**. Let's use the `TestCloudTrigger` Unique identifier in this case.
4. Enable **Wait for trigger execution to finish**. This allows you to track trigger responses.
   - Select **text/plain** as **Response type**.
   - Define the **Response data**. This is what you receive if the trigger works. Let's use the following string: "Trigger works ok."
   - Define the **Additional headers**. Use the `Heading:Value` format.
5. Deploy the configuration.
6. Open your developer portal and go to your **Cloud Trigger API v1** page.
7. Click **Try it**.
8. Paste `TestCloudTrigger` to the **triggerID** field. Click **Send**.
   - Response content includes the confirmation: "Trigger works ok."



3.8.2. Deploying Cloud Trigger with your on-premise Control Center

3.8.2.1. Configuring cloud trigger in Automation Builder

This section describes how to configure the cloud trigger in Automation that runs on your local server.

1. Open your Automation Builder. Make sure the Automation Builder is paired with your Control Center. To check, go to **File** > **Options** >**Control Center**and see if the URL address of your Control Center is there.

   > **NOTE**
   > "Paired" Automation Builder and Control Center also means that both applications use the same license key.

2. The **Configuration Items** tab opens. Click **Cloud Trigger** to create a new configuration for the cloud trigger.

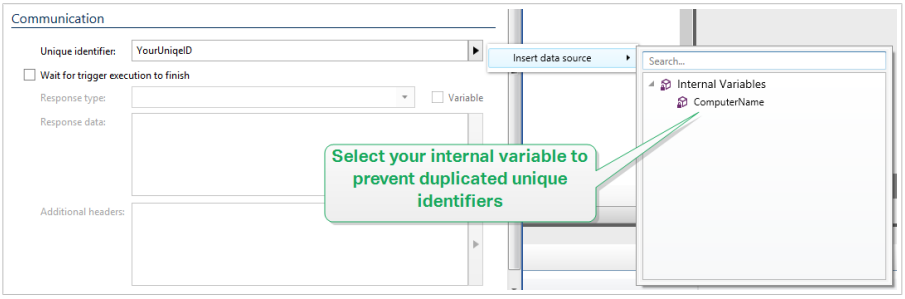
3. Set **Name** and **Description** to make your cloud trigger easy to find among other triggers.

4. Set the trigger **Communication** settings:

   - Define the **Unique identifier**. After you deploy the trigger, this unique identifier registers the trigger on your Control Center. Use only alphanumeric characters. Special characters are not permitted.
     If you are running the cloud trigger configuration on multiple computers, you must make sure each computer automatically uses its own unique identifier. To prevent unwanted duplications, insert internal variables as part of the **Unique identifier**. You can use two internal variables for this purpose:

     - **ComputerName**: The name of the computer on which the configuration is running.
     - **SystemUserName**: The Windows user name of the currently logged-in user.

     To insert internal variables to the **Unique identifier**, click Insert data source and select your internal variables.

   - **Wait for trigger execution to finish:** The HTTP protocol requires a receiver (in this case NiceLabel Automation) to send a numeric response back to the sender indicating the status of the received message. By default, NiceLabel Automation responds with code 200. This indicates Automation successfully received the data, but gives no information about the success of trigger actions.
     This option specifies that a trigger doesn't send a response immediately after receiving the data, but waits until all actions execute. Then, it sends the response code indicating a successful action execution. With this option enabled, you can send back a custom response type and data (e.g., the response to a HTTP request is label preview in PDF format).
     With cloud trigger, the relevant built-in Automation standard HTTP response codes are:

HTTP Response Code	Description
200	All actions executed successfully.
400	No configuration available.
500	There were errors during action execution.

NOTE

To send feedback to Automation about the print process, enable **synchronous** print mode. For more information, see section Synchronous Print Mode.

- **Response type:** Specifies the type of your response message. Frequently used Internet media types (also known as MIME types, or Content-types) are predefined in the drop down box. If your media type is not available in the list, type it yourself. Automation sends the response data outbound as feedback, formatted in the defined media type. **Variable** enables variable media types. If enabled, select or create a variable that contains media type.

NOTE

If you don't specify the Content-Type, NiceLabel Automation uses `application/octet-stream` as a default.

- **Response data:** Defines the content of your response message. Examples of what you can send back as an HTTP response are custom error messages, label previews, generated PDF files, print stream (spool) files, XML files with details from the print engine plus the label preview (encoded as a Base64 string).
  If your output consists of binary-only content (such as label preview or print stream), make sure you select a supported media type, e.g. `image/jpeg` or `application/octet-stream`.

- **Additional headers**: Allow you to define custom MIME headers for the HTTP response message.
  Response header syntax and examples are available in the HTTP Request action section.

TIP

With **Response data** and **Additional headers**, you can use fixed content, mix of fixed and variable content, or variable content alone. To insert variable content, click the button with an arrow to the right of the data area and insert your variable from the list. You can also create a new variable that contains the data you want to use. For more information, see section Using Compound Values.

5. Deploy and start the trigger in Automation Manager. The cloud trigger now monitors for the incoming requests.

NOTE

If your configuration requires increased availability and scalability, you can deploy multiple identical cloud triggers. To do this, install multiple instances of Automation, and deploy the cloud triggers on them. If the deployed cloud triggers share the same **Unique identifier**, the built-in load balancer in NiceLabel Cloud automatically distributes the traffic load among them.

3.8.2.2. Calling the cloud trigger (On-premise deployment)

This step makes sure that the outputs of the external business systems successfully execute cloud triggers that run locally. This is the purpose of the **CloudTrigger** operation. In the URL of the call, specify the trigger name you are calling.

To call the trigger with your unique identifier `MyCloudTrigger`, call this URL:

```
https://<YourServerName>/epm/api/trigger/<MyCloudTriggerID>
```

NOTE

The URL might start with "`http`" or "`https`" – depending on how you set up your Control Center during the installation. See the Control Center installation guide, sections Setting up website and storage for details.

For each event (output) in the external business system, call the URL as shown in the example. Each call executes the cloud trigger that runs on the local Automation server.

All calls must include the header named **Integrator-Key**.

Example

```
Integrator-Key: 9d59d7d444da412b8acfb488a01bb632
```

3.9. Scheduler Trigger

To learn more about triggers in general, see section Understanding Triggers.

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

Scheduler trigger works as a timer that starts the execution of actions in your configuration after a time period. Use the scheduler trigger to set up repetitive automated execution of time-dependent actions.

Scheduler trigger is an active trigger. This means that it does not wait for an event change, but starts executing the assigned actions as soon as the defined time interval ends.

Example

Typical usage: An ERP system produces 6000 packaging label files daily. Automation prints the labels, and stores the used label files in a dedicated directory. Due to a large number used label files, the company needs to set up automated purging of files that are older than 48 hours. Automation uses the scheduler trigger to delete the obsolete label files.

Scheduler trigger automatically takes daylight saving time (DST) clock changes into account. The trigger always takes current system time as a reference:

- On the DST starting date (summer), the trigger executes at the first hour that is past the hour of the missing defined time.

  ### Example

  Scheduler trigger time is set for 2.30 h. Clock jumps from 2.00 h to 3.00 h. The trigger executes at 3.00 h summer time.

- On the DST ending date (winter), the trigger executes only at the first occurrence of the defined time.

  ### Example

  Scheduler trigger time is set for 2.30 h. Clock jumps from 3.00 h to 2.00 h. The trigger has already executed. It does not activate the actions again when the clock is 2.30 h winter time.

3.9.1. General

This section allows you to configure the most important trigger settings.

- **Name:** Specifies the unique name of the trigger. The names helps you distinguish between different triggers when you configure them in Automation Builder, and later when you run them in Automation Manager.
- **Description:** Allows you to describe the role of this trigger. Help the users with a short explanation about what the trigger does.

3.9.2. Recurrences

Use recurrences to define how often should the scheduler trigger event repeat.

- **Execute trigger**: sets the trigger repetition time interval.

- **Every (X) seconds/minutes/hours** sets the trigger event repetition time in the available time units. Define the interval length under **Seconds/Minutes/Hours (X)**.
- **Daily** repeats the trigger event every day at the selected time. Define the time of daily repetition under **Time:**.
- **On specific days** repeats the trigger at the defined time of the selected day(s). Set the repetition using **Time:** and days.

> **NOTE**
> Define the time values in 24-hour format.

Other

Options in the **Feedback from the Print Engine** section specify communication parameters that allow you to receive print engine feedback.

> **PRODUCT LEVEL INFO**
> This functionality is available in **LMS Enterprise** and **LMS Pro**.

- **Supervised printing:** Activates synchronous printing mode. Use it whenever you want to send the print job status back to the third party application. For more information, see section Synchronous Print Mode.

Options in the **Data Processing** section specify if you want to trim the data so that it fits into a variable, or ignore the missing label variables. By default, reports errors and breaks the printing process if you try to save values that are too long in label variables, or try to set values for non-existing label variables.

- **Ignore excessive variable contents:** truncates data values that exceed the length of the variable as defined in the label designer to make them fit. This option is in effect if you are setting variable values in filters, from command files, and when you are setting values of trigger variables to label variables of the same name.

> **Example**
>
> Label variable accepts 5 characters at maximum. With this option enabled, any value longer than 5 characters is truncated to the first 5 characters. If the value is 1234567 ignores digits 6 and 7.

- **Ignore missing label variables:** When printing with command files (such as JOB file), the printing process ignores all variables that are:
  - specified in the command file (using the SET command)
  - not defined on the label

  Similar happens if you define assignment area in a filter to extract all name-value pairs, but your label contains fewer variables.

When setting values of non-existing label variables, reports an error. If this option is enabled, the printing continues.

Options in **Scripting** section specify scripting possibilities.

- **Scripting language:** Selects scripting language for the trigger. All **Execute script** actions that you use within a single trigger use the selected scripting language.

Options in the **Save Received Data** section specify the available commands for data that the trigger receives.

- **Save data received by the trigger to file:** Enable this option to save the data received by the trigger. The option **Variable** enables variable file name. Select a variable that contains path and file name.
- **On error save data received by the trigger to file:** Enable this option to save the data into the trigger only if an error occurs during the action execution. You might want to enable this option to keep the data that caused the issue ready for troubleshooting.

NOTE

Make sure you enable the Supervised printing support. If not, cannot detect errors during the execution. For more information, see section Synchronous Print Mode.

NOTE

saves the received data into a temporary file. This temporary file is deleted right after the trigger execution completes. The internal variable **`DataFileName`** points to that file name. For more information, see Internal Variables.

Security

- **Lock and encrypt trigger:** Enables trigger protection. If you enable it, the trigger becomes locked and you cannot edit it any longer. This encrypts the actions. Only users with password can unlock the trigger and modify it.

3.10. Testing Triggers

After setting up the triggers, you are only halfway done with the configuration. Before deploying the trigger, test it thoroughly for its intended operation upon incoming data and verify the execution of actions.

Automation Builder enables you to test the configuration while you are still working on it. Some actions have built-in test capabilities so you can focus on the execution of individual action. You can also test every triggers with Run Preview command. However, the final test should always be done in the real environment, providing real data and using real triggers. This is how you monitor trigger execution using Automation Manager.

Testing execution of individual actions

Some of the actions include preview option. This allows you to change input parameters and to see the result of the action on-screen.

- **Use Data Filter:** The action provides live preview of the parsed data. The rules in the selected filter are applied to the selected input data file. The result is shown in the table. If you use sub- or assignment areas, you can see the preview for every level of filter definition.
- **Execute SQL Statement:** The action allows you to preview the execution of the defined SQL statement. You can monitor the data set resulting from the SELECT statement, and number of rows affected by the UPDATE, INSERT and DELETE statements. The preview execution is transaction-safe, which means that you can roll-back all changes. You can change the input query parameters and see how they affect the result.
- **Web Service:** The action allows you to preview the execution of a selected method (function) from Web Service. You can change the input parameters and see how they affect the result.
- **Execute script:** The action checks for syntax errors in the provided script, and also executes it. You can change the input parameters and see how they affect the script execution.

Testing trigger execution and displaying label preview on-screen

To test the trigger from the ground up, use the built-in **Run Preview** functionality. You can run preview for every trigger, no matter its type. The trigger won't fire upon changes of the monitored event. Only a trigger started in the Automation Manager can do it. Instead, the trigger is going to execute actions based on the data saved in a file. Make sure you have a file that contains sample data that the trigger is going to accept in real-time deployment.

The trigger executes all defined actions (including data filtering) and displays label preview(s) on-screen. The preview simulates the printing process in every detail. The labels would print with the same composition and contents as shown in the preview. This includes the number of labels and their contents. You also learn about how many print jobs are produced, how many labels are in each job and see preview of each label. You can navigate from one label to the next in the selected print job.

Log pane displays the same information as it would be displayed in the Automation Manager. Expand the log entries to see full detail.

NOTE

After you run the preview, all actions defined for the selected trigger are run, not just the Run Preview action. Be careful when using actions that modify the data, such as Execute SQL Statement or Web service, because their execution is irreversible.

To preview the labels, do the following:

1. Open the trigger configuration.
2. Make sure the trigger configuration is saved.
3. Click the button **Run Preview** in Preview group in the ribbon.
4. Browse for the data file providing the typical contents that trigger will accept.
5. See the result in a Preview tab.

Testing deployment on pre-production server

It is advisable to deploy the configuration to Automation Manager on pre-production server before deploying it on production server. Testing in pre-production environment might identify additional configuration issues that have not been detected during trigger testing in Automation Builder alone.

Configuration performance can also be stress-tested by adding load to the trigger and monitoring how it performs. The testing provides important information about the available throughput and identifies weak points. Based on the conclusions, you can implement various system optimization techniques, such as optimizing label design to produce smaller print streams, and optimizing the overall flow of data from the existing application into NiceLabel Automation.

Important differences between real trigger testing and previewing in Automation Builder
While previewing the trigger on-screen in Automation Builder provides a quick method of trigger testing, you must not rely on it alone. There can be execution differences between previewing and running the trigger for real when you use 64-bit Windows.

Even if you have your configuration working in Automation Builder, make sure to run in for real using the Service as well.

- When you run command **Run Preview**, the configuration executes in Automation Builder, which always runs as 32-bit application. Previewing your trigger in Automation Builder only tests execution on a 32-bit platform.
- When you run triggers for real, the configuration executes in Service, which runs as 32-bit application on 32-bit Windows, and runs as 64-bit application on 64-bit Windows. For more information see section Running in Service Mode.
- Issues might arise if trigger processing is affected by platform differences (32-bit vs 64-bit):
  - **Database access:** 64-bit applications require 64-bit database drivers, and 32-bit applications require 32-bit drivers. To run configuration from Automation Builder and in the Service, you need 32-bit and 64-bit database drivers to access your database. For more information, see section Accessing Databases.
  - **UNC syntax for network files:** The service account cannot access network shared files with mapped drive letter. You have to use UNC syntax for network files. For example, use `\\server\share\files\label.nlbl` and not `G:\files\label.nlbl`, where G: is mapped to `\\server\share`. For more information see section Access to Network Shared Resources.
- If your NiceLabel Automation Service runs under a different user account that you are using for Automation Builder, the accounts might not have the same security privileges. If you can open the label in Automation Builder, the user account for the Service might not be able to access it. To run Automation Builder under the same user account as the Service, see Using the same user account to configure and to run triggers.

3.11. Using Secure Transport Layer (HTTPS)

PRODUCT LEVEL INFO
The functionality from this section is available in LMS Enterprise.

You can protect the inbound traffic to the HTTP Server Trigger and Web Service Trigger by enabling the HTTPS support. HTTPS secures the transmission of the messages exchanged over the network. The communication security uses X.509 certificates to encrypt the data flowing between the parties. Your information remains confidential from prying eyes because just the client and the NiceLabel Automation can decrypt the traffic. Even if some unauthorized user does eavesdrop on the communication he would

fail to understand the meaning of the messages, because the traffic appears as a stream of random bytes.

It makes a good security practice to encrypt the communication in cases, such as:

- You work with the sensitive and confidential data that must not be exposed to 3rd party users.
- The message must pass through networks that are outside of your control. For example, this happens when you send data to Automation over the Internet, and not from the local network.

Enabling the secure transport layer (HTTPS)

To enable secure transport for your trigger, do the following.

In the Windows system:

1. Obtain the X.509 certificate from the issuer of the digital certificates (certificate authority - CA). You need a certificate type for the 'server authentication'.

   > **NOTE**
   > If you will self-generate the certificate, make sure to import the CA certificate in the Trusted Authority store, so the CA signature can be verified on the server certificate.

2. Install the X.509 certificate in the system, where NiceLabel Automation is installed. Make sure the certificate is visible to the user account under which you run NiceLabel Automation service. It is a good practice to install the certificate in the local computer store, not the current user store. This allows NiceLabel Automation to use the certificate even if it is not running under your current logged-in user account.

   a. Open a Command Prompt window.

   b. Type **mmc** and press the ENTER key (make sure you are running it with administrative privileges).

   c. On the File menu, click **Add/Remove Snap In**.

   d. In the **Add Standalone Snap-in** dialog box, select **Certificates**.

   e. Click **Add**.

   f. In the **Certificates snap-in** dialog box, select **Computer account** and click **Next**.

   g. In the **Select Computer** dialog box, click **Finish**.

   h. On the **Add/Remove Snap-in** dialog box, click **OK**.

   i. In the Console Root window, expand **Certificates>Personal**.

   j. Right-click Certificates folder and select **All Tasks>Import**.

   k. Follow the wizard to import the certificate.

3. Retrieve the thumbprint of a certificate you have just imported.

   a. While still in the MMC double-click the certificate.

   b. In the **Certificate** dialog box, click the **Details** tab.

   c. Scroll through the list of fields and click Thumbprint.

   d. Copy the hexadecimal characters from the box. Remove the spaces between the hexadecimal numbers. For example, the thumbprint "a9 09 50 2d d8 2a e4 14 33 e6 f8 38 86 b0 0d 42 77 a3 2a 7b" should be specified as "a909502dd82ae41433e6f83886b00d4277a32a7b" in code. This is `certhash` required in the next step.

4. Bind the certificate to the IP address and port where the trigger is running. This action enables certificate on the selected port number.
   Open the **Command Prompt** (make sure you are running it with the administrative privileges) and run the following command:

```
netsh http add sslcert ipport=0.0.0.0:56000
certhash=7866c25377554ca0cb53bcdfd5ee23ce895bdfa2
appid={A6BF8805-1D22-42C2-9D74-3366EA463245}
```

   where:

   - **`ipport`** is the IP address-port pair, where the trigger is running. Leave the IP address at 0.0.0.0 (local computer), but change the port number to match port number in the trigger configuration.
   - **`certhash`** is the thumbprint (SHA hash) of the certificate. This hash is 20 bytes long and is specified as a hex string.
   - **`appid`** is GUID of the owning application. You can use any GUID here, even the one from the sample above.

In the trigger configuration:

1. In your HTTP or Web Service trigger enable the option **Secure connection (HTTPS)**.
2. Reload the configuration in the Automation Manager.

Disabling the secure transport layer (HTTPS)

In the Windows system:

- Unbind the certificate from the IP address-port pair. Run the following command in the Command Prompt (make sure you are running it with the administrative privileges):

```
netsh http delete sslcert ipport=0.0.0.0:56000
```

  where:

  - **`ipport`** is the IP address-port pair, where the trigger is running and where you bound the certificate to.

In the trigger configuration:

1. In your HTTP or Web Service trigger disable the option **Secure connection (HTTPS)**.
2. Reload the configuration in the Automation Manager.

3.12. Protecting Trigger Configuration from Editing

The trigger configuration can be protected using two methods.

- **Locking trigger.** Using this method you lock the trigger configuration file and protect it with a password. Without the password nobody can edit the trigger. Enable the option **Lock and encrypt trigger** in trigger *Settings -> Security*.
- **Setting access permissions.** Using this method you rely on the user permissions as are defined in the NiceLabel Automation Options. You can enable user groups and assign different roles to each group. If the group is assigned with the edit privileges, all members of the group can edit triggers. This method requires that you enable user login. You can use Windows users from local groups or active directory, or you can define NiceLabel Automationusers. See **User rights and access** in Configuration.

3.13. Configuring Firewall for Network Triggers

Network trigger is a trigger that runs using the TCP/IP protocol. In Automation, such triggers are TCP/IP trigger, HTTP trigger and Web Service trigger. These provide network services and are bound to the network interface card, its IP address, and the configured port number. After you deploy and start network triggers in Automation Manager, they start listening to the inbound traffic port.

Firewalls protect computers form unauthorized attempts of incoming connections. NiceLabel installer makes sure that inbound communication streams established to all ports owned by the Automation Service are allowed in Windows Firewall.

WARNING

Automation Service owns ports configured for TCP/IP triggers, but not ports defined for HTTP trigger and Web Service trigger. These ports are bound to ID 4 (SYSTEM) process and not to the Automation Service process.

Configure the firewall to allow communication on ports configured for HTTP and Web Service triggers. To create an inbound rule, do the following:

1. On the computer that is running NiceLabel Automation, in **Start** menu, select **Control Panel**, select **System and Security**, and select **Windows Firewall**.
2. In the navigation pane, select **Advanced settings**.
3. In the **Windows Firewall with Advanced Security** window, in the navigation pane, select **Inbound Rules**, and then in the Actions pane, select **New Rule**.
4. On the **Rule Type** page, select **Port**, and click **Next**.
5. On the **Protocol and Ports** page, select **Specific local ports**, and enter the port number on which your HTTP or Web Service trigger runs.

6. Click **Next**.
7. On the **Actions** page, select **Allow the connection**, and click **Next**.
8. On the **Profile** page, select the profiles, and click **Next**.
9. On the **Name** page, enter a name for the rule, and click **Finish**.

Similar steps must be taken with other firewall software.

4. Variables

Variables are used as containers for data values. You need variables to transfer values to the label in **Print Label** action, or to use values in other data-manipulation actions. Typically, a filter extracts values from data streams received by trigger and sends values to variables. For more information, see section Understanding Filters.

Usually, you send values of variables to the label template, and print the label. The mechanism that sends values of variables to labels uses the automated name mapping. The value of the variable defined in the trigger is sent to the variable defined in the label that carries the same name. You can define variables using one of the three available methods:

- **Import variables from label file:** As for the above explained automatic mapping, it makes a good practice to import your variables from the label every time. This action saves you time and ensures that the variable names match. The imported variable doesn't inherit just the variable name, but also supported variable properties, such as length and default value.
- **Manually define variables:** When manually defining variables, be extra careful to use the same names as with variables on the label. Manually define the variables that don't exist in the label, but you need them inside the trigger.

  > **NOTE**
  > An example would be variables, such as **`LabelName`**, **`PrinterName`**, **`Quantity`** and similar variables that you need to remember the label name, printer name, quantity or other meta-values assigned by the filter.

- **Enabling internal variables:** Values for internal variables are assigned by NiceLabel Automation and are available as read-only values. For more information, see section Internal Variables.

> **TIP**
> If you enable the **assignment area** (in Unstructured Text and XML filters) and **dynamic structure** (in Structured Text filter), NiceLabel Automation extracts **name:value** pairs from the trigger data and automatically sends values to the variables of the same name that are defined in the label. No manual variable mapping is necessary.

Properties

- **Name:** Specifies the unique variable name. Names are not case sensitive. Although you can use spaces in variable names, it's a better practice not to. Even more so if you use variables in scripts or in conditions on actions, because you will have to enclose them in square brackets.
- **Allowed characters:** Specifies the list of characters a value can occupy. You can select between **All** (all characters are accepted), Numeric (only digits are accepted), and Binary (all characters and control codes are accepted).
- **Limit variable length:** Specifies the maximum number of characters a variable can occupy.
- **Fixed length:** Specifies that a value must occupy exactly as many characters as defined by its length.

  NOTE

  It is necessary to limit variable length for certain label objects. An example is EAN-13 barcode, which accepts 13 digits.

- **Value required:** Specifies that the variable must contain a value.
- **Default value:** Specifies a default value. If the variable does not have any value assigned to it, the default value is always used.

4.1. Using Compound Values

Some objects in trigger configuration accept compound values. Such content can be a mixture of fixed values, variables and special characters (control codes). The objects accepting compound values are identified by a small arrow button to the right side of the object. Click the arrow button to insert either a variable or a special character.

- **Using fixed values:** Manually enter a fixed value for the variable.

  ```
  This is fixed value.
  ```

- **Using fixed values and data from variables:** You can define a compound value, combined of variable and fixed values. The variable names must be enclosed in square brackets **`[]`**. You can enter variables manually, or insert them by clicking the arrow button to the right. At processing time, the values of variables are merged together with fixed data and used as the content. For more information, see section Tips and Tricks for Using Variables in Actions.
  In this case, the content merges three variables and manually entered fixed data.

  ```
  [variable1] // This is fixed value [variable2][variable3]
  ```

- **Using special characters:** You can add special characters to the mix. Enter the special characters manually or insert them. For more information, see section Entering Special Characters.
  In this case, the value of **`variable1`** is merged with fixed data and form-feed binary character.

```
[variable1] Form feed will follow this fixed text <FF>
```

4.2. Internal Variables

Internal variables are predefined by NiceLabel Automation. Their values are assigned automatically and are available in read-only mode. The icon with lock symbol in front of the variable name distinguishes internal variables from user-defined variables. You can use internal variables in your actions in the same way as you would use the user-defined variables. The trigger related internal variables work as internal variables for each trigger.

Internal variable	Available in trigger	Description
ActionLastErrorDesc	All	Provides the description of the last occurred error. Use this value as feedback to host system to identify the fault cause.
ActionLastErrorID	All	Provides the last error ID. This is integer value. When value is 0, there was no error. You can use this value in conditions, evaluating if there was some error or not.
BytesOfReceivedData	TCP/IP	Provides the number of bytes received by the trigger.
ComputerName	All	Provides the name of the computer where the configuration runs.
ConfigurationFileName	All	Provides the path and file name of the current configuration (.MISX file).
ConfigurationFilePath	All	Provides the path of the current configuration file. Also see description for ConfigurationFileName.
DataFileName	All	Provides the path and file name of the working copy of received data. Each time the trigger accepts the data, it makes a backup copy of it to the unique file name identified by this variable.
Database	Database	Provides the database type as configured in the trigger.
Date	All	Provides the current date in the format as specified by system locale, such as 26.2.2018.
DateDay	All	Provides the current number of the day in a month, such as 26.
DateMonth	All	Provides the current number of the month in the year, such as 2.

DateYear	All	Provides the current number of the year, such as 2018.
DefaultPrinterName	All	Provides the name of printer driver, which is defined as default.
DriverType	Database	Provides the name of the driver used to connect to the selected database.
Hostname	TCP/IP	Provides the host name of device/computer connecting to the trigger.
HttpMethodName	HTTP	Provides the method name the user has provided in the HTTP request, such as GET or POST.
HttpPath	HTTP	Provides the path defined in the HTTP trigger.
HttpQuery	HTTP	Provides the contents of the query string as received by the HTTP trigger.
NumberOfRowsReturned	Database	Provides the number of rows that the trigger gets from a database.
LocalIP	TCP/IP	Provides the local IP address at which the trigger responds to. This is useful if you have a multi-homing machine with several network interface cards (NIC), and want to determine to which IP address the client connected to. This is useful for printer replacement scenarios.
PathDataFileName	All	Provides the path in the DataFileName variable, without the file name. Also see description for DataFileName.
PathTriggerFileName	File	Provides the path in the TriggerFileName variable, without the file name. Also see description for TriggerFileName.
Port	TCP/IP, HTTP, Web Service	Provides the port number as defined in the trigger.
RemoteHttpIp	HTTP	Provides the host name of device/computer connecting to the trigger.
RemoteIp	Web Service	Provides the host name of device/computer connecting to the trigger.
ShortConfigurationFileName	All	Provides the file name of the configuration file, without a path, Also see description for ConfigurationFileName.
ShortDataFileName	All	Provides the file name to the DataFileName variable, without the path. Also see description for DataFileName.

ShortTriggerFileName	File	Provides the file name to the TriggerFileName variable, without the path. Also see description for TriggerFileName.
SystemUserName	All	Provides the Windows name of the logged-in user.
TableName	Database	Provides the name of the table as used in the trigger.
Time	All	Provides the current time in the format as specified by system locale, such as 15:18.
TimeHour	All	Provides the current hour value, such as 15.
TimeMinute	All	Provides the current minute value, such as 18.
TimeSecond	All	Provides the current second value, such as 25.
TriggerFileName	File	Provides the file name that triggered actions. This is useful when you monitor a set of files in the folder, so you can identify which file exactly triggered actions.
TriggerName	All	Provides the name of the trigger as defined by the user.
Username	All	Provides the NiceLabel Automation user name of the currently logged in user. The variable has contents only if the user login is enabled.

4.3. Global Variables

Global variables are a type of variables that can be used on different labels. Global variables are defined outside the label file and remember the last-used values.

Global variables are typically defined as global counters. Such global variables provide a unique value for every label that requests a new value. File locking takes place ensuring uniqueness of each value.

Global variables are defined in label designer, the NiceLabel Automation only uses it. The source for global variables is configurable in the **Options** dialog (**File** > **Options** > **Global Variables**).

By default, NiceLabel Automation is configured to use global variables from local computer. Their default location is:

```
%PROGRAMDATA%\NiceLabel\Global Variables
```

Global variables are defined in files **GLOBAL.TDB** and **GLOBALS.TDB.SCH**.

In multi-user environments, make sure you configure all clients to use the same network-shared source for global variables, or Control Center-based global variables.

NOTE

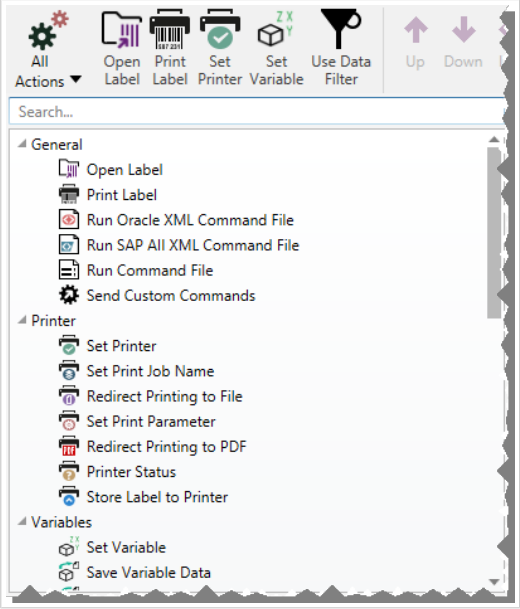
Definition and current value for global variables can be stored in a file or in Control Center (for **LMS Enterprise** and **LMS Pro** products).

5. Actions

The Actions section specifies the list of actions that execute every time a trigger fires.

5.1. Defining actions

To define an action, click the action icon in the Insert Action ribbon group. The main ribbon contains commonly used actions. To see all available actions, click **All Actions** button. To see available commands over the selected action, right-click it and select command from the list.



5.2. Nested actions

Some actions cannot be used on their own. Their specific functionality requires them to be nested below some other action. Use buttons in **Action Order** ribbon group to change action placement. Each action is identified with the ID number that shows its position in the list, including nesting. This ID number will be displayed in the error message so you can find the problematic action easier.

NOTE

The **Print Label** action is a good example of such action. You have to position it under the **Open Label** action, so it references the exact label to print.

5.3. Action execution

The actions in the list execute once per trigger. Listed actions are executed from top to bottom, so the listing order of actions is important.

There are two exceptions. The actions **For Loop** and **Use Data Filter** execute nested actions for multiple times. **For loop** action executes for as many times as defined in its properties, and the **Use Data Filter** for as many times as there are records in a data set returned from the associated filter.

NiceLabel 10 runs as service under a specified Windows user account and inherits security permissions from the account. For more details, see the section Running in Service Mode in NiceLabel Automation User Guide.

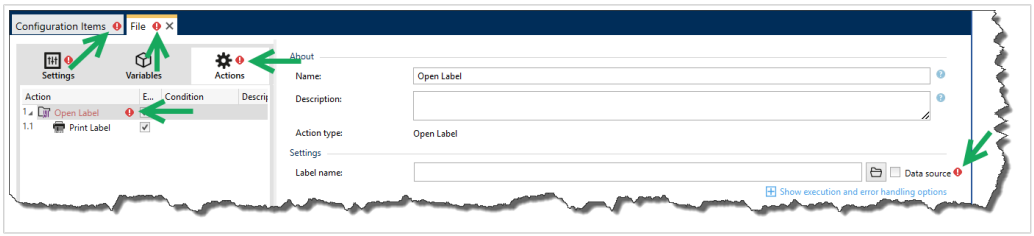
5.4. Conditional actions

Each action can be set as a conditional action. Conditional action only runs if the provided condition allows it to be run. Condition is a single line script (VBScript or Python). To define the condition, click the **Show execution and error handling options** in action properties to expand the possibilities.

In this case, the action executes only if the previous action has completed successfully, so the internal variable **`ActionLastErrorID`** has value 0. To use such condition with internal variables, you must first enable the internal variable.

5.5. Identifying actions in configuration error state

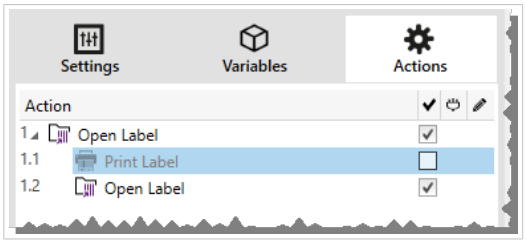
If an action is not completely configured, it is marked with a red exclamation icon. Such actions cannot execute. You can include such action in the **Action** list, but you will have to complete the configuration, before you can start the trigger. If one of the nested actions is in error state, all parent expansion arrows (to the left of the action name) are also colored red as an indicator of sub-action error.



In this case, the **Open Label** action reports configuration error. There is no parameter specified for the label name. The red exclamation icon pops up next to the erroneous parameter in the action itself, in the action list, in the Actions tab, in the trigger tab, and in the Configuration Items tab. This makes the issue easy to identify.

5.6. Disabling actions

By default, every newly created action is enabled and executes if a trigger fires. You can disable the actions that you don't need, but still want to keep the configuration. A shortcut to action enabling & disabling is a check box on the right hand side of the action name in the list of defined actions.



In this case, the **Print Label** action is still defined in the actions list, but has been disabled. Currently, it is not needed and will be ignored during the processing, but you can easily re-enable it at any time.

5.7. Copying actions

You can copy an action and paste it back into the same or any other trigger. You can use standard Windows keyboard shortcuts, or right-click the action.

Right-clicking the action displays the available contextual commands available for the currently selected object.

Automation Builder also enables you to make a selection of multiple actions, and to perform copy, paste and delete operations with them. To make a selection, Use Ctrl/Shift + Click on the required actions.

NOTE

Multiple actions can only be selected under the same parent action, i.e. all selected actions must be on the same level.

5.8. Navigating the action list

Use your mouse to select the defined action and click the respective arrow button in **Action Order** group in the ribbon. You can also use keyboard. Cursor keys move the selection in the action list, Ctrl + cursors keys move the action position up and down, and also left and right for nesting.

5.9. Describing the actions

About group allows you to describe all NiceLabel 10 actions.

- **Name**: by default, action name is defined by its type and is therefore not unique. Define a custom name to make it instantly recognizable among other actions, in logs and in potential error messages.
- **Description**: user notes for the selected action. Description is displayed in actions explorer.
- **Action Type**: read-only field which displays the type of action.

5.10. General

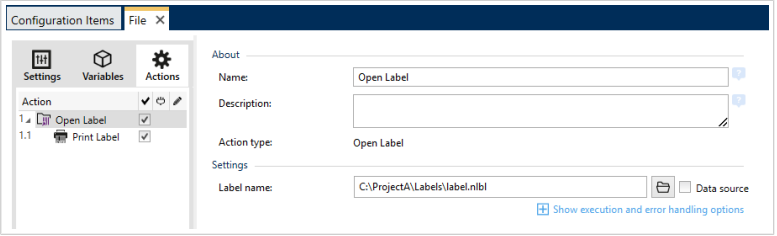
5.10.1. Open Label

Open Label action specifies the label file that is going to be printed. When the action is executed, the label template opens in memory cache. The label remains in the cache for as long as the triggers or events use it.

There is no limit on the number of labels that can be opened concurrently. If the label is already loaded and is requested again, NiceLabel Automation will first determine if a newer version is available and approved for printing, then open it.

In this example, NiceLabel 10 loads the label **`label.nlbl`** from folder **`C:\ProjectA\Labels`**.

```
C:\ProjectA\Labels\label.nlbl
```



If the specified label cannot be found, NiceLabel 10 tries to find it in alternative locations. For more information, see section Search Order for Requested Files in NiceLabelDesigner user guide.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Using Relative Paths

NiceLabel 10 supports the use of relative paths for referencing your label file. Root folder is always the folder where the solution (or configuration in case the action is used in a NiceLabel Automation configuration) is stored.

With the following syntax, the label loads relatively from the location of the configuration file. Automation Builder searches for the label in folder **`ProjectA`**, which is two levels above the current folder, and then in folder **`Labels`**.

```
..\..\ProjectA\Labels\label.nlbl
```

Settings group selects the label file.

- **Label name:** specifies the label name. It can be hard-coded, and the same label will print every time. The option **Data source** enables the file name to be dynamically defined. Select or add a variable that contains the path and/or file name if a trigger is executed or an event takes place.

  TIP
  Usually, the value to the variable is assigned by a filter.

  NOTE
  Use UNC syntax for network resources.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.

- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.10.2. Print Label

This action executes label printing. It must always be nested within the Open Label action. Nesting allows it to obtain the reference to the label that is going to be printed. This further allows you to keep multiple labels open at the same time, and enables you to specify which label should be printed.

After issuing this action, the label gets printed using the printer driver that is defined in the label template. If the defined printer driver is not found on the system, the label is printed using the system default printer driver. You can override the printer driver selection via Set Printer action.

To achieve high performance label printing, NiceLabel 10 activates two settings by default:

- **Parallel processing.** Multiple print processes are all carried out simultaneously. The number of background printing threads depends on the hardware; specifically on the processor type. Each processor core can accommodate a single printing thread. This default can be changed. For more information, see section Parallel Processing in NiceLabel Automation user guide.
- **Asynchronous mode.** As soon as the trigger pre-processing completes and the instructions for the print engine are available, the printing thread takes it over in the background. The control is returned to the trigger so it can accept the next incoming data stream as soon as possible. If synchronous mode is enabled, the control is not returned to the trigger until the print process is finished. This can take a while, but the trigger benefits from providing feedback back to data-providing application. For more information, see the section Synchronous Mode in NiceLabel Automation user guide.

> **NOTE**
> Using **Save error to variable** option in **Action Execution and Error Handling** does not yield any result in asynchronous mode, as the trigger does not receive feedback from the print process. To capture the feedback from the print process, enable synchronous mode first.

> **NOTE**
> If the Print Label action is nested under a For Loop action, Automation executes it in session printing mode. This mode acts as a printing optimization mode that prints all labels in a loop using a single print job file. For details, see Session Printing section in NiceLabel Automation user guide.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Quantity group defines the number of labels to be printed using the active form.

- **Labels:** sets the number of printed labels. **Data source** specifies or adds a variable that defines the label print quantity dynamically.

  > **NOTE**
  > Variable value is usually assigned by the **Use Data Filter**action and must be integer.

All (unlimited quantity): depending on the label template design, the labels are printed in various quantities.

Unlimited Quantity Printing Details

Typically, this option is used in two scenarios.

1. Command the printer to continuously print the same label until it is switched off, or after it receives a command to clear its memory buffer.

   > **WARNING**
   > This scenario requires NiceLabel printer driver to be installed and used for label printing.

If printing a fixed label, just a single print job is sent to the printer, with the quantity set to "unlimited". Label printers have a print command parameter which indicates "unlimited" printing. If the label is not fixed, but includes objects that change during the printing, such as counters, the printed quantity is set to maximum quantity supported by the printer. NiceLabel printer driver is aware of the printer quantity limit and print as many labels as possible.

Example

Maximum supported print quantity is 32,000. This is the amount of labels that are print after selecting the All (unlimited quantity) option.

2. The trigger doesn't provide any data, but only acts as a signal for "event has happened". The logic to acquire necessary data is included in the label. Usually, a connection to a database is configured on the label, and at every trigger the label must connect to the database, and acquire all records from the database. In this case, the **All (unlimited quantity)** option is understood as "print all records from the database".

- **Variable quantity (defined from label variable):** specifies a label variable that defines the label quantity to be printed.
  The trigger doesn't receive the number of labels to be print, so it passes the decision to the label template. The label might contain a connection to a database, which provide the label quantity, or there is another source of quantity information. A single label variable must be defined as "variable quantity".

Advanced group defines label printing details. Click **Show advanced print options** to define the **Advanced** print options:

This section specifies non-frequently used label quantity related settings.

- **Number of skipped labels:** specifies the number of labels that are skipped on the first page of labels. The sheet of labels might have been printed once already, but not entirely. You can re-use the same sheet by offsetting the starting position. This option is applicable, if you print labels to sheets of labels, not rolls of labels, so it's effective for office printers and not for label printers.
- **Identical label copies:** specifies the number of label copies to be printed for each unique label. For fixed labels, this option produces the same result as the main **Number of Labels** option. For variable labels, such as labels using counters, you can get the real label copies.
- **Label sets:** specifies how many times the entire label printing process should repeat.

Example

Trigger or event receive content with 3 lines of CSV-formatted data, so 3 labels are expected to be printed (1, 2, 3). If you set this option to 3, the printout is done in the following order: 1, 2, 3, 1, 2, 3, 1, 2, 3.

- **Metadata**: with each print job writes your printing comments to Control Center. You can see your Metadata in **History** > **Printing** > **Print Metadata** column. You can use Metadata for sorting, filtering,

and other functions in Control Center. Metadata does not affect your printing or print streams. You use Metadata for logging your additional information about your print jobs in Control Center. Metadata can include LOT number or other label variables, printer names, and user/system generated values.

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

TIP

All Advanced group values can either be hard-coded, or dynamically provided by an existing or a newly added variable.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.10.3. Run Oracle XML Command FileXML Transform

PRODUCT LEVEL INFO

Automation Builder features require **LMS Enterprise**.

This action executes printing with data obtained from an Oracle XML-formatted file.

NiceLabel Automation internally supports XML files with "Oracle XML" structure, which are defined by Oracle Warehouse Management software.

Use this action as a shortcut. It helps you execute Oracle XML files directly and without the need to parse them using XML filter and mapping the values to variables.

To be able to use this action, the XML file must conform to Oracle XML specifications. For more information, see section Oracle XML Specifications in NiceLabel Automation user guide.

Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation user guide.

PRODUCT LEVEL INFO

Requires **Automation Builder**.

How to receive a command file in a trigger and execute it

After a trigger receives the command file and you wish to execute it, complete the following steps:

1. In Automation Builder module, on **Variables** tab, click the **Internal Variable** button in the ribbon.
2. In the drop down list, enable the internal variable named **`DataFileName`**. This internal variable provides path and file name of the file that contains the data received by the trigger. In this case, its contents is command file. For more information, see section Internal Variables in the NiceLabel Automation user guide.
3. In **Actions** tab, add the action to execute the command file, such as Run Command File, Run Oracle XML Command File, or Run SAP AII XML Command File.
   For the action **Run Command File**, select the type of the command file in **File type**.
4. Enable the option **Variable**.
5. Select the variable named **`DataFileName`** from the list of available variables.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.

- **Action type**: read-only information about the selected action type.

File group defines the Oracle XML command file to be used.

- **File name**: selected Oracle XML command file. It can either be hard-coded or dynamically defined using an existing or a newly created variable.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.10.4. Run SAP AII XML Command File

PRODUCT LEVEL INFO

Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action executes printing with data from an SAP AII XML-formatted file.

NiceLabel Automation internally supports XML files with the "SAP AII XML" structure, which are defined by SAP software.

Use this action as a shortcut. it helps you execute SAP AII XML files directly without any need to parse them using XML filter and map values to variables. To be able to use this action, the XML file must conform to SAP AII XML specifications. For more information, see section SAP AII XML Specifications in NiceLabel Automation user guide.

Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation user guide.

PRODUCT LEVEL INFO

Requires **Automation Builder**.

How to receive a command file in a trigger and execute it

After a trigger receives the command file and you wish to execute it, complete the following steps:

1. In Automation Builder module, on **Variables** tab, click the **Internal Variable** button in the ribbon.
2. In the drop down list, enable the internal variable named **`DataFileName`**. This internal variable provides path and file name of the file that contains the data received by the trigger. In this case, its contents is command file. For more information, see section Internal Variables in the NiceLabel Automation user guide.
3. In **Actions** tab, add the action to execute the command file, such as Run Command File, Run Oracle XML Command File, or Run SAP AII XML Command File.
   For the action **Run Command File**, select the type of the command file in **File type**.
4. Enable the option **Variable**.
5. Select the variable named **`DataFileName`** from the list of available variables.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

File group defines the SAP AII XML command file to be used.

- **File name**: selected SAP AII XML command file. It can either be hard-coded or dynamically defined using an existing or a newly created variable.

Optional Parameters group allows defining the label name in case it is not included in the XML file.

- **Label name**: the selected label file that should be used in the command file. It can either be hard-coded or dynamically defined using an existing or a newly created variable.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.10.5. Run Command File

This action executes commands that are included in a selected command file. **All File type** options provide commands that NiceLabel 10 executes in top-to-bottom order.

Command files usually provide data for a single label, but you can define files of any level of complexity. For more information, see section Command File Types.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

File group defines the type and name of the command file that is going to be executed (JOB, XML or CSV).

- **File type.** Specifies the type of the command file to be executed.
- **File name.** Specifies the command file name.
  File name can be hard-coded, and the same command file will execute every time. The option **Variable** enables a variable file name. Select or create a variable that contains the path and/or file name if a trigger is executed or an event takes place. Usually, the value to the variable is assigned by a filter.
  Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation user guide.

PRODUCT LEVEL INFO

Requires **Automation Builder**.

How to receive a command file in a trigger and execute it

After a trigger receives the command file and you wish to execute it, complete the following steps:

1. In Automation Builder module, on **Variables** tab, click the **Internal Variable** button in the ribbon.
2. In the drop down list, enable the internal variable named **`DataFileName`**. This internal variable provides path and file name of the file that contains the data received by the trigger. In this case, its contents is command file. For more information, see section Internal Variables in the NiceLabel Automation user guide.
3. In **Actions** tab, add the action to execute the command file, such as Run Command File, Run Oracle XML Command File, or Run SAP AII XML Command File.
   For the action **Run Command File**, select the type of the command file in **File type**.
4. Enable the option **Variable**.
5. Select the variable named **`DataFileName`** from the list of available variables.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.10.6. Send Custom Commands

This action executes the entered custom NiceLabel commands.

Always nest this action under the Open Label action. This enables referencing the label to which the commands apply. For more information, see section Using Custom Commands in NiceLabel Automation user guide.

NOTE

Majority of custom commands is available with individual actions, so in most cases, custom commands are not required.

NOTE

Send Custom Commands action can be used to end the Session printing mode. This mode acts as a printing optimization mode that prints all labels in a loop using a single print job file. To end session printing, nest the Send Custom Commands action under the For Loop action and use the SESSIONEND command. For details, see sections Session Printing and Using Custom Commands in NiceLabel Automation user guide.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Script editor offers the following features:

- **Insert data source**: inserts an existing or newly created variable into the script.
- **Script editor**: opens the editor which makes scripting easier and more efficient.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.11. Printer

5.11.1. Set Printer

This action specifies the name of the printer to be used for printing the active label.

NOTE

This action overrides the printer selected in the label properties.

This action is useful when printing an identical label on multiple printers. Always nest this action under the Open Label action to provide the label with the reference for the preferred printer.

This action reads the default settings (such as speed and darkness) from the selected printer driver and applies them to the label. If you don't use the **Set Printer** action, the label gets printed using the printer defined in the label template.

WARNING

Pay attention when switching the printers, e.g. from Zebra to SATO, or even from one printer model to another model of the same brand. Printer settings might not be compatible and the label printouts might not appear identical. Also, label design optimizations for original printer, such as internal counters, and internal fonts, might not be available on the newly selected printer.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Printer group specifies the printer name to be used for the current print job.

- **Printer name**: select it from the list of locally installed printer drivers, or manually enter a printer name. Select **Data source** to dynamically select the printer using a variable. If enabled, select or create a variable that contains the printer name which is used if the action is run.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.11.2. Set Print Job Name

This action specifies the name of the print job file as it appears in the Windows Spooler. A default print job name is the name of the used label file. This action overrides it.

NOTE

Always nest the action under the Open Label action, so it applies to the adequate label file.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Print Job group defines the print job name.

- **Name**: sets the print job name. It can be hard-coded, and the same name is used for each print action. Variable enables a variable file name. Select or create a variable that contains the path and/or file name if the event happens or a trigger fires.

  NOTE

  In Automation Builder module, the variable value is usually assigned by a filter.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.11.3. Redirect Printing to File

This action diverts the print job to a file. Instead of sending the created print file to a printer port as defined in the printer driver, the printout is redirected to a file. You can append data to an existing file, or overwrite it.

This action enables you to capture printer commands in a separate file.

The action instructs Automation Builder module to redirect printing – as a result, the labels are not going to be printed. Make sure the action is followed by the Print Label action.

NOTE

NiceLabel Automation runs as service under defined Windows user account. Make sure this user account has privileges accessing the specified folder with read/write permissions. For more information, see section Access to Network Shared Resources in the NiceLabel Automation user guide.

NOTE

Redirect Printing to File action is useful for printing several different labels (.NLBL files) to a network printer while retaining the correct order of labels. If multiple .NLBL files are printed from the same trigger, Automation Builder sends each label to the printer in a separate print job, even if the target printer is the same for both labels. If a network printer is used, job of another user can be inserted between two jobs the trigger must send together. Using this action, you can append print data into the same file and send its contents to the printer using the Send Data to Printer action.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.

- **Action type**: read-only information about the selected action type.

File group of settings defines how the file selection for redirecting is done.

- **File name**:specifies the file name. It can either be hard-coded or dynamically defined using an existing or a newly created variable.
  Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation user guide.

NOTE
When using this action, make sure your user account has sufficient privileges for accessing the specified folder with read/write permissions.

File write mode group of settings selects how the file is treated in case of repeated redirects.

- **Overwrite the file**: if the specified file already exists on the disk, it is going to be overwritten.
- **Append data to the file**: the job file is added to the existing data in the provided file.

Persistence group controls the continuity of the redirect action. It defines the number of Print Label actions that are affected by the **Redirect Printing to File** action.

- **Apply to next print action**: specifies for the print redirect to be applicable to the next Print Label action only (single event).
- **Apply to all subsequent print actions**: specifies for the print redirect to be applicable to all **Print Label** action defined after the current **Redirect Printing to File** action.

NOTE
The action only redirects printing. Make sure it is followed by the Print Label action.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  > **NOTE**
  > Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.11.4. Set Print Parameter

> **PRODUCT LEVEL INFO**
> Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action allows you to fine tune the parameters that relate to the printer driver. These include parameters such as speed and darkness for label printers, or paper tray for laser printers.

Printer settings are applied to the current printout only and are not remembered during the upcoming event.

> **WARNING**
> Your parameters from **Set Print Parameter** action don't apply when you preview or reprint labels from your Control Center.
>
> You can avoid this by saving your print parameters (printer properties) in your label template or your printer driver. Printer properties may differ from driver to driver and also from printer to printer.

NOTE

If you use Set Printer action to change the printer name, make sure the **Set Print Parameter** action is used right after. Before you can apply the DEVMODE structure to the printer driver, first load the default driver settings. This is done by the Set Printer action. The DEVMODE is only compatible with the DEVMODE of the same printer driver.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Print Parameters group allows action fine tuning before printing.

- **Paper bin**: name of the paper bin that contains the label media. This option is usually used with laser and ink jet printers with multiple paper bins. The provided name of the paper bin must match the name of the bin in the printer driver. Check the printer driver properties for more details.
- **Print speed**: defines printing speed. This setting overrides the setting defined with label. The provided value must be in the range of accepted values.

Example

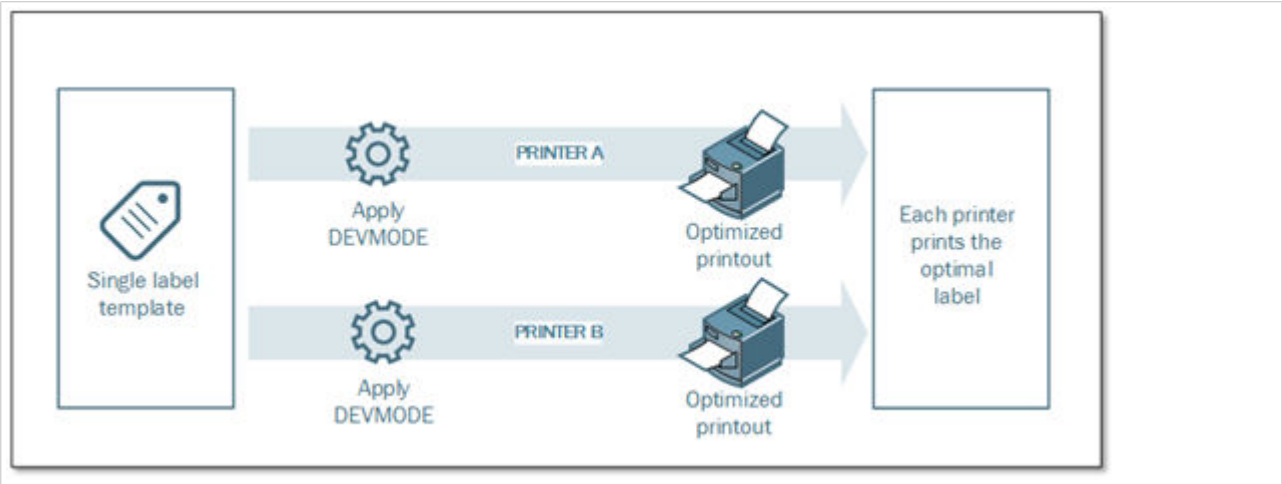
The first printer model accepts a range of values from 0 to 30, while the second printer model accepts values from -15 to 15. For more information, see printer driver properties

- **Darkness**: defines the darkness of the printed objects on the paper and overrides setting from the label. The provided value must be in range of accepted values.
- **Print offset X**: applies horizontal offset. The label printout will be repositioned by the specified number of dots in the horizontal direction. Negative offset can be defined.
- **Print offset Y**: applies vertical offset. The label printout will be repositioned by the specified number of dots in the vertical direction. Negative offset can be defined.

TIP

All print parameters can either be hard-coded or dynamically defined using an existing or a newly created variable.

Advanced group customizes the printer settings that are sent along with the print job.



Printer settings, such as printing speed, darkness, media type, offsets and similar, can be defined as follows:

- Defined in a label
- Recalled from a printer driver
- Recalled from a printer at print time

The supported methods depend on the printer driver and its capabilities. Printing mode (recall settings from label or driver or printer) is configurable in the label design. You might need to apply these printer settings at print time – they can vary with each printout.

Example

A single label should be printed using a variety of printers, but each printer requires slightly different parameters. The printers from various manufacturers don't use the same values to set the printing speed or temperature. Additionally, some printers require vertical or horizontal offset to print the label to the correct position. During the testing phase, you can determine the optimal settings for every printer you intend to use and apply them to a single label template just before printing. This action will apply the corresponding settings to each defined printer.

This action expects to receive the printer settings in a DEVMODE structure. This is a Windows standard data structure with information about initialization and environment of a printer.

Printer settings option applies custom printer settings. The following inputs are available:

- **Fixed-data Base64-encoded DEVMODE**. In this case, provide the printer's DEVMODE encoded in Base64-encoded string directly into the edit field. If executed, the action converts the Base64-encoded data back into binary form.
- **Variable-data Base64-encoded DEVMODE**. In this case, the selected data source must contain the Base64-encoded DEVMODE. Enable Data source and select the appropriate variable from the list. If executed, the action converts the Base64-encoded data back into the binary form.

- **Variable-data binary DEVMODE (available in Automation Builder)**. In this case, the selected variable must contain the DEVMODE in its native binary form. Enable **Data source** and select the appropriate variable from the list. If executed, the action uses the DEVMODE as-is, without any conversion.

  > **NOTE**
  > If the variable does not provide a binary DEVMODE, make sure that the selected variable is defined as a binary variable in the configuration.

> **NOTE**
> Make sure the Set Printer action is defined in front of this action.

Label settings overrides label properties defined in **Label Properties** in Designer. Use this option when you print your labels to a printer or media with different properties as defined in **Label Properties** in Designer. With this option you can:

- Change your label dimensions (width and height).
- Add or change label margins.
- Disable cutter.
- Disable batch printing.
- Apply different stocks by changing **Labels Across** parameter (horizontal and vertical count, gaps, processing order).
- Redefine Portrait or Landscape orientation.
- Rotate labels by 180° degrees.

Automation applies **Label settings** at print time. Label settings parameters are not saved in your label templates. You can provide Label settings as an XML payload.

Sample Label settings XML

The sample below presents a structural view of the label settings and their attributes.

> **NOTE**
> Dimension units in XML correspond to your label design units (cm, in, mm, dot). You can change units in Designer if you go to **Label Properties** > **Label Dimensions** > **Unit of measure**.

```
<LabelSettings>
    <Width>100</Width>
    <Height>30</Height>
    <Margin>
```

```
        <Left>2</Left>
        <Right>3</Right>
        <Top>4</Top>
        <Bottom>5</Bottom>
    </Margin>
    <LabelsAcross>
        <Horizontal>
            <Count>2</Count>
            <Gap>4</Gap>
        </Horizontal>
        <Vertical>
            <Count>3</Count>
            <Gap>5</Gap>
        </Vertical>
        <ProcessingOrder>HorizontalTopRight</ProcessingOrder>
    </LabelsAcross>
    <Orientation>Landscape</Orientation>
    <Rotated>true</Rotated>
    <DisableCutter/>
    <DisableBatchPrinting/>
</LabelSettings>
```

Label settings XML specification

This section contains a description of the XML file structure to define **Label settings** parameters and values.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="nonNegativeFloat">
    <xs:restriction base="xs:float">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="LabelSettings">
    <xs:complexType>
      <xs:all>
        <xs:element name="DisableCutter" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence/>
          </xs:complexType>
        </xs:element>
        <xs:element name="DisableBatchPrinting" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence/>
          </xs:complexType>
```

```
        </xs:element>
        <xs:element name="Width" type="nonNegativeFloat" minOccurs="0"
maxOccurs="1"/>
        <xs:element name="Height" type="nonNegativeFloat" minOccurs="0"
maxOccurs="1"/>
        <xs:element name="Margin" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:all>
              <xs:element name="Left" type="nonNegativeFloat" minOccurs="0"
maxOccurs="1"/>
              <xs:element name="Right"  type="nonNegativeFloat" minOccurs="0"
maxOccurs="1"/>
              <xs:element name="Top"  type="nonNegativeFloat" minOccurs="0"
maxOccurs="1"/>
              <xs:element name="Bottom" type="nonNegativeFloat" minOccurs="0"
maxOccurs="1"/>
            </xs:all>
          </xs:complexType>
        </xs:element>
        <xs:element name="LabelsAcross" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:all>
              <xs:element name="Horizontal" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:all>
                    <xs:element name="Count" type="xs:nonNegativeInteger"
minOccurs="0" maxOccurs="1" />
                    <xs:element name="Gap" type="nonNegativeFloat"
minOccurs="0" maxOccurs="1"/>
                  </xs:all>
                </xs:complexType>
              </xs:element>
              <xs:element name="Vertical" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:all>
                    <xs:element name="Count" type="xs:nonNegativeInteger"
minOccurs="0" maxOccurs="1" />
                    <xs:element name="Gap" type="nonNegativeFloat"
minOccurs="0" maxOccurs="1"/>
                  </xs:all>
                </xs:complexType>
              </xs:element>
              <xs:element name="ProcessingOrder" minOccurs="0" maxOccurs="1">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="HorizontalTopLeft"/>
```

```
                    <xs:enumeration value="HorizontalTopRight"/>
                    <xs:enumeration value="HorizontalBottomLeft"/>
                    <xs:enumeration value="HorizontalBottomRight"/>
                    <xs:enumeration value="VerticalTopLeft"/>
                    <xs:enumeration value="VerticalTopRight"/>
                    <xs:enumeration value="VerticalBottomLeft"/>
                    <xs:enumeration value="VerticalBottomRight"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:all>
          </xs:complexType>
        </xs:element>
        <xs:element name="Orientation" minOccurs="0" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Portrait"/>
              <xs:enumeration value="Landscape"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Rotated" type="xs:boolean" minOccurs="0"
maxOccurs="1"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (**`true`** or **`false`**). When the result of the expression is **`true`**, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

> **NOTE**
> Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.11.5. Redirect Printing to PDF

> **PRODUCT LEVEL INFO**
> This action is available in NiceLabelLMS Enterprise.

This action diverts the print job to a PDF document. The created PDF document retains the exact label dimensions as defined during the label design process. The rendering quality of graphics in the PDF matches the resolution of the target printer and desired printout size.

Print stream data can be appended to an existing file, or it may overwrite it.

The action instructs NiceLabel 10 to redirect printing – as a result, the labels are not printed. Make sure the action is followed by the **Print Label** action.

> **NOTE**
> NiceLabel Automation module runs as service under defined Windows user account. Make sure this user account has privileges accessing the specified folder with read/write permissions. For more information, see section Access to Network Shared Resources in NiceLabel Automation user guide.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.

- **Action type**: read-only information about the selected action type.

File group defines the redirect file.

- **File name**: specifies the file name for diverting the print job to. If hard-coded, the printing is redirected to the specified file every time. To define it dynamically, use an existing or create a new variable.
- **Overwrite the file**: if the specified file already exists on the disk, it is going to be overwritten (selected by default).
- **Append data to the file**: the job file is appended to the existing data in the provided file (deselected by default).
- **Embed fonts in PDF**: If you use some non-standard font, your solution might produce different PDF output on computers without this font. With **Embed fonts in PDF** option enabled, you embed your non-standard font into your solution and your solution creates the same PDF on all computers. **Embed fonts in PDF** option creates PDF for Archiving (PDF/A) compliant document. The PDF/A standard ensures your documents are reproduced exactly the same way, regardless of what software you use. The information required for displaying the contents like images, fonts, and color information is embedded in your PDF file.

  NOTE

  The output PDF file consumes more space when saved with this option enabled.

Persistence group allows controlling the persistence of the redirect action. Define the number of Print Label actions that are affected by the **Redirect Printing to File** action.

- **Apply to next print action**: specifies for the print redirect to be applicable to the next Print Label action only (single event).
- **Apply to all subsequent print actions**: specifies for the print redirect to be applicable to all Print Label action defined after the current Redirect Printing to File action.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  > **NOTE**
  > Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.11.6. Printer Status

> **PRODUCT LEVEL INFO**
> Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action communicates with the printer to acquire its real-time state, and contacts the Windows Spooler for additional information about the printer and its jobs.

As a result, the information about errors, spooler status, number of jobs in the spooler is collected. This uncovers potential errors and makes them easy to identify.

> **NOTE**
> Possible use case scenarios. (1) Verifying the printer status before printing. If the printer is in error state, you print the label to a backup printer. (2) Counting the number of jobs waiting in a spooler of main printer. If there are too many, you will print label to alternative printer. (3) You will verify the printer status before printing. If the printer is in error state, you will not print labels, but report the error back to the main system using any of the outbound actions, such as Send Data to TCP/IP Port, Send Data to HTTP, Execute SQL Statement, Web Service, or as the trigger response.

Live Printer Status Prerequisites

To make live printer status monitoring possible, follow these instructions:

- Use the NiceLabel Printer Driver to receive detailed status information. If using any other printer driver, you can only monitor the parameters retrieved from the Windows Spooler.
- The printer must be capable of reporting its live status. For the printer models supporting bidirectional communication see NiceLabel Download web page.
- Printer must be connected to an interface with support for bidirectional communication.
- Bidirectional support must be enabled in **Control Panel > Hardware and Sound > Devices and Printers > driver > Printer Properties > Ports tab > Enable bidirectional support**.
- If using a network-connected label printer, make sure you are using **Advanced TCP/IP Port**, not **Standard TCP/IP Port**. For more information, see the Knowledge Base article.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Printer group selects the printer.

- **Printer name:** specifies the printer name to be used for the current print job.
  You can select a printer from the list of locally installed printer drivers, or you can enter any printer name. Data source enables variable printer name. When enabled, select or create a variable that contains the printer name when a trigger is executed or an event takes place. Usually, the variable value is assigned by a filter.

Data Mapping group sets the parameters that are returned as a result of the **Printer Status** action.

> **WARNING**
> Most of the following parameters are only supported with NiceLabelprinter driver. If you are using any other printer driver, you can use only the spooler-related parameters.

- **Printer status:** specifies the printer live status formatted as a string.
  If the printer reports multiple states, all states are merged into a single string, delimited by comma ",". If there are no reported printer issues, this field is empty. Printer status might be set to **Offline**, **Out of labels** or **Ribbon near end**. Since there is no standardized reporting protocol, each printer vendor uses proprietary status messages.
- **Printer error:** boolean (true/false) value of the printer error status.
- **Printer offline:** boolean (true/false) value of the printer offline status.
- **Driver paused:** boolean (true/false) value of the driver pause status.

- **NiceLabel driver:** specifies boolean (true/false) value of the printer driver status. Provides information if the selected driver is a NiceLabel driver.
- **Spooler status:** specifies the spooler status in a string form – as reported by the Windows system. The spooler can simultaneously report several statuses. In this case, the statuses are merged using comma ",".
- **Spooler status ID:** specifies spooler status formatted as a number – as reported by the Windows system. The spooler can simultaneously report several statuses. In this case, the returned status IDs contains all IDs as flags. For example, value 5 represents status IDs 4 and 1, which translates to "Printer is in error, Printer is paused". Refer to the table below.

TIP

The action returns a decimal value, the values in the table below are in hex format, so you will have to do the conversion before parsing the response.

- **Table of spooler status IDs and matching descriptions**

Spooler status ID (in hex)	Spooler status description
0	No status.
1	Printer is paused.
2	Printer is printing.
4	Printer is in error.
8	Printer is not available.
10	Printer is out of paper.
20	Manual feed required.
40	Printer has a problem with paper.
80	Printer is offline.
100	Active Input/Output state.
200	Printer is busy.
400	Paper jam.
800	Output bin is full.
2000	Printer is waiting.
4000	Printer is processing.
10000	Printer is warming up.
20000	Toner/Ink level is low.
40000	No toner left in the printer.
80000	Current page can not be printed.
100000	User intervention is required.
200000	Printer is out of memory.
400000	Door is open.

800000	Unknown error.
1000000	Printer is in power save mode.

- **Number of jobs in the spooler:** specifies the number of jobs that are in the spooler for the selected printer.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

> **NOTE**
> Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.11.7. Store Label to Printer

> **PRODUCT LEVEL INFO**
> Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action saves label template in the printer memory. The action is a vital part of Store/Recall printing mode, using which you first store a label template into the printer's memory and later recall it. The non-changeable parts of label design are already stored in the printer, so you only have to provide the data for variable label objects at print time. For more information, see section Using Store/Recall Printing Mode in NiceLabel Automation user guide.

NOTE

The required label data transfer time is greatly minimized as there is less information to be sent. This action is commonly used for stand-alone printing scenarios, where the label is stored to the printer or applicator in the production line and later recalled by some software or hardware trigger, such as barcode scanner or photocell.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Advanced options for storing label to printer group allows you select a label and the preferred storing variant.

- **Label name to be used on the printer**: specifies the name to be used for storing the label template in printer memory. Enter the name manually or enable **Data source** to define the name dynamically using an existing or newly created variable.

  WARNING

  When storing the label to a printer, it is recommended to leave the label name under the advanced options empty. This prevents label name conflicts during the recall label process.

- **Store variant**: defines printer memory location for stored label templates. Enter the location manually or enable **Data source** to define the name dynamically using an existing or newly created variable.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.

- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  > **NOTE**
  > Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.11.8. Print PDF Document

> **PRODUCT LEVEL INFO**
> Automation Builder features require **LMS Enterprise**.

The Print PDF Document action prints static PDF documents that are not linked to the labels in your PowerForms solutions or NiceLabel Automation configurations. Use this action to print any PDF document directly from your solutions or configurations. The PDF documents can be stored on:

- Your computer
- NiceLabel Control Center
- Web server
- Shared network drives

TIP

The action is useful if you plan to equip your packages with printed PDF reports on included items, or if you want to print the packaging documentation without opening the file browser.

NOTE

When in use, the Print PDF Document action takes one printer seat from your license quota. Read the NiceLabel licensing document for more information on licensing.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Printer group specifies the printer name to be used for the current print job.

- **Printer name**: select it from the list of locally installed printer drivers, or manually enter a printer name. Select **Data source** to dynamically select the printer using a variable. If enabled, select or create a variable that contains the printer name which is used if the action is run.

File group defines the redirect file.

- **File name**: specifies which PDF to print.

NOTE

Use the UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation User Guide.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.

- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.12. Variables

5.12.1. Set Variable

This action assigns a new value to the selected variable.

Variables usually obtain their values using Use Data Filter action (available in Automation Builder) which extracts fields from the received data and maps them to variables. You might also need to set the variable values by yourself, usually for troubleshooting purposes. In Automation Builder, the variable values are not remembered between multiple triggers, but are kept while the same trigger is being processed.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Variable group defines the variable name and its value.

- **Name**: name of variable that should store the value changed.
- **Value:** value to be set to a variable. It can either be manually or dynamically defined using an existing or a newly created variable.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.12.2. Save Variable Data

PRODUCT LEVEL INFO

Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action saves values of a single or multiple variables in an associated data file.

In NiceLabel Automation module, this action allows data exchange between triggers. To read the data back into the trigger, use action Load Variable Data.

TIP

The values are saved in a CSV file with the first line containing variable names. If the variables contain multi-line values, the newline characters (CR/LF) are encoded as `\n\r`.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Settings group defines the file name.

- **File name**: data file to save the variable data to. If the name is hard-coded, values are saved into the same data file each time.
  Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation user guide.

If file exists group offers additional options to save the values.

- **Overwrite the file**: overwrites the existing data with new variable data. The old content is lost.
- **Append data to the file**: appends the variable values to the existing data files.

File Structure group defines the CSV variable data file parameters:

- **Delimiter**: specifies the delimiter type (tab, semicolon, comma or custom character). Delimiter is a character that separates the stored variable values.
- **Text qualifier**: specifies the character that qualifies the stored content as text.
- **File encoding**: specifies character encoding type to be used in the data file. **Auto** defines the encoding automatically. If required, the preferred encoding type can be selected from the drop-down list.

  TIP

  UTF-8 makes a good default selection.

- **Add names of variable in the first row**: places the variable name in the first row of the file.

Variables group defines the variables whose value should be read from the data file. Values of the existing variables are overwritten with values from the file.

- **All variables**: variable data of all variables from the data file is read.
- **Selected variable**s: variable data of listed variables is red from the data file.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.12.3. Load Variable Data

PRODUCT LEVEL INFO

Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action loads values of a single or multiple variables from the associated data file as saved by the action **Save Variable Data**. Use this action to exchange the data between triggers. You can load a particular variable or all variables that are stored in the data file.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Settings group defines the file name.

- **File name**: specifies the file for the variable data to be loaded from. If the name is hard-coded, the values are loaded from the same file each time.
  Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation user guide.

File Structure group settings must reflect the structure of the saved file from the Save Variable Data action.

- **Delimiter**: specifies delimiter type (tab, semicolon, comma or custom character). Delimiter is a character that separates the values.
- **Text qualifier**: specifies the character that qualifies content as text.
- **File encoding**: specifies the character encoding type used in the data file. **Auto** defines the encoding automatically. If needed, select the preferred encoding type from the drop-down list.

TIP
UTF-8 makes a good default selection.

Variables group defines the variables whose values should be loaded from the data file.

- **All variables**: specifies all defined variables in the data file to be read.
- **Selected variables**: specifies selection of individual variables to be read from the data file.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.

- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.12.4. String Manipulation

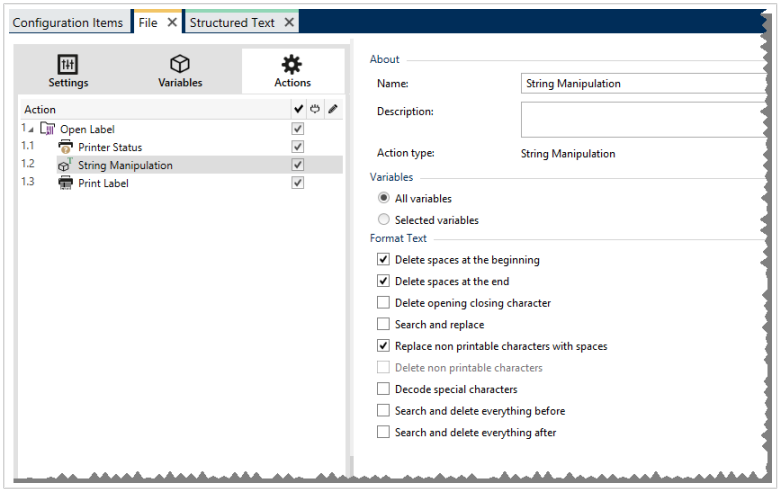
This action defines how the values of selected variables should be formatted.

The most popular string manipulation actions are: delete leading and trailing spaces, search and replace characters, and delete opening and closing quotes.

This feature is often required is a trigger receives an unstructured data file or legacy data. in such cases, the data needs to be parsed using the **Unstructured Data** filter. String Manipulation action allows you to fine-tune the data value.

NOTE

If this action doesn't provide enough string manipulation power for a particular case, use **Execute Script** action instead to manipulate your data using Visual Basic Script or Python scripts.



About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Variables group defines the variables whose values need to be formatted.

- **All variables**: specifies all the defined variables in a data file to be formatted.
- **Selected variables**: specifies a selection of variables to be formatted from the data file.

Format Text group defines string manipulation functions that apply to the selected variables or fields. Multiple functions can be used. The functions apply in the same order as seen in the editor – from top to bottom.

- **Delete spaces at the beginning**: deletes all space characters (decimal ASCII code 32) from the beginning of the string.
- **Delete spaces at the end**: deletes all space characters (decimal ASCII value 32) from the end of a string.
- **Delete opening closing characters**: deletes the first occurrence of the selected opening and closing characters that is found in the string.

Example

If using "{" as opening character and "}" as closing character, the input string `{{selection}}` is converted to `{selection}`.

- **Search and replace**: executes standard search and replace function upon the provided values for find what and replace with. Regular expressions are supported.

NOTE

Several implementations of regular expressions are present. NiceLabel 10 uses .NET Framework syntax for the regular expressions. For more information, see the Knowledge Base article.

- **Replace non printable characters with space**: replaces all control characters in the string with "space" character (decimal ASCII code 32). Non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Delete non printable characters**: deletes all control characters in the string. Non printable characters are characters with decimal ASCII values between 0–31 and 127–159.
- **Decode special characters**: decodes the characters (or control codes) that are not available on the keyboard, such as Carriage Return or Line Feed. NiceLabel 10 uses a notation to encode such characters in human-readable form, such as <CR> for Carriage Return and <LF> for Line Feed. This option converts special characters from NiceLabel syntax into actual binary characters.

Example

When you receive the data "<CR><LF>", Designer uses it as plain string of 8 characters. You will have to enable this option to interpret and use the received data as two binary characters `CR` (Carriage Return – ASCII code 13) and `LF` (Line Feed – ASCII code 10).

- **Search and delete everything before**: finds the provided string and deletes all characters in front of the defined string. The string can also be deleted.
- **Search and delete everything after**: finds the provided string and deletes all characters behind the defined string. The string can also be deleted.
- **Change case**: Changes all characters in your strings to uppercase or lowercase.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

> **NOTE**
> Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.13. Batch Printing

5.13.1. For Loop

> **PRODUCT LEVEL INFO**
> Here described product feature is available in **NiceLabel LMS Enterprise**.

This action executes all of the subordinate (nested) actions multiple times. All nested actions are executed in a loop for as many times as defined by the difference between start value and end value.

NOTE

For Loop action starts session printing mode – a printing optimization mode that prints all labels in a loop using a single print job file. For details, see Session Printing section in NiceLabel Automation user guide.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Loop Settings group includes the following options:

- **Start value**: loop starting point reference. Select Data source to define the start value dynamically using a variable value. Select or create a variable containing a numeric value for start.
- **End value**: ending point reference. Select Data source to define the start value dynamically using a variable value. Select or create a variable containing a numeric value for start.

  TIP

  Negative values are permitted for **Start value** and **End value**.

- **Save loop value to a variable**: saves the current loop step value in an existing or a newly created variable. The loop step value is allowed to contain any value between start and end value. Save the value in order to reuse it in another action to identify the current iteration.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.13.2. Use Data Filter

This action executes filter rules on the input data source. As a result, the action extracts fields from input data and maps their values to the linked variables.

Use data filter action executes the selected filter and assigns the variables with respective values.

- **Elements on lower level:** The action can create sub-level elements, identified with **"for each line"** or **"for each data block in ..."**. When you see those, the filter will extract the data not on the document level (with hard-coded field placement), but relatively from the sub areas that contain repeatable sections. In this case make sure that you position your actions below such elements. You have to nest the action under such elements.
- **Mapping variables to fields:** The mapping between trigger variables and filter fields is defined either manually, or is automated, dependent on how the filter is configured. If you have manually defined fields in the filter, you also have to manually map fields to the corresponding variable.

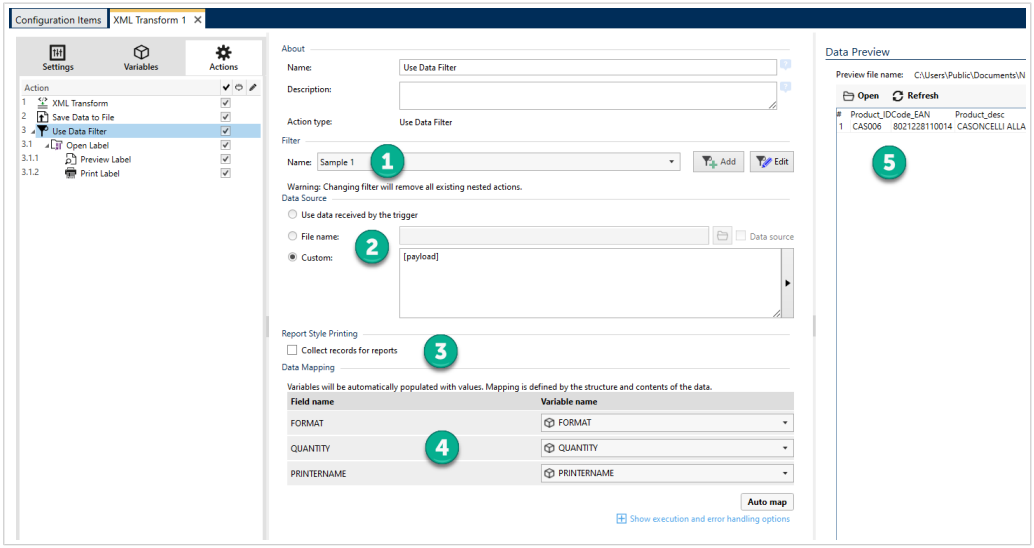
NOTE

It's a good practice to define fields using the same names as are names of the label variables. In this case the button **Auto map** maps matching names automatically.

- **Testing the execution of filter:** When the mapping of variables to fields is done, you can test the execution of the filter. The result will be shown on-screen in table. Number of lines in the table represent the number times actions will execute in the selected level. The column names represent the variable names. The cells contain values as assigned to the respective variable by the filter. The default preview file name is inherited from the filter definition, you can execute filter on any other file.
- **Collect records for reports** collects your data so you use data filters to create reports. For more information Section 9.7, “Automating Reports”.

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.



Navigating your data filter interface.

1. Filter selection.
2. Data your filter uses to execute rules.
3. Report records collection.
4. Mapping your fields (from filter) to variables (from label/trigger).
5. Filter execution preview.

For more information, see sections Understanding Filters and section Examples in NiceLabel Automation user guide.

Filter group allows you to select which filter should be used.

- **Name:** specifies the name of the filter you want to apply. It can either be hard-coded or dynamically defined using an existing or a newly created variable. The list contains all filters defined in the current configuration. You can use the bottom three items in the list to create a new filter.

NOTE

Selecting another filter removes all actions which are nested under this action. If you want to keep the currently defined actions, move them outside of the **Use Data Filter** action. If actions are accidentally lost, **Undo** your action and revert to the previous configuration.

Data Source group allows you to define the contents you want to send to the printer.

- **Use data received by the trigger:** selects the trigger-received data to be used in a filter. In this case, the action uses the original data received by the trigger and execute the filter rules upon it.

Example

If you use a file trigger, the data represents content of the monitored file. If you use a database trigger, the data is a data set returned from the database. If you use a TCP/IP trigger, the data is raw content received via socket.

- **File name:** defines path and file name of the file which contains the data upon which the filter rules will be executed. The content of the specified file is used in a filter. The option **Data source** enables the variable file name. You must select or create a variable that contains the path and/or file name.
- **Custom:** defines custom content to be parsed by the filter. You may use fixed content, mix of fixed and variable content, or variable content alone. To insert variable content, click the button with arrow to the right of data area and insert a variable from the list. For more information, see section Using Compound Values in NiceLabel Automation User Guide.

Data Preview field provides an overview of the filter execution process after the content of the previewed file name is read and the selected filter is applied to it.

The rules in the filter extract fields. The table displays the result of the extraction. Each line in the table represents data for a single label. Each column represents a variable.

To be able to observe result, configure the mapping of fields with matching variables. Depending on the filter definition, you could map the variables to fields manually, or have it done automatically.

- **Preview file name:** specifies the file that contains the data that is going to be parsed through the filter. The preview file is copied from the filter definition. If you change the preview file name, the new file name is saved.
- **Open:** selects another file upon which you want to execute filter rules.
- **Refresh:** re-runs filter rules upon the contents of the preview file name. The **Data Preview** field gets updated with the result.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.13.3. For Every Record

This action executes subordinate nested actions multiple times. All of the nested actions are executed in a loop as many times as there are records in the form table with a connected database.

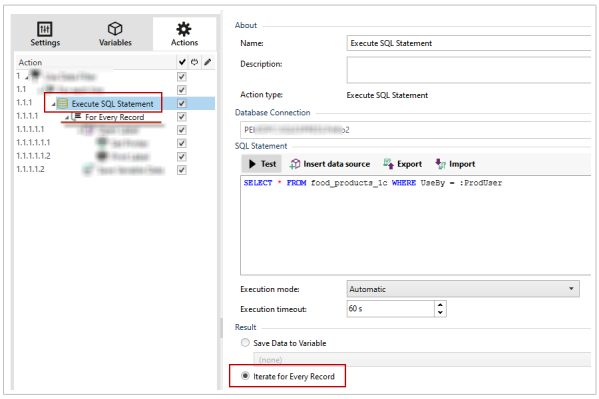
About group identifies the selected action.

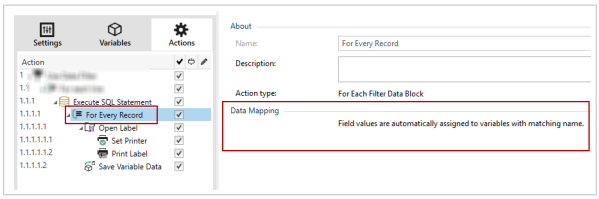
- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Settings group selects the records.

- **Form table**: form table that contains records for which an action should repeat.
- **Use all records**: repeats an action for all records in a defined table.
- **Use selected record**: repeats an action for the selected records only.

If you use the action **Execute SQL Statement** with enabled option **Iterate for Every Record**, NiceLabel automatically inserts **For every Record** action. A note appears about the automatic mapping of your variables.





Prompt variables on your label automatically connect to your database fields with the same names. See the instructions on how to create your solutions:

IMPORTANT

When you create your solution with a database connection, use prompt variables on your labels instead of database fields.

Use the same names for prompt variables as are defined for database fields, for example:

Database field: `food_products_1c.ProdCode`

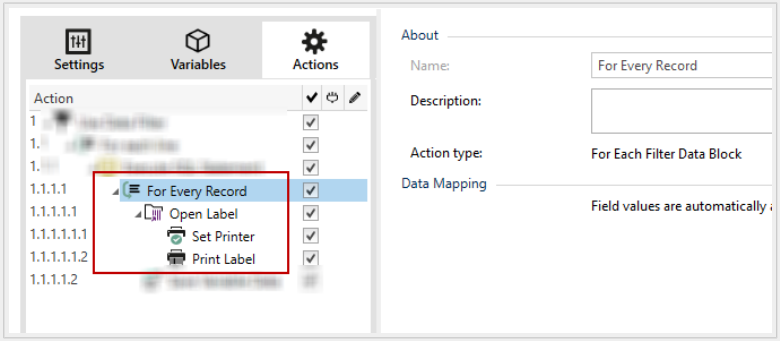
Prompt variable on your label: `ProdCode`

NiceLabel then automatically maps corresponding variables with database fields.

Example

Your solution is connected to your database. Your configuration filters requested database records and you would like to print filtered records on your labels.

Use action **For Every Record** and nest actions to print your labels.



Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.14. Data & connectivity

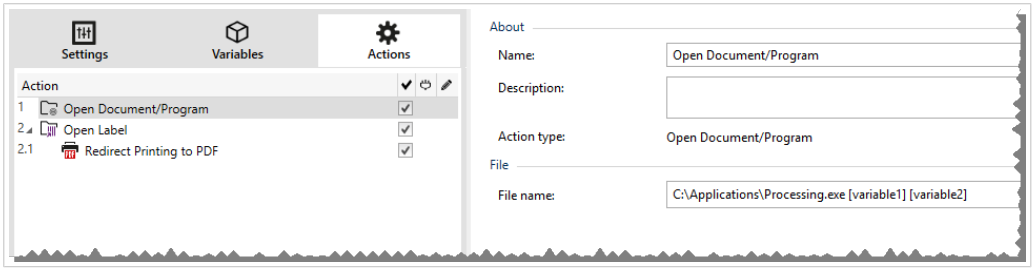
5.14.1. Open Document/Program

This action provides an interface with an external application and opens it using a command-line.

External applications can execute additional processing and provide result back to the NiceLabel 10. This action allows it to bind with any 3rd party software that can execute some additional data processing, or acquire data. External software can provide data response by saving it to file, from where you can read it into variables.

You can feed the value of variable(s) to the program by listing them in the command-line in square brackets.

```
C:\Applications\Processing.exe [variable1] [variable2]
```



> **NOTE**
> If you use this action in NiceLabel 10 solutions, it allows you to open web pages or create email messages directly from your forms. See section Creating hyperlinks and sending emails on form in NiceLabel 10 user guide.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

File group defines the file to be opened.

- **File name**: location and file name of the file or application to be opened.
  The selected file name can be hard-coded, and the same file is going to be used every time. If only a file name without path is defined, the folder with NiceLabel Automation configuration file (.MISX) is used. You can use a relative reference to the file name, in which the folder with .MISX file is used as the root folder.
  Data source: enables variable file name. Select a variable that contains the path and/or file name or combine several variables that create the file name. For more information see section Using Compound Values in NiceLabel Automation User Guide.

  > **NOTE**
  > Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation User Guide.

Execution Options group sets program opening details.

- **Hide window**: renders the window of the opened program invisible. Because NiceLabel 10 is run as a service application within its own session, it cannot interact with desktop, even if it runs with the privileges of the currently logged user. Microsoft has prevented this interaction in Windows Vista and newer operating systems for security reasons.
- **Wait for completion**: specifies for action execution to wait for this action to be completed before continuing with the next scheduled action.

TIP

Enable this option if the action that follows depends on the result of the external application.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.14.2. Save Data to File

PRODUCT LEVEL INFO

Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action saves variable value or other data streams (such as binary data) in a selected file. The NiceLabel Automation service must have write access to the defined folder.

File group defines the file to be opened.

- **File name**: location of the file to be opened within this action.
  Path and file name can be hard-coded, and the same file is going to be used every time. If only a file name without path is defined, the folder with NiceLabel Automation configuration file (.MISX) is used. You can use a relative reference to the file name, in which the folder with .MISX file is used as the root folder.
  Data source: enables variable file name. Select a variable that contains the path and/or file name or combine several variables that create the file name. For more information, see section Using Compound Values in NiceLabel Automation User Guide.

If file exists group handles options in case of an already existing file.

- **Overwrite the file**: overwrites existing data with new data. The old content is lost.
- **Append data to the file**: appends variable values to the existing data files.

Content group defines which data is going to be written in the specified file.

- **Use data received by the trigger:** original data as received by the trigger is going to be saved in the file. Effectively, this option makes a copy of the incoming data.
- **Custom**: saves content as provided in the text area. Fixed values, variable values and special characters are permitted. To enter variables and special characters, click the arrow button to the right of the text area. For more information, see section Combining Values in an Object in NiceLabel Automation User Guide.
- **Encoding**: encoding type for the sent data. **Auto** defines the encoding automatically. If needed, select the preferred encoding type from the drop-down list.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.14.3. Read Data from File

PRODUCT LEVEL INFO

The described feature is available in **NiceLabel LMS Enterprise** and **NiceLabel LMS Pro**.

This action reads content of the provided file name and saves it in a variable. Content of any file type, including binary data can be read.

Usually, Automation Builder module receives data for label printing using a trigger. E.g. if a file trigger is used, the content of trigger file is automatically read and can be parsed by filters. However, you might want to bypass filters to obtain some external data. Once you execute this action and have the data stored in a variable, you can use any of the available actions to use the data.

This action is useful:

- If you must combine data received by the trigger with data stored in a file.

WARNING

If you load data from binary files (such as bitmap image or print file), make sure the variable to store the read content is defined as a **binary variable**.

- When you want to exchange data between triggers. One triggers prepares data and saves it to file (using the Save Data to File action), the other trigger reads the data.

File group defines the file to read the content from.

- **File name**: location of the file to be read within this action.
  Path and file name can be hard-coded, and the same file is going to be used every time. If only a file name without path is defined, the folder with NiceLabel Automation configuration file (.MISX) is used. You can use a relative reference to the file name, in which the folder with .MISX file is used as the root folder.
  Data source: enables variable file name. Select a variable that contains the path and/or file name or combine several variables that create the file name. For more information see section Using Compound Values in NiceLabel Automation User Guide.

  NOTE
  Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation User Guide.

Content group sets file content related details.

- **Variable**: variable that stores the file content. At least one variable (existing or newly created) should be defined.
- **Encoding**: encoding type for the sent data. **Auto** defines the encoding automatically. If needed, select the preferred encoding type from the drop-down list.

  NOTE
  Encoding cannot be selected if the data is read from a binary variable. In this case, the variable contains the data as-is.

Retry on Failure group defines how the action execution should continue if the specified file becomes inaccessible.

TIP
Automation Builder module might become unable to access the file, because it is locked by another application. If an application still writes data to the selected file and keeps it locked in exclusive mode, no other application can open it at the same time, not even for reading. Other possible causes for action retries are: file doesn't exist (yet), folder does not exist (yet), or the service user doesn't have the privileges to access the file.

- **Retry attempts**: defines the number of retry attempts for accessing the file. If the value is set to 0, no retries are made.
- **Retry interval**: time interval between individual retries in milliseconds.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.14.4. Delete File

PRODUCT LEVEL INFO

The described feature is available in **NiceLabel LMS Enterprise** and **NiceLabel LMS Pro**.

This action deletes a selected file from a drive.

NiceLabel Automation module runs as service under a defined Windows user account. Make sure that account has the permissions to delete the file in a specified folder.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.

- **Action type**: read-only information about the selected action type.

File group sets the file related details.

- **File name**: the name of the file to be deleted. **File name** can be hard-coded. **Data source** dynamically defines the **File name** using an existing or newly created variable.
  Path and file name can be hard-coded, and the same file is going to be used every time. If only a file name without path is defined, the folder with NiceLabel Automation configuration file (.MISX) is used. You can use a relative reference to the file name, in which the folder with .MISX file is used as the root folder.
  Data source option enables variable file name. Select or create a variable that contains the path and/or file name or combine several variables that create the file name. For more information see section Using Compound Values in NiceLabel Automation User Guide.

  NOTE
  Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources in NiceLabel Automation User Guide.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE
  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.14.5. Execute SQL Statement

PRODUCT LEVEL INFO

Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action sends SQL commands to a connected SQL server and collects results. Use commands SELECT, INSERT, UPDATE, and DELETE.

Use Execute SQL Statement action to achieve these two goals:

- **Obtain additional data from a database:** In Automation Builder module, a trigger receives data for label printing, but not all of the required values. For example, a trigger receives values for **`Product ID`** and **`Description`**, but not for **`Price`**. We have to look up the value for **`Price`** in the SQL database.
  SQL code example:

  ```
  SELECT Price FROM Products
  WHERE ID = :[Product ID]
  ```

  The **`ID`** is field in the database, **`Product ID`** is a variable defined in the trigger.

- **Update or delete records in a database:** After a label is printed, update the database record and send a signal to the system that the particular record has already been processed.
  SQL code example:
  Set the table field **`AlreadyPrinted`** value to **`True`** for the currently processed record.

  ```
  UPDATE Products
  SET AlreadyPrinted = True
  WHERE ID = :[Product ID]
  ```

  Or delete the current record from a database, because it's not needed anymore.

  ```
  DELETE FROM Products
  WHERE ID = :[Product ID]
  ```

  The **`ID`** is field in the database, **`Product ID`** is a variable defined in the trigger.

NOTE

To use values of variables inside SQL statements, insert colon (:) in front of variable names. This signals that a variable name follows.

IMPORTANT

When you create your solution with a database connection, use prompt variables on your labels instead of database fields.

Use the same names for prompt variables as are defined for database fields, for example:

Database field: `food_products_1c.ProdCode`

Prompt variable on your label: `ProdCode`

NiceLabel then automatically maps corresponding variables with database fields.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Database Connection group defines the database connection that is used for the statement.

TIP

Before you can send an SQL sentence to a database, set up the database connection. Click the **Define** button and follow the on-screen instructions. You can connect to a data source that can be controlled using SQL commands, so you cannot use text (CSV) or Excel files.

SQL Statement group defines an SQL statement or query to be executed.

TIP

Statements from Data Manipulation Language (DML) are allowed to execute queries upon existing database tables.

Use standard SQL statements, such as SELECT, INSERT, DELETE and UPDATE, including joins, function and keywords. The statements in DDL language that are used to create databases and tables (CREATE DATABASE, CREATE TABLE), or to delete them (DROP TABLE) are not permitted.

- **Test**: opens **Data Preview** section. Simulate execution (selected by default) tests the execution of SQL statements. Click **Execute** to run the simulation.

TIP

Data Preview section allows you to test the execution of your SQL statement upon a live set of data. To protect the data from accidental updates, make sure the option **Simulate execution** is enabled. The statements INSERT, DELETE and UPDATE will execute. This enables you to gain feedback on how many records will be affected, then all of the transactions will be reversed.

If you use trigger variables in the SQL statement, you will be able to enter their values for the test execution.

- **Insert data source**: inserts predefined or newly created variables into an SQL statement.
- **Export/Import**: enables exporting and importing SQL statements to/from an external file.
- **Execution mode**: specifies the explicit mode of SQL statement execution.

TIP

In cases of complex SQL queries, it becomes increasingly difficult to automatically determine what is the supposed action. If the built-in logic has troubles identifying your intent, manually select the main action.

  - **Automatic**: determines the action automatically.
  - **Returns set of records (SELECT)**: receives the data set with records.
  - **Does not return set of records (INSERT, DELETE, UPDATE)**: use this option if executing a query that does not return the records. Either insert new records, delete or update the existing records. The result is a status response reporting the number of rows that were affected by your query.
- **Execution timeout**: allows you to define the time delay for sending your commands to the SQL server. Use the execution timeout if you are sending multiple consecutive SQL commands that require longer processing time.
  Type the requested timeout duration in seconds. By default, the execution timeout duration is 60 s. If you want your database provider to define the timeout, type in 0 s.

Result group allows you to set how the SQL statement result should be stored, and to define action iteration.

- **Save Data to Variable**: selects or creates a variable to store the SQL statement result. This option depends on the selected **Execution mode**.
  - **Result of SELECT statement.** After you execute a SELECT statement, it results in a data set of records. You receive a CSV-formatted text content. The first line contains field names returned in a result. The next lines contain records.

> **NOTE**
>
> To extract the values from the returned data set and to use them in other actions, define and execute the action Use Data Filter upon the contents of this variable (this action is available in Automation Builder).

- **Result of INSERT, DELETE and UPDATE statements.** If you use INSERT, DELETE and UPDATE statements, the result is a number indicating the number of records affected in the table.

- **Iterate for Every Record**. If enabled, NiceLabel automatically adds a new action **For Every Record**. See more about this action in a dedicated topic.

> **NOTE**
>
> Automatic database field mapping is enabled. Prompt variables on your label automatically connect to your database fields with the same names. For example:
>
> Database field: `food_products_1c.ProdCode`
>
> Prompt variable on your label: `ProdCode`

Retry on failure group allows you to configure the action to continually retry establishing the connection to a database server in case the first attempt is unsuccessful. If the action fails to connect within the defined number of attempts, an error is raised.

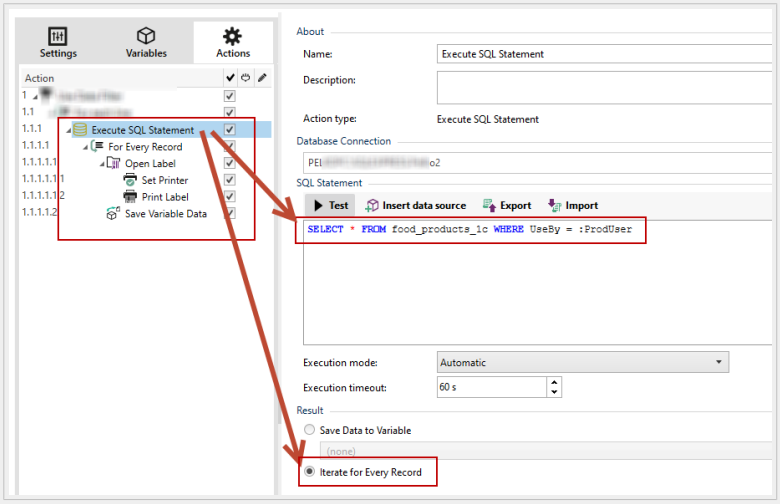
- **Retry attempts**: specifies the number of tries to connect to the database server.
- **Retry interval**: specifies the duration of time between individual retry attempts.

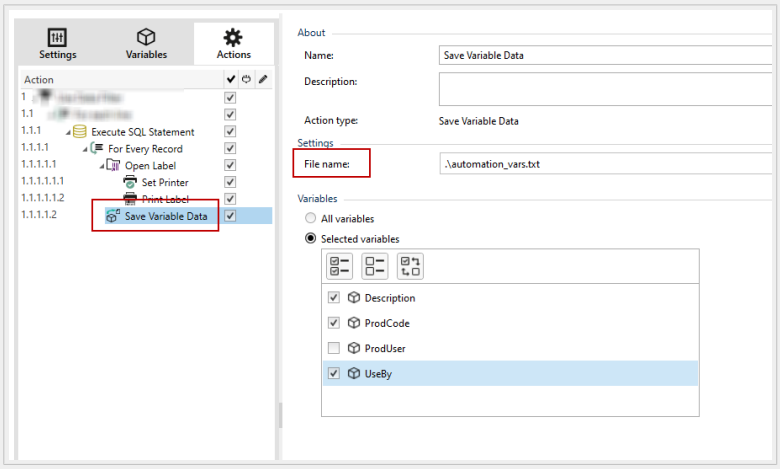
Example

You would like to print labels with data from your `food_products_1c` database but only records with predefined field value `UseBy`.

You define `UseBy` value with variable `ProdUser`. In this case, **ProdUser** value is "3".

After each label is printed, NiceLabel writes database values in a text file on your disc. Use the following actions:





Only labels where `UseBy` value equals "3" are printed and values are written in a text file:

```
"Description";"ProdCode";"UseBy"
"Choux pastry";"977000000011";"3"
"Croline";"977000000012";"3"
"Danish pastry";"977000000013";"3"
"Kouign-amann";"977000000022";"3"
"Muskazine";"977000000026";"3"
"Pineapple bun";"977000000029";"3"
"Schneeball";"977000000033";"3"
"Schuxen";"977000000034";"3"
"Tortell";"977000000041";"3"
```

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.

- **Condition:** defines one-line programming expression that must provide a Boolean value (**`true`** or **`false`**). When the result of the expression is **`true`**, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.14.6. Send Data to TCP/IP Port

This action sends data to any external device which accepts TCP/IP connection on a predefined port number.

Send Data to TCP/IP Port establishes connection with a device, sends the data and terminates the connection. The connection and communication is governed by the client – server handshake that occurs when initiating or terminating the TCP connection.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Connection Settings group sets connection details.

- **Reply to sender**: Enables direct reply to the socket from which the trigger data originates. Use this option to provide feedback about the printing process.

> **NOTE**
> This option is available in NiceLabel Automation.

Prerequisites for **Reply to sender** setting are:

- Remote party does not close the communication channel, once the message gets delivered.
- **Send Data to TCP/IP Port** action is used within the **TCP/IP Server** trigger.
- Do **not** configure the Execution Event in the **TCP/IP Server** trigger as **On client disconnect**.

- **Destination (IP address:port)**: destination address and port of the TCP/IP server. Hard-code the connection parameters and use fixed host name or IP address or use variable connection parameters by clicking the right arrow and selecting a predefined variable. For more information, see section Combining Values in an Object in NiceLabel Automation user guide.

Example

If the variable **`hostname`** provides the TCP/IP server name and the variable **`port`** provides the port number, enter the following parameter for the destination:

```
[hostname]:[port]
```

- **Disconnect delay**: prolongs the connection with the target socket for the defined time intervals after the data has been delivered. Certain devices require more time to process the data. Insert the delay value manually or click the arrows to increase or decrease it.
- **Save data reply in a variable**: selects or creates a variable that stores the server reply. Any data received from the TCP/IP server after passing the "disconnect delay" is stored in this variable.

Content group defines the content to be sent to a TCP/IP server.

> **TIP**
> Use fixed content, mix of fixed and variable content, or variable content alone. To enter variable content, click the button with arrow to the right of data area and insert a variable from the list. For more information, see section Combining Values in an Object in NiceLabel Automation user guide.

- **Data**: content to be sent outbound.
- **Encoding**: encoding type for the sent data. Auto defines the encoding automatically. If needed, select the preferred encoding type from the drop-down list.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.14.7. Send Data to Serial Port

This action sends data to a serial port. Use it to communicate with external serial-port devices.

TIP

Make sure the port settings match on both ends – in the configured action and on the serial-port device. Serial port can be used by a single application in the machine. To successfully use the port from this action, no other application may use the port at the same time, not even any printer driver.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.

- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Port group defines the serial port.

- **Port name**: name of the port to which the external device connects to. This can either be a hardware COM port or a virtual COM port.

Port Settings group defines additional port connection settings.

- **Bits per second**: speed rate used by the external device to communicate with the PC. The usual alias used with the setting is "baud rate". Select the value from the drop-down menu.
- **Data bits**: number of data bits in each character. 8 data bits are almost universally used in newer devices. Select the value from the drop-down menu.
- **Parity**: method of detecting errors in a transmission. The most common parity setting, is "none", with error detection handled by a communication protocol (flow control). Select the value from the drop-down menu.
- **Stop bits**: halts the bits sent at the end of every character allowing the receiving signal hardware to detect the end of a character and to resynchronize with the character stream. Electronic devices usually use a single stop bit. Select the value from the drop-down menu.
- **Flow control**: serial port may use interface signals to pause and resume the data transmission.

Content group defines the content to be sent to serial port.

TIP

Fixed content, mix of fixed and variable content, or variable content alone are permitted. To enter variable content, click the button with arrow to the right of data area and insert a variable from the list. For more information, see section Combining Values in an Object in NiceLabel Automation user guide.

- **Data**: content to be sent outbound.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.

- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.14.8. Read Data from Serial Port

This action collects data received via serial port (RS-232) and saves it in a selected variable. Use this action to communicate with external serial port devices.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Port group defines the serial port.

- **Port name**: name of the port to which an external device connects to. This can either be a hardware COM port or a virtual COM port.

Port Settings group defines additional port connection settings.

- **Bits per second**: speed rate used by the an external device to communicate with the PC. The usual alias used with the setting is "baud rate".
- **Data bits**: specifies the number of data bits in each character. 8 data bits are almost universally used in newer devices.
- **Parity**: specifies the method of detecting errors in a transmission. The most common parity setting, is "none", with error detection handled by a communication protocol (flow control).
- **Stop bits**: halts the bits sent at the end of every character allowing the receiving signal hardware to detect the end of a character and to resynchronize with the character stream. Electronic devices usually use a single stop bit.
- **Flow control**: serial port may use interface signals to pause and resume the data transmission.

Example

A slow device might need to handshake with the serial port to indicate that data should be paused while the device processes received data.

Options group includes the following settings:

- **Read delay**: optional delay when reading data from serial port. After the delay, the entire content of the serial port buffer is read. Enter the delay manually or click the arrows to increase or decrease the value.
- **Send initialization data**: specifies the string that is sent to the selected serial port before the data is read. This option enables the action to initialize the device to be able to provide the data. The option can also be used for sending a specific question to the device, and to receive a specific answer. Click the arrow button to enter special characters.

Data Extraction group defines how the defined parts of received data are extracted.

- **Start position**: starting position for data extraction.
- **End position**: ending position for data extraction.

Result group defines a variable for data storing.

- **Save data to variable**: select or create a variable to store the received data.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.

- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  > **NOTE**
  > Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.14.9. Send Data to Printer

This action sends data to a selected printer. Use it to send pre-generated printer streams to any available printer.

NiceLabel Automation module uses the installed printer driver in pass-through mode just to be able to send data to the target port, such as LPT, COM, TCP/IP or USB port, to which the printer is connected.

> **NOTE**
> Possible scenario. Data received by the trigger must be printed out on the same network printer, but on different label templates (.NLBL label files). The printer can accept data from various workstations and will usually print the jobs in the received order. Automation Builder module will send each label template in a separate print job, making it possible for another workstation to insert its job between the jobs created in our own Automation Builder module. Instead of sending each job separately to the printer, merge all label jobs (using the action Redirect Printing to File) and send a single "big" print job to the printer.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Printer group selects the printer.

- **Printer name**: name of the printer to send the data to. Select the printer from the drop-down list of locally installed printer drivers, enter a custom printer name, or define it dynamically using an existing or newly created variable.

Data Source group defines the content to be sent to printer.

- **Use data received by the trigger:** trigger-received data it used. In this case, you want the received printer stream to be used as an input to the filter. Your goal is to redirect it to a printer without any modification. The same result can be achieved by enabling the internal variable **`DataFileName`** and using the contents of the file it refers to. For more information, see section Using Compound Values in NiceLabel Automation user guide.
- **File name**: path and file name of the file containing a printer stream. Content of the specified file is sent to a printer. Select **Data source** to define the file name dynamically using a variable value.
- **Variable**: variable (existing or new) that contains the printer stream.
- **Custom**: defines custom content to be sent to a printer. Fixed content, mix of fixed and variable content, or variable content alone are permitted. To enter variable content, click the button with arrow to the right of data area and insert a variable from the list. For more information, see section Combining Values in an Object in NiceLabel 10 user guide.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (**`true`** or **`false`**). When the result of the expression is **`true`**, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.14.10. HTTP Request

PRODUCT LEVEL INFO

Automation Builder features require **LMS Enterprise**.

This action sends data to the destination Web server using the selected HTTP method. HTTP and HTTPS URI schemes are allowed.

HTTP works as a request-response protocol in client-server computing model. In this action, NiceLabel 10 acts as a client that communicates with a remote server. This action submits a selected HTTP request message to a server. The server returns a response message, which can contain completion status information about the request and may also contain requested content in its body.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Connection Settings group sets connection parameters.

NOTE

This action supports Internet Protocol version 6 (IPv6).

- **Destination**: address, port and destination (path) of the Web server.

TIP

If a Web server runs on default port 80, skip the port number. Hard-code the connection parameters and use a fixed host name or IP address. Use a variable value to define this option dynamically. For more information, see section Using Compound Values in NiceLabel Automation user guide.

Example

If the variable **`hostname`** provides the Web server name and the variable **`port`** provides the port number, you can enter the following for the destination:

```
[hostname]:[port]
```

- **Request method**: available request methods.
- **Timeout**: timeout duration (in ms) during which the server connection should be established and response received.
- **Save status reply in a variable**: variable to store the status code received from the server.

TIP

Status code in range 2XX is a success code. Common "OK" response is code 200. Codes 5XX are server errors.

- **Save data reply in a variable**: variable to store the data received from the server.

Authentication group enables you to secure the Web server connection.

- **Enable basic authentication**: allows you to enter the required credentials to connect to the Web server. User name and password can either be fixed or provided using a variable.

NOTE

HTTP Basic authentication (BA) uses static standard HTTP headers. The BA mechanism provides no confidentiality protection for the transmitted credentials. They are merely encoded with Base64 in transit, but are not encrypted or hashed in any way. Basic Authentication should be used over HTTPS.

- **Show password**: unmasks the password characters.

Content group defines the contents to be sent to a Web server.

- **Data**: content to be sent outbound. Fixed content, mix of fixed and variable content, or variable content alone are permitted. To enter variable content, click the button with arrow to the right of

data area and insert variable from the list. For more information, see section Combining Values in an Object in NiceLabel 10 user guide.

- **Encoding**: encoding type for the sent data.

  TIP

  Auto defines the encoding automatically. If needed, select the preferred encoding type from the drop-down list.

- **Type**: Content-Type property of the HTTP message. If no type is selected, the default **`application/x-www-form-urlencoded`** is used. If an appropriate type is not listed, define a custom one or set a variable that would define it dynamically.

Additional HTTP Headers are requested by certain HTTP servers (especially for REST services).

- **Additional headers**: hard coded headers or headers obtained from variable values. To access the variables, click the small arrow button to the right hand side of the text area. For more information, see section Combining Values in an Object in NiceLabel 10 user guide.
  Certain HTTP servers (especially for REST services) require custom HTTP headers to be included in the message. This section allows you to provide the required HTTP header.
  HTTP headers must be entered using the following syntax:

  ```
  header field name: header field value
  ```

  For example, to use the header field names **`Accept`**, **`User-Agent`** and **`Content-Type`**, you could use the following syntax:

  ```
  Accept: application/json; charset=utf-8
  User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/31.0.1650.63 Safari/537.36
  Content-Type: application/json; charset=UTF-8
  ```

  You can hard code the header field names, or you can obtain their values from trigger variables. Use as many custom header fields as you want, just make sure that each header field is placed in a new line.

  NOTE

  The entered HTTP headers override the already defined headers elsewhere in the action properties, such as **Content-Type**.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.14.11. Web Service

PRODUCT LEVEL INFO

Automation Builder features require **LMS Enterprise**.

Web Service is a method of communication between two electronic devices or software instances. Web Service is defined as a data exchange standard. It uses XML format to tag the data, SOAP protocol is used to transfer the data, and WSDL language is used to describe the available services.

This action connects to a remote Web service and executes the methods on it. Methods can be described as actions that are published on the Web Service. The action sends inbound values to the selected method in the remote Web service, collects the result and saves it in selected variables.

After importing the WSDL and adding a reference to the Web Service, its methods are listed in the Method combo box.

NOTE

You can transfer simple types over the Web Service, such as string, integer, boolean, but not the complex types. The WSDL must contain single binding only.

NOTE

You plan to print product labels. Your trigger would receive only a segment of the required data. E.g. the trigger receives the value for **`Product ID`** and **`Description`** variables, but not the **`Price`**. The price information is available in a separate database, which is accessible over Web service call. Web service defines the function using a WSDL definition. For example, function input is **`Product ID`** and its output is **`Price`**. The Web Service action sends **`Product ID`** to the Web service. It executes and makes an internal look up in its database and provide the matching **`Price`** as the result. The action saves the result in a variable, which can be used on the label.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Web Service Definition group includes the following settings:

NOTE

This action supports Internet Protocol version 6 (IPv6).

- **WSDL**: location of WSDL definition.
  WSDL is usually provided by the Web service. Typically, you would enter the link to WSDL and click **Import** to read the definition. If facing troubles while retrieving the WSDL from online resource, save the WSDL to a file and enter the path with file name to load the methods from it. NiceLabel 10 automatically detects if the remote Web Service uses a document or RPC syntax, and whether it communicates appropriately or not.
- **Address**: address at which the Web Service is published.
  Initially, this information is retrieved from the WSDL, but can be updated before the action is executed. This is helpful for split development / test / production environments, where the same list of actions is used, but with different names of servers on which the Web Services run.
  Fixed content, mix of fixed and variable content, or variable content alone are permitted. To enter variable content, click the button with arrow to the right hand side of data area and insert variable

from the list. For more information, see section Combining Values in an Object in NiceLabel 10 user guide.

- **Method**: methods (functions) that are available for the selected Web service. The list is automatically populated by the WSDL definition.
- **Parameters**: input and output variables for the selected method (function).
  Inbound parameters expect an input. For testing and troubleshooting reasons, you can enter a fixed value and see the preview result on-screen. Typically, you would select a variable for inbound parameter. Value of that variable will be used as input parameter. The outbound parameter provides the result from the function. You must select the variable that will store the result.
- **Timeout**: timeout after which the connection to a server is established.

Authentication enables basic user authentication. This option defines user credentials that are necessary to establish an outbound call to a remote web service.

- **Enable basic authentication**: enables defining the **Username** and **Password** that can be entered manually or defined by variable values. Select **Data sources** to select or create the variables.
- **Show password**: uncovers the masked **Username** and **Password** characters.
  Details about security concerns, are available in section Securing Access to your Triggers in NiceLabel Automation user guide.

Data Preview field allows you to perform a test Web service execution.

- **Execute** button executes a Web service call.
  It sends values of inbound parameters to the Web service and provides the result in the outbound parameter. Use this functionality to test the execution of a Web service. You can enter values for inbound parameters and see the result on-screen. When satisfied with execution, replace the entered fixed value for inbound parameter with a variable from the list.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.15. Other

5.15.1. Get Label Information

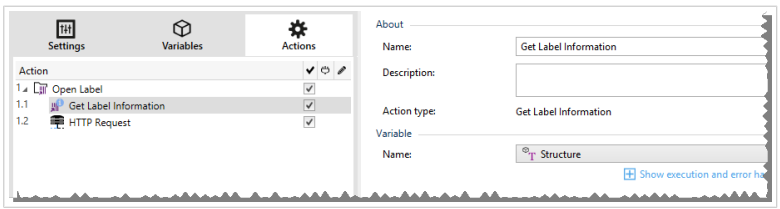
PRODUCT LEVEL INFO

Automation Builder features require **LMS Enterprise**.

This action returns structural information about the associated label file. It provides information about the label dimensions, printer driver and lists all label variables, and their main properties.

The Get Label Information action returns the original information as saved in the label file. Additionally, it also provides information after the print process has been simulated. The simulation ensures that all labels variables get the value as they would have during a normal print. Also, the label height information provides correct dimensions in case you define the label as a variable-height label (in this case, the label size depends on the amount of data to be printed). The action returns the dimensions for a label size, not for a page size.

The action saves label structure information in a selected variable. You can then send the data back to the system using the HTTP Request action (or a similar outbound data connectivity action), or send it back in trigger response, if you use a bidirectional trigger.



NOTE

This action must be nested under the Open Label action.

Variable group selects or creates a variable that stores the structural information about a label.

- **Name:** specifies the variable name. Select or create a variable which stores the XML-formatted label information.
  - If you want to use the information from the XML inside this trigger, you can define the and execute it with Use Data Filter action (Automation Builder only).
  - If you want to return the XML data as a response in your HTTP or Web Service trigger, use this variable directly in the **Response data** field of the trigger configuration page.
  - If you want to save the XML data to a file, use the Save Data to File action.

Additional settings group allows you to enable the use of provisional values.

- **Use provisional values:** replaces missing data source values with provisional values.

TIP

See section Variable in NiceLabel 10 Designer user guide for detailed description of provisional values.

Sample Label Information XML

The sample below presents a structural view of the label elements and their attributes as they are returned.

```
<?xml version="1.0" encoding="UTF-8"?>
<Label>
    <Original>
        <Width>25000</Width>
        <Height>179670</Height>
        <PrinterName>QLS 3001 Xe</Printer>
    </Original>
    <Current>
        <Width>25000</Width>
        <Height>15120</Height>
        <PrinterName>QLS 3001 Xe</Printer>
    </Current>
    <Variables>
        <Variable>
            <Name>barcode</Name>
            <Description></Description>
```

```
            <DefaultValue></DefaultValue>
            <Format>All</Format>
            <CurrentValue></CurrentValue>
            <IncrementType>None</IncrementType>
            <IncrementStep>0</IncrementStep>
            <IncrementCount>0</IncrementCount>
            <Length>100</Length>
        </Variable>
    </Variables>
</Format>
```

Label Information XML Specification

This section contains a description of the XML file structure as returned by the Get Label Information action.

NOTE

All measurement values are expressed in the 1/1000 mm units. For example width of 25000 is 25 mm.

- **<Label>:** this is a root element.
- **<Original>:** specifies label dimensions and printer name as stored in the label file.
  - **Width:** this element contains the original label width.
  - **Height:** this element contains the original label height.
  - **PrinterName:** this element contains the printer name for which the label has been created for.
- **Current:** specifies label dimensions and printer name after the simulated print has been completed.
  - **Width:** this element contains the actual label width.
  - **Height:** this element contains the actual label height. If a label is defined as a variable-height label, it can increase along with label objects. For example, Text Box and RTF object sizes may increase in vertical direction and cause the label to expand as well.
  - **PrinterName:** this element contains printer name that will be used for printing.

    Example

    A printer different from the original one is going to be used if the original printer driver is not installed on this computer, or if the printer has been changed using the Set Printer action.
- **<Variables> and <Variable>:** the element **`Variables`** contains a list of all prompt label variables, each defined in a separate **`Variable`** element. The prompt variables are the ones listed in the print dialog box when you print label from NiceLabel 10 . If there are no prompt variables defined in the label, the element **`Variables`** is empty.

- **Name:** contains variable name.
- **Description:** contains variable description.
- **DefaultValue:** contains default value as defined for the variable during the label design process.
- **Format:** contains the acceptable type of variable content (characters).
- **IsPrompted:** contains information whether or not the variable is prompted at print time or not.
- **PromptText:** contains text that prompts the user for value input.
- **CurrentValue:** contains the actual value that is used for printing.
- **IncrementType:** contains information, if the variable is defined as a counter or not. If identified as a counter, it tells what kind of counter it is.
- **IncrementStep:** contains information about the counter step. Counter value increments/ decrements for this value on the next label.
- **IncrementCount:** contains information about the point of counter value incrementing/ decrementing. Usually, the counter changes value on every label, but that can be changed.
- **Length:** contains maximum number of stored characters in a variable.
- **IsPickListEnabled:** contains information whether or not the user selects variable values from a pick list.
- i**PickListValues:** contains the actual (selectable) pick list values.

XML Schema Definition (XSD) for Label Specification XML

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="Format" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Label">
        <xs:complexType>
            <xs:all>
                <xs:element name="Original">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Width" type="xs:decimal"
minOccurs="1" />
                            <xs:element name="Height" type="xs:decimal"
minOccurs="1" />
                            <xs:element name="PrinterName" type="xs:string"
minOccurs="1" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Current">
```

```
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Width" type="xs:decimal"
minOccurs="1" />
                            <xs:element name="Height" type="xs:decimal"
minOccurs="1" />
                            <xs:element name="PrinterName" type="xs:string"
minOccurs="1" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Variables">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Variable" minOccurs="0"
maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="Name"
type="xs:string" minOccurs="1" />
                                        <xs:element name="Description"
type="xs:string" minOccurs="1" />
                                        <xs:element name="DefaultValue"
type="xs:string" minOccurs="1" />
                                        <xs:element name="Format"
type="xs:string" minOccurs="1" />
                                        <xs:element name="CurrentValue"
type="xs:string" minOccurs="1" />
                                        <xs:element name="IncrementType"
type="xs:string" minOccurs="1" />
                                        <xs:element name="IncrementStep"
type="xs:integer" minOccurs="1" />
                                        <xs:element name="IncrementCount"
type="xs:integer" minOccurs="1" />
                                        <xs:element name="Length"
type="xs:string" minOccurs="1" />
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:all>
        </xs:complexType>
```

```
    </xs:element>
</xs:schema>
```

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

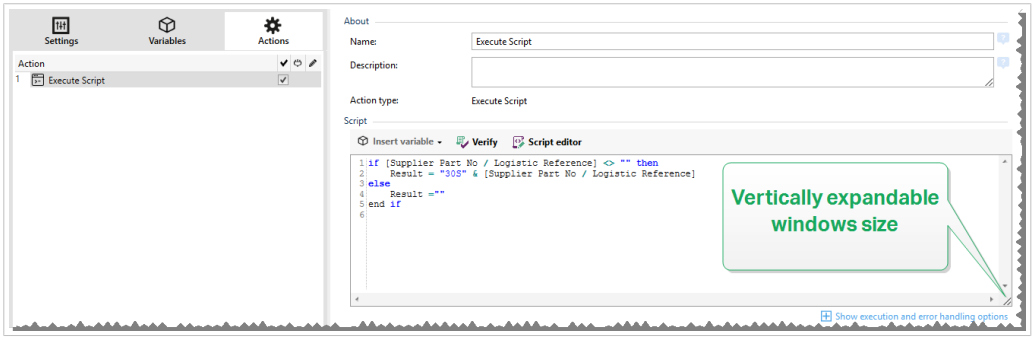
- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.15.2. Execute Script

This action enhances the software functionality by using custom VBScript or Python scripts. Use this function if the built-in actions don't meet your data manipulation requirements.

Scripts can include the trigger variables – both internal variables and the variables defined or imported from labels.

Make sure that Windows account under which the service runs has the privileges to execute the commands in the script.



NOTE

The script type is configured per trigger in the trigger properties. All Execute Script actions within a single trigger must be of the same type.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Script editor offers the following features:

- **Insert data source:** inserts an existing or newly created variable into the script.
- **Verify:** validates the entered script syntax.
- **Script editor:** opens the editor which makes scripting easier and more efficient.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

> **NOTE**
> Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

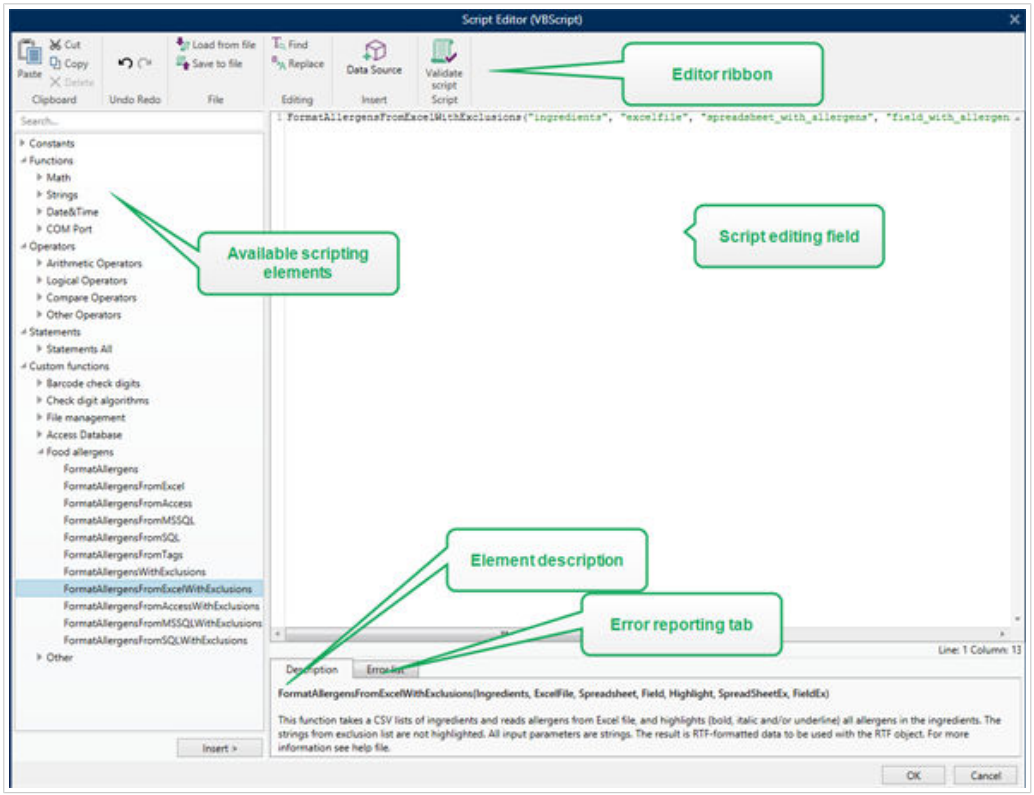
Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.15.2.1. Script Editor

NiceLabel 10 provides a script editor which makes your Python or VBScript scripting easier, error-free and time efficient.



The selection of scripting languages that should be used in Script editor differs between NiceLabel Designer Pro and Automation Builder:

- In Designer, double-click on the form design surface to open Form Properties >Additional Settings > Form Scripting Language.

- In Automation Builder, go to Configuration items > click Edit to open trigger properties >Settings > Other > Scripting.

NiceLabel 10 uses .NET variant of Python named IronPython. It works as a fully compatible implementation of Python scripting language which also supports .NET methods.

Editor Ribbon includes commonly used commands which are distributed over multiple functional groups.

- Clipboard group offers Cut, Copy, Paste and Delete commands.
- Undo Redo group allows undoing or repeating script editing actions.
- File group allows loading and saving scripts in a file.
    - Load from file: loads a script from an external previously saved textual file.
    - Save to file: stores the currently edited script in a textual file.
- Editing group allows finding and replacing strings in a script.
    - Find: locates the entered string in the script.
    - Replace: replaces string in the script.
- Insert group: Data Source command inserts existing or newly defined data sources into the script.
- Script group: Validate script command validates of the entered script's syntax.

Available scripting elements contain all available script items which can be used when building a script. Double-click the element or click the Insert button to insert the element at cursor position into the script.

Element description provides basic information about the inserted script element.

Error list includes the errors which are reported after the Validate script command is run.

5.15.3. Message

Use the Message action to write custom strings (For example, custom warning messages, variable values, and comments). The message action creates custom entries in your Automation Manager log files. Automation log files contain application-generated information, warnings, and error descriptions. Use Message logs to track your message variables during configuration, troubleshooting, and debugging.

To configure Message actions, do the following:

1. Go to **All Actions** and select **Message** from the drop-down **Action** menu.
2. Rename your action and insert your description.
3. Configure your message **Content**: **Caption**, **Message**, and **Message Type**.
   Message Types include:
    - **Information**

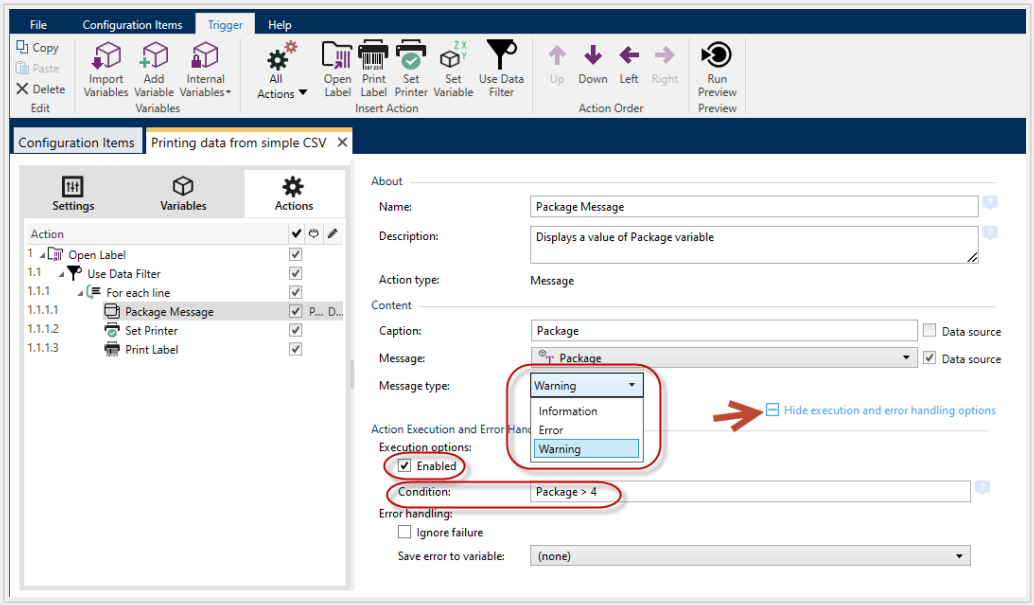
- **Error**
- **Warning**

4. Expand **Show execution and error handling options** to set conditions for showing messages, ignoring failures, and saving Automation errors to variables.

NiceLabel Automation Manager displays colored Messages (For example, red errors and orange warnings) in your Automation Log pane.

Example:

You print Pasta labels with Automation. Your trigger receives variable values from ERP generated CSV files. When your variable "Package" value is greater than 4, Automaton Log creates a warning.



*Configuring **Message** actions.*

Your result in Automation Manager looks like this:

NOTE

When you set your Message severity to Error, your triggers do not go to error state. Printing is still possible.

Use Message logs for:

- configuration troubleshooting
- debugging your solutions
- tracking values of your selected variables

displaying your customized warnings and error messages

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.15.4. Verify License

This action reads the activated license and executes the actions nested below this action only if a certain license type is used.

TIP

Verify License action provides protection of your trigger configuration from being run on unauthorized machines.

NOTE

License key that activates software can also encode the Solution ID. This is the unique number that identifies the solution provider that sold the NiceLabel 10 license.

If the configured Solution ID matches the Solution ID encoded in the license, the target machine is permitted to run nested actions, effectively limiting execution to licenses sold by the solution provider.

The triggers can be further encrypted and locked so only authorized users are allowed to open the configuration. For more information, see section Protecting Trigger Configuration in NiceLabel Automation user guide.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.

- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

License Information group allows you to select the license ID.

- **License ID:** defines the ID number of the licenses that are allowed to run the nested actions.
  - If the entered value is not the License ID that is encoded in the license, the nested actions is not executed.
  - If the entered value is set to 0, the actions execute if a valid license is found.

NOTE

Digital Partner UID can also be used as License ID. This option is available for members of NiceLabel Digital Partner Program.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.15.5. Try

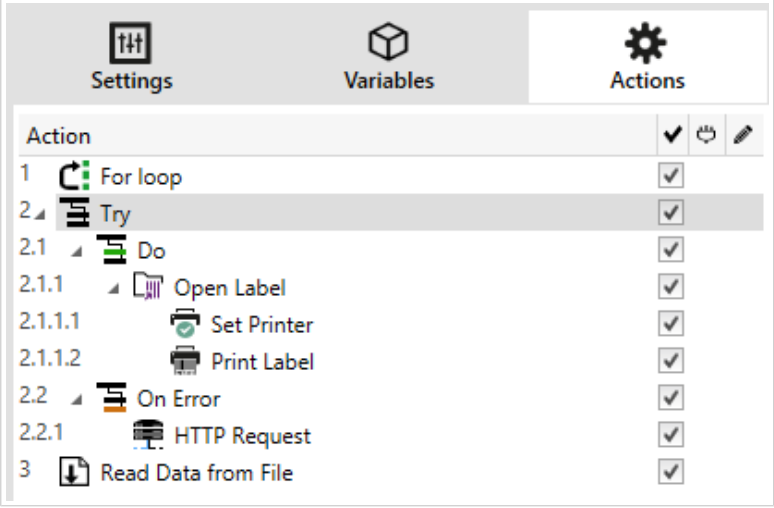
> **PRODUCT LEVEL INFO**
> Automation Builder features require **LMS Enterprise**.

This action allows you to:

- Monitor errors while the actions are being executed.
- Run an alternative set of actions if an error occurs.

Try action creates **Do** and **On error** placeholders for actions. All actions that should execute if a trigger fires, must be placed inside the **Do** placeholder. If no error is detected when executing actions from **Do** placeholder, these are the only actions that ever execute. However, if an error does happen, the execution of actions from **Do** placeholder stops and the execution switches over to actions from **On error** placeholder.

> **NOTE**
> You must enable Synchronous Printing to catch errors with **On error**.



Example

If any action in the Do placeholder fails, the action execution stops and resumes with the actions in the On Error placeholder. If Try would be placed on its own, that would terminate the trigger execution. In our case, Try is nested under the For loop action. Normally, any error in Do placeholder would also stop executing the For loop action, even if there are still further steps until the For loop is complete. In this

case, the Save Data to File action does not execute as well. By default, each error breaks the entire trigger processing.

However, you can also continue with the execution of the next iteration in the For loop action. To make this happen, enable the Ignore failure option in the Try action. If the data from the current step in For Loop causes an error in the Do placeholder, the actions from On Error execute. After that, the Save Data to File in level 2 execute and then the For loop action continues to execute in the next iteration.

TIP

This action provides for an easy error detection and execution of "feedback" or "reporting" actions. For example, if an error happens during trigger processing, you can send out a warning. For more information, see section Print Job Status Feedback in NiceLabel Automation user guide.

NOTE

Important! The **Try** action gives expected results with asynchronous actions. If your Try loop includes the Print Label action that fails, the action execution still completes the Try loop, and does not switch to the **On error** actions as expected. The result for not switching to the **On error** actions is the Print Label action that runs in synchronous mode by default. To avoid this, make sure supervised printing is on. Go to trigger settings > **Other** > **Feedback from the Print Engine** and enable **Supervised printing**.

Read more about supervised printing in section Synchronous Print Mode.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.

- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.15.6. XML Transform

PRODUCT LEVEL INFO

The functionality from this section is available in LMS Enterprise.

This action transforms an XML document into another document using the provided transformation rules. The rules must be provided by a .XSLT definition in a file, or by another variable source.

The action allows you to convert complex XML documents into XML documents with a more manageable structure. XSLT stands for XSL Transformations. XSL stands for Extensible Stylesheet Language, and works as a stylesheet language for XML documents.

XML Transform action stores the converted XML document in the selected variable. The original file is left intact on the disk. If you want to save the converted XML document, use action Save Data to File.

NOTE

Typically, you would use the action to simplify XML documents provided by the host application. Defining XML filter for the complex XML document might take a while, or in some cases the XML is just too complex to be handled. As alternative, you would define the rules to convert XML into structure that can be easily handled by the XML filter, or even skipping the need for a filter altogether. You can convert XML document into natively-supported XML, such as Oracle XML and then simply executing it using the Run Oracle XML Command File action.

TIP

Example for this action is installed with the product. To open it, go to **Help** > **Sample Files** > **XML Transformations** and run the XML Transformations.misx configuration. Details are available in the **Readme** file.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Data Source group defines the XML data to be transformed.

- **Use data received by the trigger:** defines that the trigger-received data it used. The same result can be achieved by enabling the internal variable **`DataFileName`** and using the contents of file it refers to. For more information, see section Using Compound Values in NiceLabel Automation user guide.
- **File name:** defines the path and file name of the file containing the XML file to be transformed. Contents of the specified file is used. Data source enables the file name to be defined dynamically. Select or create a variable that contains the path and/or file name. The action opens the specified file and applies transformation on file contents, which must be XML- formatted.
- **Variable:** selects or creates the variable that contains printer stream. The contents of selected variable is used and it must contain XML structure.

Transformation Rules Data Source (XSLT) group defines the transformation rules (.XSLT document) that are going to be applied to the XML document.

- **File name:** defines path and file name of the file containing the transformation rules (.XSLT).
- **Custom:** defines custom contents. You can use fixed content, mix of fixed and variable content, or variable content alone. To insert a variable content, click the button with arrow to the right of data area and insert variable from the list. For more information, see section Using Compound Values in NiceLabel Automation user guide.

Save Result to Variable group defines the variable to store the transformed file.

- **Variable:** selects or creates a variable that is going to contain the result of the transformation process. E.g. if you use the rules that convert complex XML into simpler XML, the content of the selected variable is a simple XML file.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

5.15.7. Group

This action configures multiple actions within the same container. All actions placed below the **Group** action belong to the same group and are going to be executed together.

This action provides the following benefits:

- **Better organization and displaying of action workflow.** You can expand or collapse the Group action and display the nested actions only when needed. This helps keep the configuration area cleaner.
- **Defining conditional execution.** You can define a condition in the Group action just once, not individually for each action. If the condition is met, all actions inside the Group are executed. This can save a lot of configuration time and can reduce the number of configuration errors. Group action provides a good method to define IF..THEN execution rules for multiple actions.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

NOTE

Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.15.8. Log Event

PRODUCT LEVEL INFO

Automation Builder features require **NiceLabel LMS Enterprise** or **NiceLabel LMS Pro**.

This action logs an event to NiceLabel Control Center for history and troubleshooting purposes.

NOTE

To make Log event action active, make sure that print job logging to NiceLabel Control Center is enabled.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Event Data group provides information about the logged event.

- **Information:** basic description of the event that will be included in the NiceLabel Control Center event log. Up to 255 characters are allowed in this area.
- **Details:** detailed description of the event to be logged in the NiceLabel Control Center. Up to 2000 characters are allowed in this area.

TIP

The descriptions entered in **Information** and **Details** fields allows you to filter out the events in Control Center **All Activities History**. When working with Control Center, go to **History** > **All Activities** > **Define filter**. For more details, read the Control Center User Guide.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (**`true`** or **`false`**). When the result of the expression is **`true`**, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.15.9. Preview Label

PRODUCT LEVEL INFO

Automation Builder features require **LMS Enterprise**.

This action executes the print process and provides label image preview. By default, the preview is saved to disk as JPEG image, but you can choose other image format. You can also control the size of the created preview image. The action generates preview for a single label.

Once you have the label preview created in a file, you can send the file to a third party application using one of the outbound actions, such as Send Data to HTTP, Send Data to Serial Port, Send Data to TCP/IP Port, or use it as response message from bidirectional triggers, such as and Web Service Trigger. The third party application can take the image and show is as label preview to the user.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.

- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Preview group defines the file to be previewed and its details.

- **File name:** specifies the path and file name. If hard-coded, the same file is used every time. If you only use the file name without path, the folder with configuration file (.MISX) is used. You can use a relative reference to the file name, where folder with .MISX file is used as root folder. **Data source** option enables variable file name. Select or create a variable that contains the path and/or file name after a trigger is executed. Usually, the value to the variable is assigned by a filter.
- **Image type:** specifies the image type which is used for saving the label preview.
- **Preview label back side (2-sided labels):** enables preview of the back label. This is useful, if you use double-sided labels and want to preview the label's back side.

Example

For example, if your label template defines dimension as 4" × 3" and the label printer resolution is set to 200 DPI, the resulting preview image has dimensions of 800 × 600 pixels. Width equals 4 inches times 200 DPI, which results in 800 pixels. Height equals 3 inches times 200 DPI, which results in 600 pixels.

Additional settings group allows you to enable the use of provisional values.

- **Use provisional values:** replaces missing data source values with provisional values and displays them in the label preview.

TIP

Provisional value defines a custom placeholder variable value in an object while designing labels or forms. In a label object, the provisional value is replaced by the real variable value at print time.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  > **NOTE**
  > Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables **`ActionLastErrorId`** and **`ActionLastErrorDesc`**.

5.15.10. Create Label Variant

> **PRODUCT LEVEL INFO**
> Automation Builder features require **LMS Enterprise**.

This action allows you to create a review-ready variant of an existing label. Label objects in such variants have locked data source values. Their content is defined by the current value of the applicable data source.

The purpose of creating a review-ready variant of a label with “locked” data sources is to make the label suitable for approval process in which data and template need to be approved together. Instead of viewing a label without defined content for its objects, the approver reviews a variant with defined values. This allows him to quickly see and approve the final label layout with actual values that are going to be used for printing.

> **TIP**
> Label approval process is applicable to labels that are stored in Control Center Document Storage. You can apply various approval workflow types for the stored labels and label variants. Approval workflow selection depends on the requirements of your business environment. See NiceLabel 10 Control Center User Guide for more details.

About group identifies the selected action.

- **Name**: allows you to define a custom action name. This makes actions easily recognizable on the solution's list of actions. By default, the action name is taken from its type.
- **Description**: custom information about the action. Enter a description to explain the purpose and role of action in a solution.
- **Action type**: read-only information about the selected action type.

Settings group defines the label file to be converted and the output file (label variant).

- **Label name:** the name of the label file to be converted into a review-ready variant with locked data source values. **Data source** dynamically defines the **Label name** using an existing or newly created variable.
- **Print time data sources:** this option allows you to define the data sources whose values are going to be provided at the actual print time. If a data source is listed in this field, its value is not locked and can be provided at print time. Typical examples are data sources for production values like LOT number, expiry date, etc.

  TIP

  Insert only data source names without square brackets, separated by commas or listed in a column using Enter key.

- **Output file name:** the name of the label variant file that is going to be ready for review. **Data source** dynamically defines the **Label name** using an existing or newly created variable.

There are several rules that apply to the review-ready label variant:

1. Data source values are locked by default. To exclude the data sources from being locked, list them in **Print time data sources** field to keep them active on the review-ready label. You can define their values at print time.
2. Counter variables, functions, database fields and global variables are converted to non-prompted variables.
3. Graphics are embedded.
4. The destination label variant stored in NiceLabel Control Center Document Storage is automatically checked in. Original **Label name** and **Print time data sources** are used as check-in comment.
5. Label variants can be opened in NiceLabel 10 Designer in locked state.
6. Label files generated with this action cannot be imported.
7. If label variants are stored in printer memory, the recall command can only provide values for print time data sources.
8. If using NiceLabel Control Center, label preview in Document Storage allows editing of print time data sources.

9. Current time and current date variables cannot be set as Print time data sources on the review-ready label variant.

Action Execution and Error Handling

Each action can be set as a conditional action. Conditional actions only run when the defined conditions allow them to be run. To define these conditions, click **Show execution and error handling options**.

Execution options are:

- **Enabled:** specifies if the action is enabled or disabled. Only enabled actions will execute. This functionality may be used while testing a form.
- **Condition:** defines one-line programming expression that must provide a Boolean value (`true` or `false`). When the result of the expression is `true`, the action will execute. Condition offers a way to avoid executing actions every time.

Error handling options are:

- **Ignore failure**: specifies whether an error should be ignored or not. With **Ignore failure** option enabled, the execution of actions continues even if the current action fails.

  NOTE

  Nested actions that depend on the current action do not execute in case of a failure. The execution of actions continues with the next action on the same level as the current action. The error is logged but does not break the execution of the action.

Example

At the end of printing, you might want to send the status update to an external application using **HTTP Request** action. If the printing action fails, action processing stops. In order to execute the reporting even after the failed print action, the **Print Label** action must have the option **Ignore failure** enabled.

- **Save error to variable**: allows you to select or create a variable to save the error to. The same cause of the error is also saved to internal variables `ActionLastErrorId` and `ActionLastErrorDesc`.

6. Running and Managing Triggers

6.1. Deploying Configuration

After you have configured and tested the triggers in Automation Builder, deploy the configuration using NiceLabel Automation service and start the triggers. At that time, the triggers become active and start monitoring the defined events.

To deploy the configuration, use any of the following methods.

Deploying from Automation Builder

1. Start Automation Builder.
2. Load the configuration.
3. Go to **Configuration Items** tab.
4. Click the **Deploy Configuration** button in the Deploy ribbon group.
   The configuration loads inside Automation Manager running on the same machine.
5. Start the triggers you want to activate.

If this configuration has already been loaded, the deployment forces its reload, while keeping the triggers status active.

Deploying from Automation Manager

1. Start Automation Manager.
2. Go to **Triggers** tab.
3. Click **+Add** button and browse for configuration on the disk.
4. Start the triggers you want to make active.

Deploying using command-line

To deploy the configuration **`C:\Project\Configuration.MISX`** and to run the included trigger named as **`CSVTrigger`** using command- line, enter the following:

```
NiceLabelAutomationManager.exe ADD c:\Project\Configuration.MISX
NiceLabelAutomationManager.exe START c:\Project\Configuration.MISX CSVTrigger
```

For more information, see section Controlling the Service with Command-line Parameters.

6.2. Event Logging Options

PRODUCT LEVEL INFO

Some of the described functionalities in this section require purchase of **NiceLabel LMS** products.

NiceLabel Automation logs events on various locations, depending on the deployment scenario. The first two logging features are available with every NiceLabel Automation product level.

- **Logging to log database:** Logging to internal log database is always enabled. Internal log database keeps record of all events with all details. When viewing the logged information, you can use filters to display events that match the rules. For more information, see section Using Event Log.
  The data is stored in an SQLite database. This is a temporary log repository – the events are removed from the database on a weekly basis. The housekeeping interval is configurable in Options. Records of old events are deleted from the database, but the database does not get compacted (vacuumed), so it might still occupy the disk space. To compact the database, use a 3rd party SQLite management software.
- **Logging to Windows Application Event Log:** Important events are saved to Windows Application Event Log in case NiceLabel Automation fails to start. This ensures a secondary resource for logged events.
- **Logging to Control Center:** Logging to Control Center is available in **LMS Enterprise** and **LMS Pro** products. Control Center is a Web-based management console that records all events on one or more NiceLabel Automation servers. The data is stored in Microsoft SQL Server database. You can search through the collected data, and additionally, the application also supports automated alerts in case of certain events, printer management, document storage, revision control system (versioning), workflows, and label reprinting.

  NOTE

  For more information, see the Control Center user guide.

6.3. Managing Triggers

Automation Manager is the management part of NiceLabel Automation software. When using Automation Builder for configuring the triggers, you are using Automation Manager to deploy and run the triggers in production environment. The application allows you to load triggers from different configurations, see their live status, start/stop them, and see execution details in log file.

You can customize the view on the loaded configurations and their triggers. The last view is remembered and applied when you run Automation Manager for the next time. If you enable view **By status**, triggers from all open configurations that share the same status are displayed together. If you enable view by **Configurations**, triggers from the selected configuration are displayed together, no matter what their status is. Trigger status is color-coded in the trigger icon for easier identification.

The displayed trigger details change in real time – as soon as the trigger events are detected. You can see the information items, such as trigger name, type of trigger, how many events have already been processed, how many errors were detected, and the time that passed since the last event. If you hover your mouse above the number of already processed triggers, you see the number of trigger events waiting to be processed.

NOTE

The loaded configuration is cached in memory. If you make a change to the configuration in Automation Builder, the Automation Manager does not automatically apply it. To apply the change, reload the configuration.

Loading configuration

To load a configuration, click the **+Add** button and browse for the configuration file (.MISX). Triggers from the configuration load in suspended state. You have to start triggers to make them active. For more information, see section Deploying Configuration.

The list of loaded configurations and statuses for each trigger is remembered. If the server is restarted for whatever reason, NiceLabel Automation Service restores the trigger state from before the restart.

Configuration reloading and removal

After you update and save the configuration in Automation Builder, the changes are not automatically applied in Automation Manager. To reload the configuration, right-click the configuration name, and click **Reload Configuration**. This reloads all triggers. If you have file caching enabled, the reload forces synchronization off all files that are used by the triggers.

Starting / stopping triggers

If you load triggers from a configuration, their default state is stopped. To start the trigger, click the **Start** button in the trigger area. To stop the trigger, click **Stop**. You can select more triggers from the same configuration and start / stop all of them simultaneously.

You can also control starting/stopping of a configuration using command-line. For more information, see section Controlling the Service with Command-line Parameters.

Handling trigger conflicts

Triggers can be in error state due to below listed situations. You cannot start triggers in error until you resolve the issue.

- **Trigger not configured correctly or completely:** In this case, the trigger is not configured, mandatory properties are not defined, or actions defined for this printer are not configured. You cannot start such trigger.
- **Trigger configuration overlaps with another trigger:** Two triggers cannot monitor the same event.

Example

Two file triggers cannot monitor the same file. Two HTTP triggers cannot accept data on the same port. If the trigger configuration overlaps with another trigger, the second trigger cannot run because the event is already captured by the first trigger. For more information, see Log pane for that trigger.

Resetting the error status

When the trigger execution causes an error, the trigger icon color changes to red, trigger receives error status, and event details are logged to the logging database. Even if all of the upcoming events complete successfully, the trigger remains in error state until you confirm that you understand the error and that you want to clear the status. To acknowledge the error, click the icon next to the error counter under trigger details.

Using notification pane

Notification pane is the area above the list of triggers in the Triggers tab where important messages are displayed. Notification area displays application **status messages**, such as "Trial mode" or "Trial mode expired", or **warning messages**, such as "Tracing has been enabled".

Viewing Logged data

Every trigger activity is logged in the database, including trigger start/stop events, successful execution of action and errors encountered during processing. Click the Log button to see logged events just for the selected trigger. For more information, see section Using Event Log.

6.4. Using Event Log

All activities in NiceLabel Automation are logged to a database to enable history and troubleshooting. When you click the **Log** button in the Triggers tab, events for that particular trigger are displayed. The log pane displays information for all events that are related to the defined filter.

Logging data is useful for troubleshooting. If a trigger or action cannot be executed, the application records an error description in the log file that helps you identify and resolve the issue.

NOTE

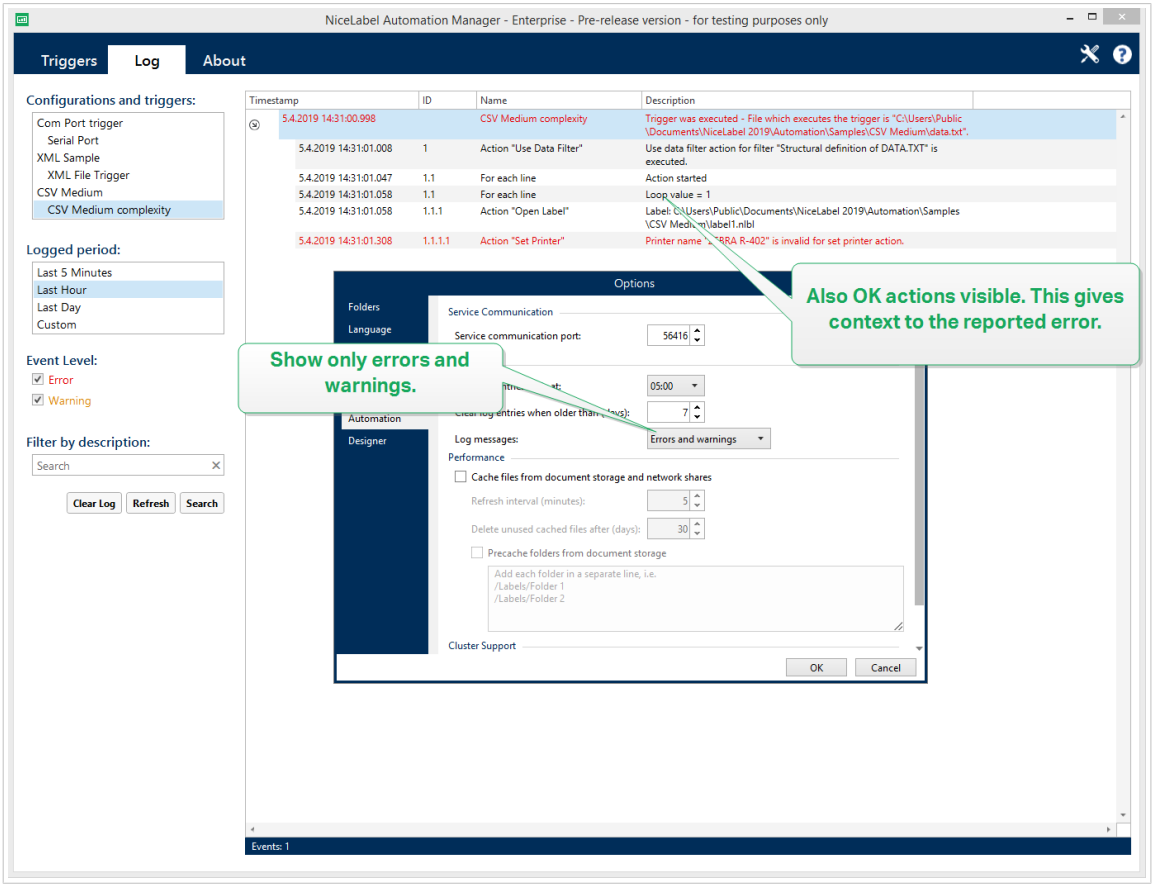
The default data retention time is 7 days and is configurable in Options. To minimize log database size on busy systems you might want to reduce the retention period.

Filtering events

The configurable filters:

- **Configuration and triggers:** Specifies which events to display – events from the selected trigger, or events from all triggers in the selected configuration.
- **Logged period:** Specifies the time frame in which the events occurred. Default time frame is **Last 5 minutes**.
- **Event level:** Specifies the type (importance) of the events you want to display:
  - **Error** is the type of event that breaks the execution.
  - **Warning** is the type of event during which errors happen, but are configured to be ignored.
  - **Information** is the type of event that logs all non-erroneous information.

> **NOTE**
>
> In case of errors and warnings, Automation also displays the entire sequence of successfully executed actions in a trigger.



Log level is configurable in **Options**.

- **Filter by description:** You can display all events that contain the provided string. Use this option to troubleshoot busy triggers. The filter is applied to the trigger description field.

Clearing the log database

You can clear the log from Automation Builder. To clear the log database, click the **Clear Log** button.

> **WARNING**
>
> Use log clearing with caution, because it is an irreversible command. Clear log removes **ALL** logged events from the database, and is applied to all triggers, not just to the current trigger.

Automated log database cleanups

Automation allows you to set up regular automated cleanups for log entries about the successfully executed triggers. This is how you make sure the growing log database does reduce the system performance.

To schedule the automated log database cleanups:

1. Open file **`product.config`** in text editor.
   The file is located here:

   ```
   %PROGRAMDATA%\NiceLabel\NiceLabel 10\product.config
   ```

2. Create a backup copy of the **`product.config`** file.
3. Automation uses two parameters to clean up the triggers. Add these two parameters to your **`product.config`** file.
   - **`/IntegrationService/LogSuccessfulTriggerPurgeInterval>`**. This parameter defines the length of the time interval between two consecutive cleanups. Type the interval length in minutes.
   - **`/IntegrationService/LogSuccessfulTriggerPurgeRemovalAge>`**. This parameter checks the age of the messages about the successfully executed actions.

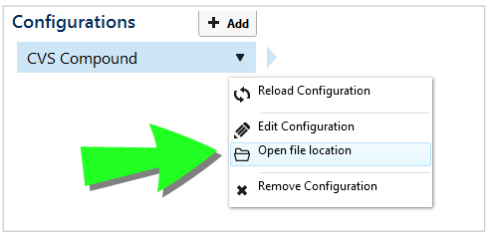
```
<configuration>
    <IntegrationService>
        <LogSuccessfulTriggerPurgeInterval>1</
LogSuccessfulTriggerPurgeInterval>

<LogSuccessfulTriggerPurgeRemovalAge>1<LogSuccessfulTriggerPurgeRemovalAg
e>
    </IntegrationService>
</configuration>
```

6.5. If your configuration fails to load...

When deployed, your Automation configuration runs as a Windows process in the background. The Automation Manager using which you manage and monitor your configuration is merely an interface that represents the actual Automation services. In certain cases, the configuration that you developed, tested, and deployed fails to load. There are multiple possible reasons. Follow the proposed solutions to get your Automation configuration up and running:

1. Configuration file was removed, renamed or moved to another location. You can check to which file your deployed configuration points in Automation Manager:

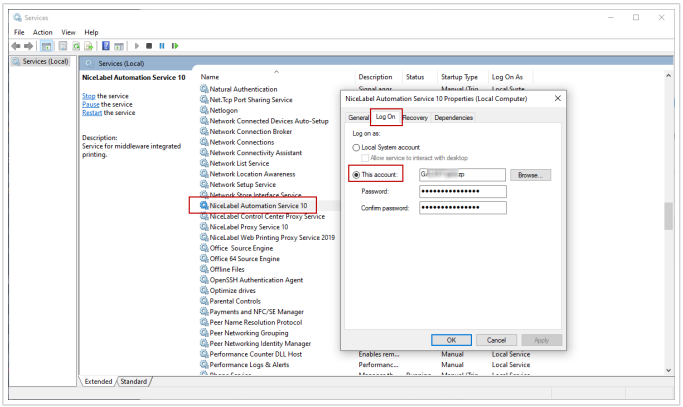


   Make sure the .misx configuration file is available at the specified location, and the it still has the same name as in Automation Manager. If you moved or renamed your configuration file, open it in Automation Builder and deploy the configuration again.

2. Configuration file is on a network location which cannot be accessed due to network connection problem. Check the network connectivity of your computer/server that stores the configuration.

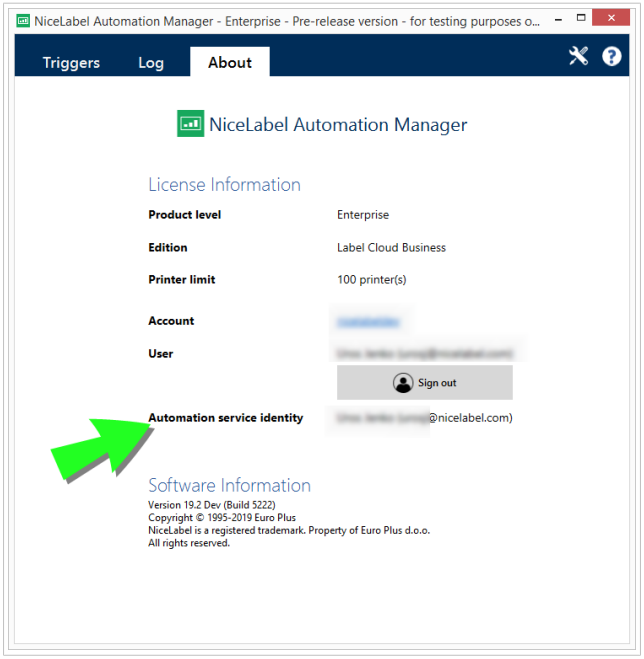
3. **Automation service does not have permission to access the configuration file.** Check permissions for user account used by Automation service. This error indicates issues with the Automation service running in the background.
   Possible solutions are:

   - If your configuration file is stored on a local disk or network share, check that you are running your configuration under your local user or domain credentials. Running your configuration under Local System Account may restrict access to shared network folders and printers. Open Services, and see the properties of the NiceLabelAutomation Service 10.

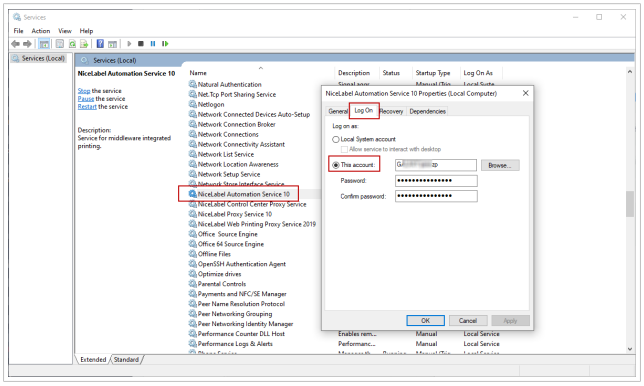


   - If your configuration file is stored in the document storage of your Control Center, these are the possible scenarios:

- Your Control Center uses the application authentication. The reason why your configuration fails to load, is incorrect **Automation service identity**. The Automation service identity must match the user identity as defined in your Control Center.
  You can find your configuration's Automation service identity in **Automation Manager** > **About tab**.



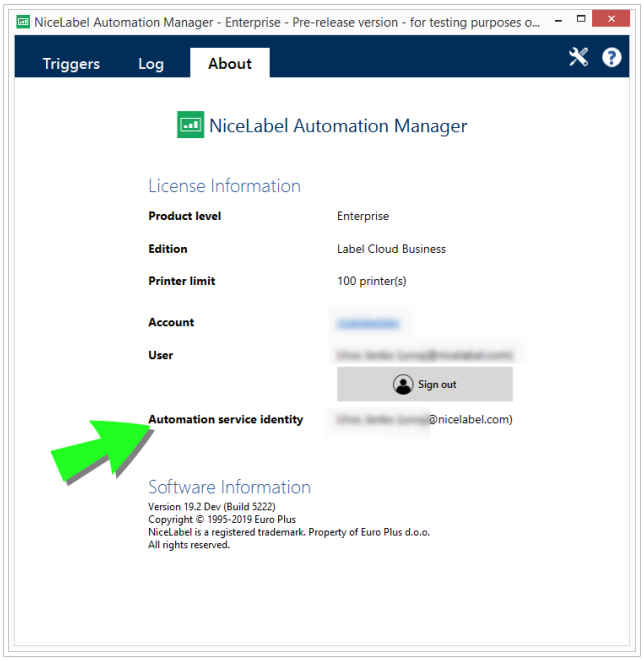
- Your Control Center uses the Windows authentication. Your configuration fails to load because you try to run it as a user with insufficient privileges on your Control Center. Check under which user you are running the Automation service. Open **Services**, and see the properties of the **NiceLabelAutomation Service 2019**.



> **NOTE**
> In both cases, see NiceLabelControl Center user guide for more details on the available authentication methods and user privileges.

- Your Control Center runs in NiceLabel Cloud. The reason why your configuration fails to load, is incorrect Automation service identity. The Automation service identity must match the NiceLabel Cloud user sign in.

You can find your configuration's **Automation service identity** in **Automation Manager** > **About tab**.



7. Performance and Feedback Options

7.1. Parallel Processing

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

NiceLabel Automation supports parallel processing for both – inbound and outbound processing. This ensures maximum efficiency on any system with installed software. NiceLabel Automation executes multiple tasks simultaneously, while still preserving the order in which the triggers have been added. Throughput of label job processing depends greatly on the hardware in use.

Inbound Parallel Processing

You can run many triggers on the same machine. They all simultaneously respond to changes in the monitored events. Each trigger remembers the data of unprocessed events on the queue list. This list buffers incoming data in case none of the print processes is available at the moment. As soon as one of the print processes becomes available, the first job is taken from queue using FIFO (First In, First Out) principle. This ensures the correct order of inbound data processing. However, it does not ensure the FIFO principle for printing. See the next section below.

NOTE

Parallel processing means more than just running multiple triggers at once. Each trigger can also allow concurrent connections. TCP/IP, HTTP, and Web Service triggers all accept concurrent connections from many clients. Also, file trigger can be configured to monitor a set of files in a folder. This is configurable by file mask.

Outbound Parallel Processing

Usually, the result of a trigger is label printing process. For this process, you are using data received by the trigger to print it on labels. NiceLabel Automation service runs the printing processes (aka "print engines") in parallel in the background. Modern processors have two or more independent central processing units called cores. Multiple cores can run multiple instructions at the same time, increasing the overall speed of processing. In case of NiceLabel Automation, multiple cores increase print job processing, and ultimately label printing performance.

By default, each NiceLabel Automation instance runs each printing process as a separate thread on every available core. The more powerful CPU you have, the more throughput is available. This maximizes the usage of the available CPU power. The software installs with reasonable defaults defining that every available core accommodates a single thread for print processing. Under normal circumstances, there is no need to make any changes on the defaults. If the configuration setup requires a change, see section Changing Multi-threaded Printing Defaults.

If there are multiple printing processes available, the data from the first event can be printed by one printing process, while the data from the second event could be printed by a different printing process simultaneously, if the second printing process is available at that time. If the second event did not provide a significant amount data, the printing process might provide the data for the printer faster than the first printing process, breaking the order. In such case, data from the second event could print before the data from the first event. To ensure FIFO principle also for printing, see section Synchronous Print Mode.

7.2. Caching Files

To improve the time-to-first label and performance in general, NiceLabel Automation supports file caching. When loading the labels, images and database data from network shares, you might experience delays while printing the labels. NiceLabel Automation must fetch all required files before the printing process can begin.

There are two levels of caching that complement each other.

- **Memory cache:** Memory cache stores the already used files. The labels that have been used at least once are loaded in the memory cache. When a trigger requests a printout of the same label, the label is immediately available for printing process. Memory cache is enabled by default. Its contents are cleared after you remove or reload a configuration. The label file is checked for changes for each Open Label action. If there is a newer label available, it loads automatically, replacing the old version in the cache.

  NOTE

  After a label is not in use for 8 hours, it is offloaded from the memory cache.

- **Persistent cache:** Persistent cache stores data to disk – its role is to provide medium term storage for files. Caching is managed per file object. After a file is being requested from the network share, the service first verifies if the file is already present in cache, and uses it. If a file is not in the cache,

it is fetched from network share and cached for future use. The cache service continuously updates cache contents with new versions of files. You can configure the time intervals for version checking in Options menu.

Prolonging the Time Period for Label Offloading

After a label is used for the first time, it is loaded in the memory cache. The label becomes available for instant printing the next time it is required. Memory cache housekeeping process removes all labels that haven't been in use for 8 hours or more.

To prolong the time interval in which labels are offloaded from memory cache, do the following:

1. Navigate to the NiceLabel Automation System folder.
   `%PROGRAMDATA%\NiceLabel\NiceLabel 10`
2. Make a backup copy of the file **`product.config`**.
3. Open **`product.config`** in a text editor. The file has XML structure.
4. Add the element **`Common/FileUpdater/PurgeAge`**.
5. This parameter defines the number of seconds that define the time frame for keeping labels in memory cache. NiceLabel Automation keeps track of the time when each label was used for printing for the last time. When that time frame reaches the defined threshold, the labels are unloaded from memory.

   NOTE

   Default value: 28800 (8 hours). Maximum value is 2147483647.

   The **`product.config`** file should have the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <Common>
        <FileUpdater>
            <PurgeAge>28800</PurgeAge>
        </FileUpdater>
    </Common>
...
</configuration>
```

6. After you save the file, NiceLabel Automation Service automatically applies the setting.

Enabling Persistent Cache

> **PRODUCT LEVEL INFO**
> The functionality from this section is available in LMS Enterprise.

To enable and configure persistent cache, open the Option, select NiceLabel Automation and enable **Cache remote files**.

- **Refresh cache files:** Defines the time interval in minutes in which the files in the cache will be synchronized with the files in the original folder. This is the time interval that you allow the system to use the old version of the file.
- **Remove cache files when older than:** Defines the time interval in days for removing all files in cache that haven't been accessed for the specified time duration.

NiceLabel Automation uses the following local folder to cache remote files:

```
%PROGRAMDATA%\NiceLabel\NiceLabel 10\FileCache
```

> **NOTE**
> File caching supports label and picture file formats. After you enable file caching, restart Automation service to make the changes take effect.

Forcing Reload of the Cache Content

NiceLabel Automation automatically refreshes cache content upon the defined time interval. Default value is 5 minutes.

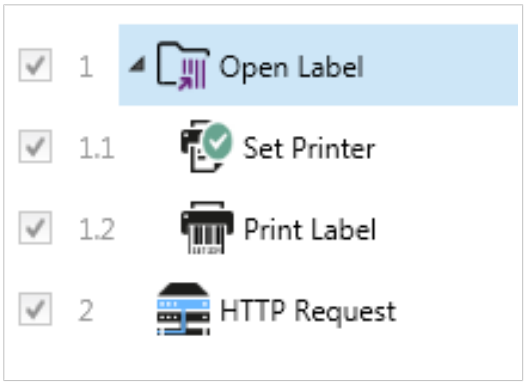
To manually force cache reloading, do the following:

1. Open Automation Manager.
2. Locate the configuration that contains the trigger for which you want to force-reload labels.
3. Right-click the configuration.
4. Select **Reload Configuration**.

7.3. Error Handling

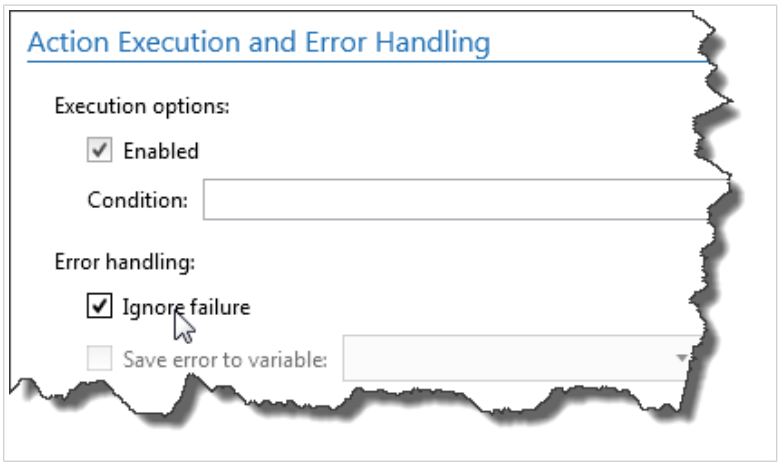
If an error occurs while executing an action, NiceLabel Automation stops executing all actions in the trigger. If there are actions defined after the current action that reports an error, these actions are not executed.

For example, actions are defined as shown in the screenshot below. If **Set Printer** action fails due to invalid name or inaccessible printer, the actions **Print Label** and **HTTP Request** are not executed. Action processing stops at **Set Printer**, Automation Manager shows the trigger in error state and the trigger status feedback (if enabled) reports "wrong printer specified / printer not accessible".



However, in this particular case you don't want to use synchronous feedback which is sent automatically if enabled in the trigger that supports synchronous feedback. Status feedback must be provided asynchronously using the **HTTP Request** action after the print job is created (or not). After the print process completes, update an application with its status. To achieve this, send the application an HTTP formatted message.

In this case, the **HTTP Request** action must be executed regardless of success of all the actions in the list above it. Enable the **Ignore failure** option for all actions that are placed above the **HTTP Request** action. The option is available under **Execution and error handling** options of an action.



If a particular action fails, NiceLabel Automation starts executing the next action on the higher level of hierarchy.

Example

If the **Set Printer** action on level 1.1 fails, the execution does not continue with **Print Label** action on level 1.2 because it will likely fail as well. It continues with the **HTTP Request** action on level 2, because it is the next action in the higher-level hierarchy.

The same logic can be implemented for looping actions, such as **Use Data Filter**, **Loop** and **For Each Record**. With these actions, you iterate through all members on the list. If processing of one member fails from whatever reason, by default NiceLabel Automation stops processing all other members and reports an error. If you enable **Ignore failure** option, the processing of the failed member stops, but NiceLabel Automation continues with the next member. At the end, the error is reported anyway.

7.4. Synchronous Print Mode

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

7.4.1. Asynchronous Print Mode

The default operation mode of NiceLabel Automation is asynchronous mode. Asynchronous mode is a form of printing, during which a trigger sends data for printing and closes connection with the print subsystem. The trigger does not wait for the result of the printing process and does not receive any feedback. Immediately after the data is sent, the trigger is ready to accept a new incoming data stream.

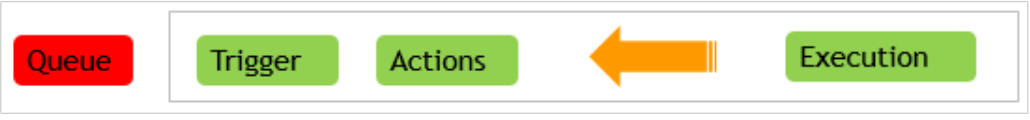
Asynchronous mode boosts trigger performance and increases the number of triggers that can be processed in a time frame. Each printing process has buffer in front – this is where the trigger feeds the print requests in. The buffer accommodates for trigger spikes and makes sure no data is lost.

If an error occurs during processing, it still gets logged in Automation Manager (and NiceLabel Control Center, if you use it), but the trigger by itself is not aware of it. When running Automation in asynchronous print mode, you cannot define conditional actions that would execute, if the trigger execution is in error.

7.4.2. Synchronous Print Mode

In comparison to asynchronous mode, synchronous mode doesn't break connection when the printing process begins. In this mode, the trigger sends data for printing and keeps the connection to the print subsystem established for as long as it is busy executing actions. When the printing process is complete (successfully or with an error), the trigger receives feedback about the status.

You can use this information inside the actions that are defined in the same trigger and decide to execute another action in case an error occurs. You can also send print job status back to the data-issuing application. For more information, see section Print Job Status Feedback.



Example

You can report printing status to the ERP application that provided the data.

You are going to use the synchronous mode if you want to receive status feedback within the trigger, or if you want to ensure FIFO printing mode. In this case, the data that is received with trigger events is printed using the same order as during its reception.

NOTE

When a trigger runs in synchronous printing mode, it communicates with a single printing process only. Enabling synchronous printing mode ensures the FIFO method of manipulating events in the outbound direction (printing). By default, multi-core processing cannot ensure the printing order.

Enabling the Synchronous Print Mode

Print mode is definable per-trigger. To enable synchronous mode in a trigger, do the following:

1. Open the properties of the trigger.
2. Go to **Settings** tab.
3. Select **Other**.
4. In section **Feedback from the Print Engine**, enable option **Supervised printing**.

7.5. Print Job Status Feedback

The application that provides data for label printing into NiceLabel Automation might expect to receive information about print job statuses. Feedback can be as simple as "All OK" in case of a successful print job generation, or detailed error description in case of an issue. Due to performance reasons, NiceLabel Automation NiceLabel Automation disables feedback possibility by default. This ensures high-throughput printing as trigger doesn't care about the print process execution. The errors are logged to log database, but the trigger does not handle them.

You can also use this method to send feedback about other data the trigger collects. This can be status of network printers, number of jobs in the printer spooler, list of labels in a folder, list of variables in the specified label file, and many more.

NOTE

To enable feedback support from print engine, enable synchronous print mode. For more information, see section Synchronous Print Mode.

Enable print job status feedback using one of the two available methods.

Trigger provides feedback about print job status (Synchronous feedback)
Some triggers have built-in feedback possibility. If synchronous print mode is enabled, the trigger is internally aware of the job status. The client can send the data into trigger, keep the connection open and wait for the feedback. To use this feedback method, select and use a trigger type that supports providing feedback.

If an error happens with any of the actions, the internal variable **`ActionLastErrorDesc`** contains the detailed error message. You can send its value as-is or customize it.

For more information, see details about the respective trigger types.

- Web Service Trigger: This trigger type supports feedback by design. The WSDL (Web Service Description Language) document describes details about the Web Service interface and instructs how to enable feedback. You can use the default reply that returns error description in case the print action failed. Or, you can customize the response and send back content of any variable. The variable itself can contain any data, including label preview or label print job (binary data).
- HTTP Server Trigger: This trigger type supports feedback by design. NiceLabel Automation uses standard HTTP response codes to indicate print job status. You can customize the HTTP response and send back content of any variable. The variable itself can contain any data, including label preview or label print job (binary data).
- TCP/IP Server Trigger: This trigger supports feedback, but not automatically. To make it deliver feedback, configure the data-providing client not to break the connection once the data is sent. After the print process completes, the next action on the list (Send Data to TCP/IP Port) might have the setting Reply to sender enabled. Provide feedback over the established still-open connection.

Action provides feedback about print job status (Asynchronous feedback)

In case of triggers that do not natively support feedback or in case you want to send feedback messages during the trigger processing, define an action that sends feedback to a selected destination. In this case, the data-providing application can close the connection as soon as the trigger data is delivered.

Example

You used the TCP/IP trigger to capture the data. The client dropped connection immediately after the data was sent, so we cannot reply over the same connection. In such cases, you can use another channel to send feedback. You can configure any of the outbound-connectivity actions, such as Execute SQL Statement, Open Document/Program, HTTP Request, Send Data to TCP/IP Port, and others. You would place such action under the Print Label action.

If you want to send feedback only for specific status, such as "error occurred", you can use the following methods.

- **Using condition on action:** Print job status is exposed in two internal variables (**`ActionLastErrorID`** and **`ActionLastErrorDesc`**). The first one contains error ID or contains value **`0`** in case of no reported errors. The second one contains a detailed error message. Use values of these variables in conditions on actions that you want to execute in case of errors. For example, you would use the action **HTTP Request** after printing. The action would send feedback in if an error occurred. To enable such feedback, do the following:

  1. Open trigger properties.
  2. In ribbon group **Variable**, click the **Internal Variables** button and enable variable **`ActionLastErrorID`**.
  3. Go to Actions tab.
  4. Add the action **Send Data to HTTP**.
  5. Inside the action's properties expand **Show execution and error handling options**.
  6. For **Condition**, enter the following. The action with this condition executes only if an error occurs and **`ActionLastErrorID`** contains the error ID (any value greater than **`0`**). By default, the conditions runs using VBScript syntax.

     ```
     ActionLastErrorID > 0
     ```

  7. You also have to enable the **Ignore failure** option on each action you expect to fail. This instructs Automation not to stop executing actions entirely, but to continue with the next action on the same hierarchical level.

     NOTE

     For more information, see section Error Handling.

- **Using action Try:** Action Try eliminates the need for coding conditions. The action gives you two placeholders. Placeholder **Do** contains actions that you want to run. If any error occurs when running them, the execution breaks, and actions in the **On error** placeholder are executed. You would use outbound-connectivity actions in this placeholder to provide a print job status feedback. For more information, see section Try.

7.6. Excluding printers from automated printing

In certain cases, your printing environment requires you to exclude some of your printers from the automated printing process. Possible reasons why you should exclude printers from the automated printing are your company printing policy or the limitations of you license.

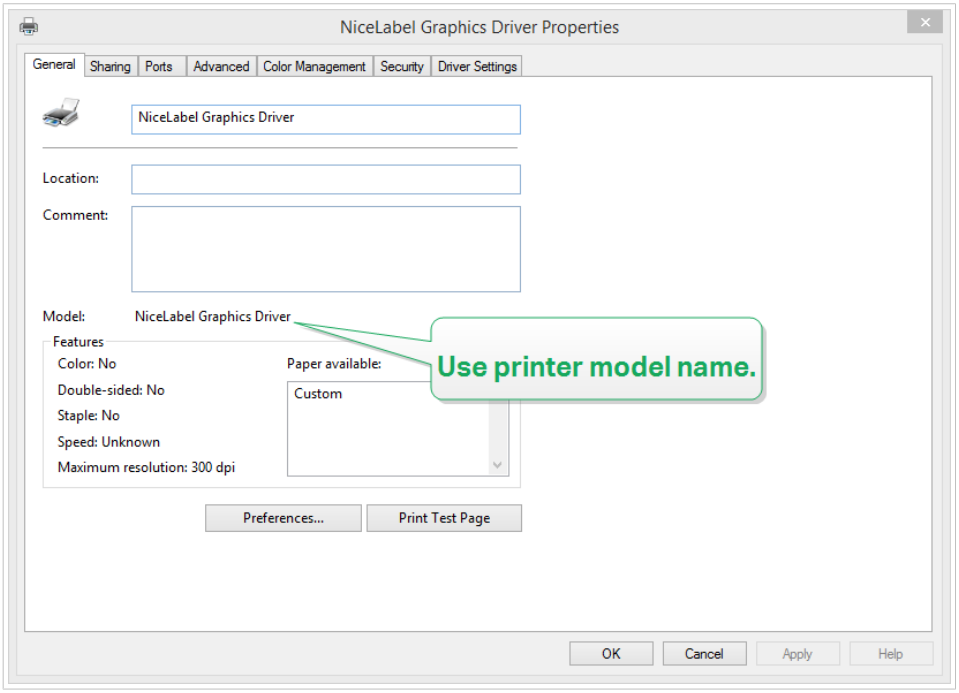
By default, Automation prevents automated printing using file printers, such as Microsoft Shared Fax Driver, Microsoft Print To PDF, Microsoft XPS Document Writer, and similar. These file printers require the users to manually select the location for their "printouts". The request for manual user intervention causes the print engine to stop, reporting an error.

To prevent the Automation from using specific printers in the running configurations:

NOTE

When specifying the excluded printers from the automated printing in the **`product.config`** file, you must also explicitly list your file printers.

1. Open file **`product.config`** in text editor.
   The file is located here:
   `%PROGRAMDATA%\NiceLabel\NiceLabel 10\product.config`
2. Create a backup copy of the **`product.config`** file.
3. Automation uses two parameters to check which printer model and printer port should be excluded from automated printing. Add these parameters to your **`product.config`** file.
   - **`/IntegrationService/DisabledPrinterDrivers`** and type the printer models you want to exclude from automated printing.
   - **`/IntegrationService/DisabledPrinterPorts`** and type the ports you want to exclude from automated printing.




```
<configuration>
    <IntegrationService>
        <DisabledPrinterDrivers>NiceLabel Graphics
Driver,NicePrinter1200dpi</DisabledPrinterDrivers>
        <DisabledPrinterPorts>LPT1,LPT2</DisabledPrinterPorts>
    </IntegrationService>
</configuration>
```

4. Once you define the printer models and printer ports in the **`product.config`** file, you can still run your Automation configurations, but your updated settings prevent printing on the listed printers. Automation reports an error if the these printers are part of the running configurations.

> **NOTE**
> Automated printing does not stop if the excluded printers are used in the Redirect Printing to File action.

7.7. Using Store/Recall Printing Mode

Store and Recall printing mode optimizes the printing process. It enhances printer response by reducing the amount of data that needs to be sent during repetitive printing tasks.

With store and recall mode activated, NiceLabel Automation does not need to resend the complete label data for each printout. Instead, the labels (templates) are stored in the printer memory. Fixed objects are stored as such, while placeholders are defined for the variable objects. NiceLabel Automation only sends data for variable label objects and the recall commands. The printer applies received data in the

placeholders on the stored label and prints the label by recalling it from the memory. Typically, a few bytes of data are sent to the printer, compared to a few kilobytes as would be the case during normal printing.

The action consists of two processes:

- **Store label:** During this process, the application creates a description of the label template formatted in the selected printer's command language. When done, the application sends the created command file to the printer memory and stores it. You can store a label from label designer or from NiceLabel Automation using the Store label to printer action.

  > **NOTE**
  > The label must have the store and recall printing mode defined in its properties before you can store it to printer.

- **Recall (print) label:** A label stored in the printer memory is printed out immediately. Using the recall process, NiceLabel Automation creates another command file to instruct the printer which label from its memory should be printed. The actual amount of data sent to the printer depends on the current situation. For fixed labels without any variable contents, the recall command file only contains the recall label command. For variable labels that contain variable fields, the command file includes the values for these variables and the recall label command.
  To recall a label from NiceLabel Automation just use one of the usual printing actions. When executed, the action analyzes the label and enables the appropriate printing mode: normal print or recall print, as defined in the label.

> **WARNING**
> Before activating this mode, make sure the appropriate printer driver is selected for the label printer. Not all label printers have the ability to use the store and recall printing mode. The printer driver for which the label was created in the label designer must also be installed on the machine where NiceLabel Automation is running.

7.8. High-availability (Failover) Cluster

> **PRODUCT LEVEL INFO**
> The functionality from this section is available in LMS Enterprise.

NiceLabel Automation supports Microsoft high-availability (fail-over) cluster. A fail-over cluster is a group of independent computers that work together to increase the availability of label printing through NiceLabel Automation. The clustered servers (called nodes) are connected by physical cables and by software. If one or more of the clustered nodes fail, other nodes begin to provide service (a process known as fail-over). 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. The clients providing data connect to the IP address that belongs to the entire cluster, and not to the individual node IP addresses.

To enable NiceLabel Automation to perform in a high-availability cluster, do the following:

- Set up Microsoft Failover Clustering feature on your Windows Servers.
- Install NiceLabel Automation on each node.
- Enable fail-over cluster support in NiceLabel Automation properties on each node.
  Do the following:
  1. Open **File** > **Options** > **Automation**.
  2. Under **Cluster Support** group, enable **Failover Cluster Support**.
  3. Browse for the folder, located outside of both nodes, but still accessible with full access privileges to NiceLabel Automation software. Important system files that both nodes need will be copied to this folder.
- Configure the cluster to start NiceLabel Automation on the second node in case the master node is down.

7.9. Load-balancing Cluster

PRODUCT LEVEL INFO

The functionality from this section is available in LMS Enterprise.

NiceLabel Automation supports Microsoft load-balancing cluster. Load-balancing cluster is a group of independent computers that work together to increase the availability and scalability of label printing through NiceLabel Automation. The clustered servers (called nodes) are connected by physical cables and by software. The incoming requests for label printing are distributed among all nodes in a cluster. The clients providing data connect to the IP address belonging to the cluster, and not to individual node IP addresses.

NOTE

You can use TCP/IP-based triggers with load-balancing cluster. These triggers are TCP/IP Server Trigger, HTTP Server Trigger, Web Service Trigger, and Cloud Trigger.

To enable NiceLabel Automation for load-balancing, do the following:

- Set up Microsoft Load-balancing Clustering feature in your Windows Servers.
- Install NiceLabel Automation on each node.
- Load the same configuration files in Automation Manager on each node.

8. Understanding Data Structures

This section demonstrates data structures that are frequently used in automation scenarios. When working with various data files, we have to analyze their structure, extract the relevant values from fields of interest, and print them on labels. Each of the below listed case is used in sample configurations that are included in the Automation installation package.

- Text Database
- Compound CSV
- Binary Files
- Legacy Data
- Command Files
- XML Data
- JSON Data

8.1. Binary Files

Binary files contain plain text and binary characters, such as control codes (characters below ASCII code 32). The Unstructured Data Filter supports binary characters. You can use binary characters to define fields positions, and you can also use binary characters for field values.

Typical example would be data export from a legacy system, in which the data for each label is delimited with a Form Feed character **<FF>**.

8.1.1. Example

In this case, Automation trigger captures the print stream. The yellow-highlighted data section must be extracted from the stream and sent to a different printer. The filter is configured to search for **<FF>** as field-end position.

```
<ESC>%-12345X@PJL USTATUSOFF
@PJL INFO STATUS
@PJL USTATUS DEVICE=ON
<ESC>%-12345X<ESC>%-12345X



^^02^L
^^02^O0270
D11
H15
PE
SE
Q0001
131100000300070001-001-001
1e42055007500500001001019
1322000001502859
W
E
<FF><ESC>%-12345X<ESC>%-12345X@PJL USTATUSOFF
<ESC>%-12345X
```

For more information, see section Examples.

8.2. Command Files

Command files are plain text files containing commands that are executed one at a time from top to bottom. NiceLabel Automation supports native command files, as well as Oracle and SAP XML command files. For more information see section Command Files Specifications, Oracle XML Specifications, and SAP AII XML Specifications.

8.2.1. Example

The label **`label2.nlbl`** is going to print to **`CAB A3 203DPI`** printer.

```
LABEL "label2.nlbl"
SET code="12345"
SET article="FUSILLI"
SET ean="383860026501"
SET weight="1,0 kg"
PRINTER "CAB A3 203DPI"
PRINT 1
```

For more information, see section Examples.

8.3. Compound CSV

Compound CSV is a text file that contains data in two structures – in a standard CSV structure, and in a multi-line header that uses a non-standard structure. The contents of a compound CSV file cannot be parsed using a single filter. To parse data in both structures, configure two separate filters:

- Structured Text Filter for fields in CSV structure
- Unstructured Data Filter for fields in the header non-standard structure

Configuration that includes a compound CSV file, requires two actions to execute both filters on the received data.

8.3.1. Example

Data items from line 3 until the end of the document have CSV structure and are parsed by the Structured Text filter. Data items in the first two lines do not have any particular structure and are parsed using Unstructured Data filter.

```
OPTPEPPQPF0 NL004002 ;F75-TEP77319022891-001-001
OPT2 zg2lbprt.p 34.1.7.7 GOLF+ label
print"printer";"label";"lbl_qty";"f_logo";"f_field_1";"f_field_2";"f_field_3"
"Production01";"label.nlbl";"1";"logo-nicelabel.png";"ABCS1161P";"Post: ";"1"
"Production01";"label.nlbl";"1";"logo-nicelabel.png";"ABCS1162P";"Post: ";"2"
"Production01";"label.nlbl";"1";"logo-nicelabel.png";"ABCS1163P";"Post: ";"3"
"Production01";"label.nlbl";"1";"logo-nicelabel.png";"ABCS1164P";"Post: ";"4"
"Production01";"label.nlbl";"1";"logo-nicelabel.png";"ABCS1165P";"Post: ";"5"
```

For more information, see section Examples.

8.4. Legacy Data

Legacy data are unstructured or semi-structured exports from legacy applications. These exports do not use CSV or XML data structures. To extract relevant data from such files, use the Unstructured Data Filter and define the positions of fields of interest. The filter in Automation extracts field values and makes them available for printing on labels.

8.4.1. Example

The file below follows no structure related rules. Each field must be configured manually.

```
HAWLEY      ANNIE      ER12345678 ABC      XYZ
                 9876543210
PRE OP      07/11/12      F 27/06/47      St. Ken Hospital      3

G015 134 557 564 9      A- 08/11/12      LDBS F-          PB      1
G015 134 654 234 0      A- 08/11/12      LDBS F-          PB      2
G015 134 324 563 C      A- 08/11/12      LDBS F-          PB      3



                 Antibody Screen: Negative
    Store Sample :
    SAMPLE VALID FOR 24 HOURS, NO TRANSFUSION HISTORY SUPPLIED

07/11/12      B,31.0001245.E      O Rh(D) Pos          PHO
                          RLUH      BT
```

For more information, see section Examples.

8.5. Text Database

Text database is an alias for text files with structured fields, such as CSV (comma separated file), or text files with fixed-width fields. In either case, you can click the **Import Data Structure** button and follow the wizard to import the fields. If you have a data file with delimited structure and the number of fields varies from one copy to another, enable the **Dynamic structure** feature and let NiceLabel Automation handle the data extraction and mapping to variables automatically. For more information, see section Enabling Dynamic Structure.

8.5.1. Example

- **File with delimited fields:** The first line in the file contains field names that the filter can import.

```
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
```

- **File with fixed-width fields:** Fields contain a fixed number of characters.

```
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
```

For more information, see section Examples.

8.6. XML Data

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

XML stands for eXtensible Markup Language. XML tags are not predefined, you are free to define your own tags to describe the data. XML is designed to be self-descriptive.

XML structure is defined by elements, attributes (and their values), and text (element text).

8.6.1. Examples

Oracle XML

Processing of Oracle XML is an integral part of the software. You don't have to configure any filters to extract the data, just run the built-in action Run Oracle XML Command File. For more information on the XML structure, see section Oracle XML Specifications.

```
<?xml version="1.0" standalone="no"?>
<labels _FORMAT="case.nlbl" _PRINTERNAME="Production01" _QUANTITY="1">
    <label>
        <variable name="CASEID">0000000123</variable>
        <variable name="CARTONTYPE"/>
        <variable name="ORDERKEY">0000000534</variable>
        <variable name="BUYERPO"/>
        <variable name="ROUTE"></variable>
        <variable name="CONTAINERDETAILID">0000004212</variable>
        <variable name="SERIALREFERENCE">0</variable>
        <variable name="FILTERVALUE">0</variable>
        <variable name="INDICATORDIGIT">0</variable>
        <variable name="DATE">11/19/2012 10:59:03</variable>
    </label>
</labels>
```

General XML

If the XML structure in not natively supported in the software, define the XML filter, and define rules to extract the data. For more information, see section Understanding Filters.

```
<?xml version="1.0" encoding="utf-8"?>
<asx:abap xmlns:asx="http://www.sap.com/abapxml" version="1.0">
    <asx:values>
        <NICELABEL_JOB>
            <TIMESTAMP>20130221100527.788134</TIMESTAMP>
            <USER>PGRI</USER>
            <IT_LABEL_DATA>
                <LBL_NAME>goods_receipt.nlbl</LBL_NAME>
                <LBL_PRINTER>Production01</LBL_PRINTER>
                <LBL_QUANTITY>1</LBL_QUANTITY>
                <MAKTX>MASS ONE</MAKTX>
                <MATNR>28345</MATNR>
                <MEINS>KG</MEINS>
                <WDATU>19.01.2012</WDATU>
                <QUANTITY>1</QUANTITY>
                <EXIDV>012345678901234560</EXIDV>
            </IT_LABEL_DATA>
        </NICELABEL_JOB>
    </asx:values>
</asx:abap>
```

NiceLabel XML

Processing of NiceLabelXML is an integral part of the software. You don't have to configure any filters to extract the data, just run the built-in action Run Command File. For more information on the XML structure, see section XML Command File.

```
<nice_commands>
    <label name="label1.nlbl">

        <session_print_job printer="CAB A3 203DPI" skip=0 job_name="job name
1" print_to_file="filename 1">
            <session quantity="10">
                <variable name="variable name 1" >variable value 1</variable>
            </session>
        </session_print_job>

        <print_job printer="Zebra R-402” quantity="10" skip=0
identical_copies=1 number_of_sets=1 job_name="job name 2"
print_to_file="filename 2">
            <variable name="variable1" >1</variable>
            <variable name="variable2" >2</variable>
            <variable name="variable3" >3</variable>
        </print_job>
    </label>
</nice_commands>
```

For more hands-on information on how to work with XML data, see section Examples.

8.7. JSON Data

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

JavaScript Object Notation (JSON) is an open-standard file format. JSON uses human-readable text to transmit data objects consisting of name–value pairs, and array data types (or any other serializable value). JSON is a very common data format used for asynchronous browser–server communication, including as a replacement for XML.

There are multiple online resources describing the similarities and differences between JSON and XML. The table below describes a portion of them:

JSON	XML
It is JavaScript Object Notation	It is Extensible markup language
It is based on JavaScript language.	It is derived from SGML.
It is a way of representing objects.	It is a markup language and uses tag structure to represent data items.
It does not provides any support for namespaces.	It supports namespaces.
It supports array.	It doesn't supports array.
Its files are very easy to read as compared to XML.	Its documents are comparatively difficult to read and interpret.
It doesn't use end tag.	It has start and end tags.
It is less secure.	It is more secure than JSON.
It doesn't supports comments.	It supports comments.
It supports only UTF-8 encoding.	It supports various encoding.

Source: https://www.geeksforgeeks.org/difference-between-json-and-xml/

Examples

```
{
  "DELIVERYNOTE": {
    "LIST_CUSTOMER_INFO": {
      "CUSTOMER_INFO": {
        "CUSTOMER_NAME": "Customer A",
        "CUSTOMER_STREET_ADDRESS": "Test St",
        "CUSTOMER_POST_ADDRESS": "1234, Test City",
        "CUSTOMER_NUMBER": "1234",
        "CURRENCY": "EUR",
        "DELIVERY_METHOD": "Express delivery",
        "EDI_INFORMATION": "EDI",
        "ORDER_TYPE": "CSO",
        "ORDER_NUMBER": "123",
        "LIST_ITEM": {
          "ITEM": [
            {
              "ARTICLE_NUMBER": "0001",
              "ARTICLE_NAME": "Collins Complete Woodworker's Manual",
              "PRICE": "23.3"
            },
            {
              "ARTICLE_NUMBER": "0002",
              "ARTICLE_NAME": "Be Careful What You Wish For (Clifton
Chronicles)",
              "PRICE": "16.6"
            },
            {
              "ARTICLE_NUMBER": "0003",
              "ARTICLE_NAME": "Mockingjay (part III of Hunger Games Trilogy)",
              "PRICE": "25.0"
            }
          ]
        }
      }
    }
  }
}
```

```
{
    "NICELABEL_JOB": {
      "TIMESTAMP": "20130221100527.788134",
      "USER": "PGRI",
       IT_LABEL_DATA": {
        "LBL_NAME": "goods_receipt.nlbl",
        "LBL_PRINTER": "Production01",
```

```
        "LBL_QUANTITY": "1",
        "MAKTX": "MASS ONE",
        "MATNR": "28345",
        "MEINS": "KG",
        "WDATU": "19.01.2012",
        "QUANTITY": "1",
        "EXIDV": "012345678901234560"
      }
    }
  }
}
```

9. Reference and Troubleshooting

9.1. Command File Types

9.1.1. Command Files Specifications

Command files contain instructions for the print process. These instructions are expressed with NiceLabel commands. Commands are executed one at a time from beginning to the end of the file. The files support Unicode formatting, so you can include multi-lingual contents. Command files come in three different flavors.

9.1.2. CSV Command File

The commands available in CSV command files are a subset of NiceLabel commands. The following commands are available: **LABEL**, **SET**, **PORT**, **PRINTER** and **PRINT**.

CSV stands for Comma Separated Values. CSV file is a text file in which values are delimited by comma (**,**) character. Such text file can contain Unicode value (important for multi-language data). Each line in the CSV command file contains commands for a single label print action.

The first row in a CSV command file must contain commands and variable names. The order of commands and names is not important, but all records in the same data stream must follow the same structure. Variable **`name-value`** pairs are extracted automatically and sent to the referenced label. If the variable with the name from CSV does not exist on the label, no error message is displayed.

9.1.2.1. Sample CSV Command File

The sample presents the structural view on the fields that you can use in the CSV command file.

```
@Label,@Printer,@Quantity,@Skip,@IdenticalCopies,NumberOfSets,@Port,Product_ID,
 Product_Name
label1.nlbl, CAB A3 203 DPI, 100, , , , , 100A, Product 1
label2.nlbl, Zebra R-402, 20, , , , , 200A, Product 2
```

CSV Commands Specification

The commands in the first line of data must be expressed with at (@) character. The fields without @ at the beginning are names of variables. These fields are extracted with their values as **`name-value`** pairs.

- **@Label:** Specifies the label name to be used. It's a good practice to include label path and filename. Make sure the service user can access the file. For more information, see section Access to Network Shared Resources in NiceLabel Automation user guide. A required field.
- **@Printer:** Specifies the printer to be used. It overrides the printer defined in the label. Make sure the service user can access the printer. For more information, see section Access to Network Shared Resources. Optional field.
- **@Quantity:** Specifies the number of labels to be printed. Possible values: numeric value, VARIABLE or UNLIMITED. For more information, see section Print Label. A required field.
- **@Skip:** Specifies the number of labels to be skipped at the beginning of the first printed page. This feature is useful if you want to re-use the partially printed sheet of labels. Optional field.
- **@IdenticalCopies:** Specifies the number of label copies that should be printed for each unique label. This feature is useful when printing labels with data from database or when you use counters, and you need label copies. Optional field.
- **@NumberOfSets:** Specifies the number of times the printing process should repeat. Each label set equals a single occurrence of printing process. Optional field.
- **@Port:** Specifies the port name for the printer. You can override the default port as specified in the printer driver. You can also use it to redirect printing to file. Optional field.
- **Other field names:** All other fields define names of variables from the label. The field contents are saved to the variable of the same name as its value.

9.1.3. JOB Command File

JOB command file is a text file that contains NiceLabelcommands. The commands execute in top-to-bottom order. The commands usually start with LABEL (to open label), then SET (to set variable value) and finally PRINT (to print label). For more information about the available commands, see section Using Custom Commands.

9.1.3.1. Sample JOB Command File

This JOB file open **`label2.nlbl`**, sets variable values and prints a single label. Because no PRINTER command is used to redirect printing, the gets printed using the printer name as defined in the label.

```
LABEL "label2.nlbl"
SET code="12345"
SET article="FUSILLI"
SET ean="383860026501"
SET weight="1,0 kg"
PRINT 1
```

9.1.4. XML Command File

The commands available in the XML Command files are a subset of NiceLabel commands. The following commands are available: **LOGIN**, **LABEL**, **SET**, **PORT**, **PRINTER**, **SESSIONEND**, **SESSIONSTART** and **SESSIONPRINT**. The syntax for these commands differs to some extent if they are used in an XML file.

Root element in the XML Command file is **<Nice_Commands>**. The next element that must follow is **<Label>**. This element specifies the label to be used.

To start printing the labels , there are two available methods: print labels normally using the element **<Print_Job>**, or print labels in a session using the element **<Session_Print_Job>**. You can also change the printer to which the labels are printed, and additionally set the variable value.

9.1.4.1. Sample XML Command File

The sample below serves as a structural overview of the elements and their attributes as you can use them in an XML command file.

```
<nice_commands>
    <label name="label1.nlbl">

        <session_print_job printer="CAB A3 203DPI" skip=0 job_name="job name
1" print_to_file="filename 1">
            <session quantity="10">
                <variable name="variable name 1" >variable value 1</variable>
            </session>
        </session_print_job>

        <print_job printer="Zebra R-402" quantity="10" skip=0
identical_copies=1 number_of_sets=1 job_name="job name 2"
print_to_file="filename 2">
            <variable name="variable1" >1</variable>
            <variable name="variable2" >2</variable>
            <variable name="variable3" >3</variable>
        </print_job>
    </label>
</nice_commands>
```

XML Command File Specification

This section contains description of the XML Command file structure. There are several elements that contain attributes. Some attributes are required while the others are optional. Some attributes can occupy pre-defined values only, while for the others, you can specify custom values.

- **<Nice_Commands>:** This is a root element.

- **<Label>:** Specifies the label file to be opened. If the label is already opened, it does not re-open. The label file must be accessible from this computer. For more information, see section Access to Network Shared Resources. This element can occur several times within the command file.
    - **Name:** This attribute contains label name. You can include the path along with the label name. Required.
- **<Print_Job>:** The element that contains data for a single label job. This element can occur several times within the command file.
    - **Printer:**: Use this attribute to override the printer defined in the label. The printer must be accessible from this computer. For more information, see section Access to Network Shared Resources. Optional.
    - **Quantity:** Use this attribute to specify the number of labels to be printed. Possible values: numeric value, VARIABLE or UNLIMITED. For more information on parameters, see section Print Label. Required.
    - **Skip:** Use this attribute to specify how many labels should be skipped at the beginning. This feature is useful if you print a sheet of labels to laser printer, but the sheet is partial already printed. For more information, see the see section Print Label. Optional.
    - **Job_name:** Use this attribute to specify the name of your job file. The specified name is visible in the print spooler. For more information, see section Set print job name. Optional.
    - **Print_to_file:** Use this attribute to specify the file name to which you want to save the printer commands. For more information, see section Redirect Printing to File. Optional.
    - **Identical_copies:** Use this attribute to specify the number of copies you need for each label. For more information, see section Print Label. Op
- **<Session_Print_Job>:** The element that contains commands and data for one or more sessions. The element can contain one or more **<Session>** elements. It considers session print rules. You can use this element several times within the command file. For available attributes, look up the attributes for the **<Print_Job>** element. All of them are valid, you just cannot use the quantity attribute. See the description of **<Session>** element to find out how to specify label quantity in session printing.
- **<Session>:** The element that contains data for one session. When printing in session, all labels are encoded into a single print job and sent to the printer.
    - **Quantity:** Use this attribute to specify the number of printed labels. Possible values: numeric value, string VARIABLE, or string UNLIMITED. For more information on parameters, see section Print Label. Required.
- **<Variable>:** The element that sets the variable values on the label. This element can occur several times within the command file.
    - **Name:** The attribute contains the variable name. Required.

XML Schema Definition (XSD) for XML Command File

```
<?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="optional" />
                    <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>
```

9.1.5. Oracle XML Specifications

Oracle defines the XML format so that the XML contents can be understood, parsed and printed on labels. The XML Document Type Definition (DTD) defines the XML tags that are used in an XML file. Oracle generates XML files according to this DTD, and any 3rd party software translates XML files according to this DTD.

To execute an Oracle XML command file, use the Run Oracle XML Command File action.

9.1.5.1. XML DTD

The following is the XML DTD that is used while forming an XML for both – synchronous and asynchronous XML formats. The DTD defines elements that are used in the XML file, list of their attributes, and next level elements.

```
<!ELEMENT labels (label)*>
<!ATTLIST labels _FORMAT CDATA #IMPLIED>
<!ATTLIST labels _JOBNAME CDATA #IMPLIED>
<!ATTLIST labels _QUANTITY 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>
```

9.1.5.2. Sample Oracle XML

This is the Oracle XML providing data for one label (there is only one **<label>** element).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE labels SYSTEM "label.dtd">
<labels _FORMAT ="Serial.nlbl" _QUANTITY="1" _PRINTERNAME="" _JOBNAME="Serial">
    <label>
        <variable name= "item">O Ring</variable>
        <variable name= "revision">V1</variable>
        <variable name= "lot">123</variable>
        <variable name= "serial_number">12345</variable>
        <variable name= "lot_status">123</variable>
        <variable name= "serial_number_status">Active</variable>
        <variable name= "organization">A1</variable>
    </label>
</labels>
```

When executing this sample Oracle XML file, the label **`serial.nlbl`** is printed with the following variable values.

Variable name	Variable value
item	O Ring
revision	V1
lot	123
serial_number	12345
lot_status	123
serial_number_status	Active
organization	A1

There will be **`1`** printed copy of the label with spooler job name **`Serial`**. The printer name is not specified in the XML file, so the label prints to the printer as defined in the label template.

9.1.6. SAP AII XML Specifications

NiceLabel Automation can present itself as an RFID device controller, capable of encoding RFID tags and printing labels. For more information about SAP AII XML specifications, see the document **SAP Auto-ID Infrastructure Device Controller Interface** on SAP web page.

To execute such command file, use the Run SAP AII XML Command File action.

9.1.6.1. Sample SAP AII XML

This is the SAP AII XML providing data for one label (note that there is only one **`<label>`** element).

```
<?xml version="1.0" encoding="UTF-8"?>
<Command xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Command.xsd">
    <WriteTagData readerID="DEVICE ID">
        <Item>
            <FieldList format="c:\SAP Demo\SAP label.nlbl"
jobName="Writer_Device20040929165746" quantity="1">
            <Field name="EPC">00037000657330</Field>
            <Field name="EPC_TYPE">SGTIN-96</Field>
            <Field name="EPC_URN">urn:autoid:tag:sgtin:3.5.0037000.065774.8</
Field>
            <Field name="PRODUCT">Product</Field>
            <Field name="PRODUCT_DESCRIPTION">Product description</Field>
            </FieldList>
        </Item>
    </WriteTagData>
</Command>
```

When executing this sample SAP AI XML file, the label **`c:\SAP Demo\SAP label.nlbl`** is printed with the following variable values.

Variable name	Variable value
EPC	00037000657330
EPC_TYPE	SGTIN-96
EPC	urn:autoid:tag:sgtin:3.5.0037000.065774.8
PRODUCT	Product
PRODUCT_DESCRIPTION	Product description

There will be **`1`** printed copy of the label with spooler job name **`Writer_Device2004092916574`**. Printer name is not specified in the XML file, so the label will be printed using the printer that is defined in the label template.

9.2. Custom Commands

9.2.1. Using Custom Commands

NiceLabel commands are used in command files to control label printing. NiceLabel Automation executes the commands within command files in top-to-bottom order. For more details, see section Command Files Specifications.

You can use specific custom commands if they are available in your NiceLabel Automation product in form of an action.

Example

It is possible to the use the **SETPRINTPARAM** command if you can select the **Set Print Parameter** action (available with product levels Pro and Enterprise).

COMMENT

```
;
```

When developing a command file, it is good practice to document your commands. This helps you decode what the script really does when you look at the code after some time. Use semicolon (`;`) at the beginning of the line. Everything following the semicolon is treated as comment and does not get processed.

CLEARVARIABLEVALUES

```
CLEARVARIABLEVALUES
```

This command resets variable values to their default values.

CREATEFILE

```
CREATEFILE <file name> [, <contents>]
```

This command creates a text file. Use the text file to give signal to a third party application that the print process has begun or ended, depending on the location where you put the command. Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources.

DELETEFILE

```
DELETEFILE <file name>
```

Deletes the specified file. Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources.

EXPORTLABEL

```
EXPORTLABEL ExportFileName [, ExportVariant]
```

This command automates the "Export to printer" command that is available in label designer. The label is exported directly to the printer and stored in printer memory for off-line printing. The user recalls the label using printer keyboard or by sending a command file to the printer. The same functionality is available also with Store label to printer action.

NOTE

To specify the label for exporting, use the **LABEL** command first.

- **ExportFileName:** This mandatory parameter defines the file name of generated printer commands.
- **ExportVariant:** Some printers support multiple export variants. When exporting manually, the user can select the export variant in the dialog. With the EXPORTLABEL command, you have to specify which export variant you want to use. The variants are visible in the label designer after you enable the Store/Recall printing mode.
  The first variant in the list has value 0. The second variant has value 1, etc.
  If you do not specify any variant type, value 0 is used as default.

For more information about off-line printing, see section Using Store/Recall Printing Mode.

IGNOREERROR

```
IGNOREERROR <on> [,<off>]
```

This command specifies that the error occurring in the JOB file does not terminate the printing process if the following errors occur:

- Incorrect variable name is used.
- Incorrect value is sent to the variable.
- Label does not exist / is not accessible.
- Printer does not exist / is not accessible.

LABEL

```
LABEL <label name> [,<printer_name>]
```

This command opens the label to be printed. If the label is already loaded, it does not get re-opened. You can include the path name. Enclose the label name in double quotes if the name or path contains spaces. Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources.

The optional **`printer_name`** specifies the printer, for which the label is opened. Use this setting if you want to override the printer name that is saved in the label template. If the driver for the provided printer name is not installed or not available, the command raises an error.

MESSAGEBOX

```
MESSAGEBOX <message> [,<caption>]
```

This command logs the custom **`message`** into the trigger log. If the message contains space characters or commas, enclose the text in double quotes (").

PORT

```
PORT <file name> [, APPEND]
```

This command overrides port as defined in the printer driver and redirects printing to a file. If file path or file name contain spaces, enclose the value in double quotes ("). Use UNC syntax for network resources. For more information, see section Access to Network Shared Resources.

The **APPEND** parameter is optional. By default, the file gets overwritten. Use this parameter to append data into the existing file.

Once you use the PORT command in JOB file, it remains valid until the next PORT command, or until the end of file (whichever comes first). If you use PRINTER command after the PORT command has been executed, the PORT setting overwrites the port defined for the selected printer. If you want to use the actual port that is defined for the selected printer, you have to use another PORT command with empty value, such as **`PORT = ""`**.

PRINT

```
PRINT <quantity> [,<skip> [,<identical label copies> [,number of label sets]]]
```

This command starts the printing process.

- **Quantity:** Specifies the number of printed labels.
  - **<number>:** The specified number of labels gets printed.
  - **VARIABLE:** Specifies that a label variable is defined as *variable quantity* and that it contains the number of labels to be printed. The label determines how many labels get printed.
  - **UNLIMITED:** If you use a database to acquire values for objects, unlimited printing prints as many labels as there are records in the database. If you do not use a database, the maximum number of labels that thermal printer internally supports gets printed.
- **Skip:** Specifies the number of labels you want to skip on the first page. The parameter is used for printing labels on sheets of paper. If the part of the page has already been used, reuse the same sheet by shifting the starting location of the first label.
- **I dentical label copies:** Specifies how many copies of the same label must be printed.
- **Number of label sets.** Specifies the number of times the whole printing process should repeat itself.

NOTE

Make sure the quantity values are provided as numeric value, not as a string value. Do not enclose the value in double quotes.

PRINTER

```
PRINTER <printer name>
```

This command overrides the printer as defined in the label file. If the printer name contains space characters, enclose it in double quotes (**"**).

Use the printer name as displayed in the status line of label design application. Printer names usually match the listed names under Printers and Faxes in Control Panel, but not always. If using network printers, you might see the name displayed in syntax **`\\server\share`**.

PRINTJOBNAME

```
PRINTJOBNAME
```

This command specifies the print job name as displayed in Windows Spooler. If the name contains space characters or commas, enclose the value in double quotes (**"**).

SESSIONEND

```
SESSIONEND
```

This command closes print stream. Also see **SESSIONSTART**.

NOTE

`SESSIONEND` must be sent as the only item in Send Custom Command action. If you would like to send additional commands, use separate Send Custom Command actions.

SESSIONPRINT

```
SESSIONPRINT <quantity> [,<skip>]
```

This command prints the currently referenced label and adds it into the currently open session-print stream. You can use multiple SESSIONPRINT commands one after another and join the referenced labels in a single print stream. The stream does not close until you close it using SESSIONEND command. The meaning of quantity and skip parameters is the same as with NiceCommand PRINT. Also see **SESSIONSTART** command.

- **Quantity:** Specifies the number of labels to be printed.
- **Skip:** Specifies the number of labels you want to skip on the first page. The parameter is used for printing labels on sheets of paper. If the part of the page has already been used, you can reuse the same sheet by shifting the start location of the first label.

SESSIONSTART

```
SESSIONSTART
```

This command initiates the session-print type of printing.

The three session-print-related commands (**SESSIONSTART**, **SESSIONPRINT**, **SESSIONEND**) are used together. When you use command PRINT, data for each label is sent to the printer in a separate print job. If you want to join label data for multiple labels into a print stream, use the session print commands. To do so, start with the SESSIONSTART command, followed by any number of SESSIONPRINT commands. The sequence ends with SESSIONEND command.

Use these commands to optimize label printing process. Printing of labels that belong to a single print job is much faster than printing labels using multiple print jobs.

Use the rules below to make sure the session print does not break:

- You cannot change the label within a session.
- You cannot change the printer within a session.
- You must set values for all variables from the label within a session, even if some of the variables have empty values.

SET

```
SET <name>=<value> [,<step> [,<number or repetitions>]]
```

This command assigns the variable **`name`** with **`value`**. The variable must be defined on the label, or an error is raised. If the variable isn't on the label, an error occurs. **`Step`** and **`number of repetitions`** are parameters for counter variables. These parameters specify the counter increment value and the number printed labels before the counter changes value.

If a value contains spaces or comma characters, enclose the text in double quotes (**"**). Also see **TEXTQUALIFIER**.

If you want to assign a multi-line value, use **`\r\n`** to encode a newline character. **`\r`** is replaced with CR (Carriage Return) and **`\n`** is replaced with LF (Line Feed).

Be careful when setting values to variables that provide data for pictures on the label, as backslash characters might be replaced with other characters.

Example

If you assign a value "c:\My Pictures\raw.jpg" to the variable, the "\r" is replaced with CR character.

SETPRINTPARAM

```
SETPRINTPARAM <paramname> = <value>
```

This command allows you to fine-tune the printer settings just before printing. The supported parameter for printer settings (**`paramname`**) are:

- **PAPERBIN:** Specifies the tray that contains label media. If the printer is equipped with more than one paper / label tray, you can control which one is used for printing. The name of the tray should be acquired from the printer driver.
- **PRINTSPEED:** Specifies the printing speed. The acceptable values vary from one printer to the other. See printer's manuals for exact range of values.
- **PRINTDARKNESS:** Specifies the printing darkness / contrast. The acceptable values vary from one printer to another. See printer manuals for exact range of values.
- **PRINTOFFSETX:** Specifies the left offset for all printing objects. The value for parameter must be numeric, positive or negative, in dots.
- **PRINTOFFSETY:** Specifies the top offset for all printing objects. The value for parameter must be numeric, positive or negative, in dots.
- **PRINTERSETTINGS:** Specifies the custom printer settings to be applied to the print job. The parameter requires the entire DEVMODE for the target printer, provided in a Base64-encoded string. The DEVMODE contains all parameters from the printer driver at once (speed, darkness, offsets and other). For more information, see section Understanding printer settings and DEVMODE.

  NOTE

  The Base64-encoded string must be provided inside double quotes (").

TEXTQUALIFIER

```
TEXTQUALIFIER <character>
```

Text-qualifier is the character that embeds data value that is assigned to a variable. Whenever a data value includes space characters, it must be included with text-qualifiers. The default text qualifier is a double quote character ("). Because double quote character is used as shortcut for inch unit of measure, it is sometimes difficult to pass the data with inch marks in the JOB files. You can use two double quotes to encode one double quote, or use TEXTQUALIFIER.

Example

```
TEXTQUALIFIER %

SET Variable = %EPAK 12"X10 7/32"%
```

9.3. Access to Network Shared Resources

This section defines best practice steps for using network shared resources.

- **User privileges for service mode.** The execution component of NiceLabel Automation runs in service mode under a specified user account inheriting access privileges of that account. For NiceLabel Automation to be able to open label files and user printer drivers, the associated user account must have granted the same privileges. For more information, see section Running in Service Mode.
- **UNC notation for network shares.** When accessing the file on a network drive, make sure you are using the UNC syntax (Universal Naming Convention), and not the mapped drive letters. UNC is a naming convention that specifies and maps network drives. NiceLabel Automation tries to replace the drive-letter syntax with UNC syntax automatically.

Example

If a file is accessible as **`G:\Labels\label.nlbl`**, refer to it in UNC notation as **`\\server\share\Labels\label.nlbl`** (where G: drive is mapped to **`\\server\share`**).

- **Notation for accessing files in Control Center.** If you open a file in Document Storage within Control Center, you can use HTTP notation such as **`http://servername:8080/label.nlbl`**, or WebDAV notation such as **`\\servername@8080\DavWWWRoot\label.nlbl`**.
  Additional notes:
  - User account under which the NiceLabel Automation service runs is also used to get files from Document Storage. This user must be configured in Control Center Configuration. This makes sure the user has access to the Document Storage files.
  - WebDAV access can only be used with Windows user authentication type in Control Center.

  NOTE
  The Document Storage is available with products **LMS Enterprise** and **LMS Pro**.
- **Printer drivers availability.** To print labels to network shared printer, you will have to make the printer driver available on the server where NiceLabel Automation is installed on. Make sure the user account that NiceLabel Automation Service runs under has access to the printer driver. If the network printer was just installed on the machine, NiceLabel Automation might not see it until your restart the Service. To allow automatic notification of new network printer drivers, you have to enable the appropriate inbound rule in the Windows firewall. For more information, see the Knowledge Base article.

9.4. Document Storage and Versioning of Configuration Files

Document Storage is a functionality of NiceLabel Control Center. It enables the NiceLabel Control Center to perform as a shared file repository on the server, in which users can store their files, retrieve them, and control their revisions.

Document Storage contextual tab enables you to perform document storage actions straight from Automation Builder. This makes accessing and opening the Automation file in NiceLabel Control Center unnecessary.

> **NOTE**
> This contextual tab requires connection with NiceLabel Control Center. LMS Enterprise license is mandatory for such configurations.

Revisioning group allows you to perform the available document storage actions:

- **Check Out**: Checks out the file from NiceLabel Control Center document storage and makes it available for editing. The checked-out file is marked and locked for editing for any other user. All other users will see the current revision of the file, while the author already works on a new draft.

> **NOTE**
> After opening a document from the document storage (**File** > **Open** > **Document Storage**), the editing commands remain disabled until you check out the document.

- **Check In**: Checks the file to NiceLabel Control Center document storage after the editing is done. When you check in the file, its revision number increments by one. The entered comment is added to file log.
- **Discard Checkout**: Discards checkout of the current file and gives other users full access to the file.

  > **WARNING**
  > If you click **Discard Checkout**, all changes since the last file checkout will be lost.

- **Document Storage**: Opens document storage location of the connected NiceLabel Control Center.

9.5. Accessing Databases

Whenever NiceLabel Automation needs to get data from a database, make sure that the necessary database driver is installed on the Windows system. Database drivers are provided by the company developing the database software. The driver that you install must match the bitness of your Windows system. NiceLabel software always runs in the bitness of your Windows system.

9.5.1. 32-bit Windows

If you have 32-bit Windows, you can only install 32-bit database drivers. The same database driver is be used to configure the trigger in Automation Builder and to execute trigger execution in NiceLabel Automation Service. All NiceLabel Automation components run as 32-bit applications.

9.5.2. 64-bit Windows

If you have 64-bit Windows, you can install 64-bit or 32-bit database drivers. The applications that are running in 64-bit use 64-bit database drivers. The applications that are running in 32-bit use 32-bit database drivers.

By default, Automation Service runs as 64-bit process. As such, it uses 64-bit database drivers to make a database connection. If 64-bit database drivers are not available on the system on which Automation Service is running, the database connection task offloads to the **NiceLabel Proxy** process. This process always runs as 32-bit process.

9.6. Automatic Font Replacement

You might design your label templates to print text objects using internal printer fonts. These are the fonts that are stored in your printer's memory. If you try to print such labels to a different kind or printer, the selected internal fonts might not be available. The new printer probably supports an entirely different set of internal fonts. The font layout might be similar in such case, but is available under a different name.

Font mismatch might also occur if the Truetype font that you are using on your labels is not installed on the computer on which you run Designer to design and print labels.

You can configure Designer to automatically replace the fonts used on the label with compatible fonts. I such case, Designer maps and replaces the fonts using their names. If the original font is not available, Designer uses the first available replacement font defined in the mapping table.

If there are no suitable replacement fonts, Designer uses the Arial Truetype font.

NOTE

After configuring font replacement, mapping rules execute when you change the printer for your label.

Configuring the Font Mapping

1. Open File Explorer and navigate to the following folder:

```
%PROGRAMDATA%\NiceLabel\NiceLabel 10
```

2. Copy **fontmapping.def** file to **fontmapping.local.def**.
3. Open the **fontmapping.local.def** file in your favorite text XML editor.
4. Inside the **FontMappings** element, create a new element with a custom name.
5. Inside the new element, create at least two elements named as **Mapping**.
    - Value of the first element named Mapping must contain the name of the original font.
    - Value of the second element named Mapping must contain the name of the replacement font.

NOTE

Additional Mapping elements with new font names are allowed. If the first replacement font is not available, Designer tries the next one. If no replacement fonts are available, Arial Truetype is used instead.

NOTE

The file **fontmapping.local.def** is your file and is preserved during the upgrades. On the other hand, **fontmapping.def** belongs to NiceLabel and is overwritten during the upgrades. Do not modify the **fontmapping.def** file.

Sample Mapping Configuration

In the example below, two font mappings are defined.

- The first mapping converts any **Avery** font into a matching **Novexx** font. For example, **Avery YT100** font gets replaced with **Novexx YT100**, **Avery 1** font gets replaced with **Novexx 1**. If the Novexx font is not available, **Arial** Truetype font is used instead.
- The second mapping converts **Avery YT100** font into **Novexx YT104**. If that font is not available, then **Zebra 0** font is used. If this font is also not available, **Arial** Truetype is used instead.
- The second mapping overrides the first one.

```
<?xml version="1.0" encoding="utf-8"?>
<FontMappings>
    <AveryNovexx>
        <Mapping>Avery</Mapping>
        <Mapping>Novexx</Mapping>
    </AveryNovexx>
    <TextReplacement>
        <Mapping>Avery YT100</Mapping>
        <Mapping>Novexx YT104</Mapping>
        <Mapping>Zebra 0</Mapping>
    </TextReplacement>
</FontMappings>
```

9.7. Automating Reports

PRODUCT LEVEL INFO

This functionality is available in **LMS Enterprise** and **LMS Pro**.

Combine your business system data with labels containing **Reports**. Create reports in NiceLabel Designer and fill your reports with data from your business systems in NiceLabel Automation.

Your completed reports use your business system data to print. NiceLabel Automation:

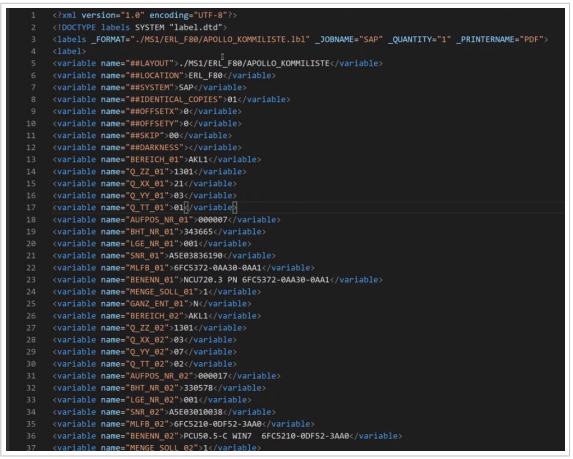
- **Receives** your business system **data**.
- **Parses your data** with a **data filter**.
- **Populates your Report** with parsed data.
- **Executes printing** for your new populated reports with a **trigger**.

9.7.1. Creating temporary databases

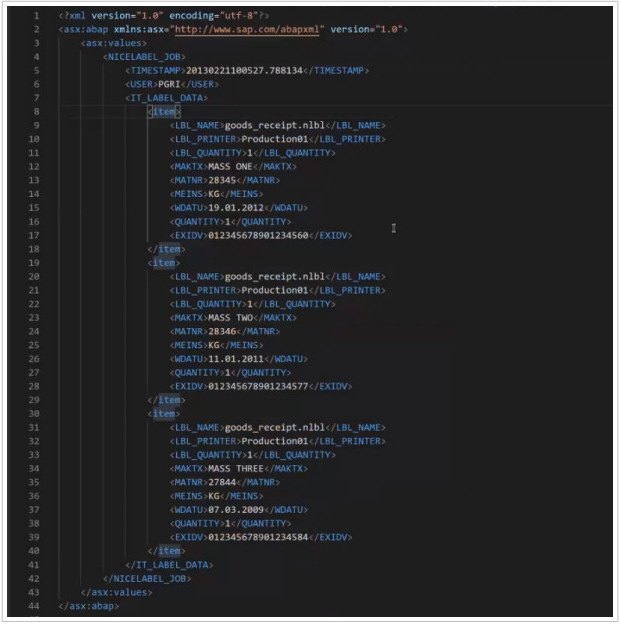
You need to design your **Reports** before you automate them. To design reports, you need to connect them to a **database**.

If you use data from external business systems for reports, you have **no database** to work with. To work correctly, report data needs a **hierarchical structure** with clearly defined elements.

Create a **temporary database** with data from your business system (Usually XML or JSON data). With your temporary database, design and configure your reports. **Your temporary database is only for setup**. Configured reports print using Automation triggers, not your temporary database.



Data from your business system with no hierarchy.



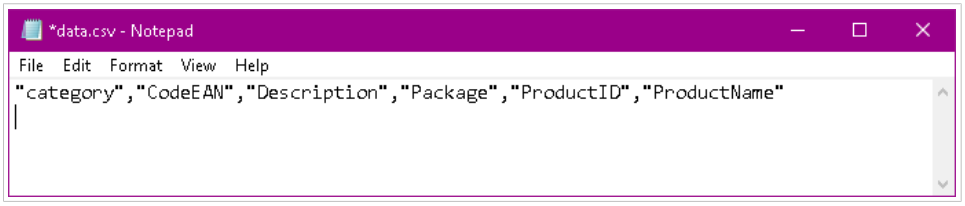
Data with hierarchy you can use to print reports.

```
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Pasta.xsd" generated="2020-08-31T15:43:51">
  <Total>211</Total>
  <Pasta>
    <ProductID>PAS501</ProductID>
    <CodeEAN>8021228310001</CodeEAN>
    <ProductName>BIGOLI 250G</ProductName>
    <Package>6</Package>
    <category>Long pasta</category>
    <Description>A long, thin, cylindrical pasta of Italian origin. Spaghetti is made of semolina or flour and
    water.</Description>
  </Pasta>
  <Pasta>
    <ProductID>PAS502GI</ProductID>
    <CodeEAN>8021228310018</CodeEAN>
    <ProductName>TAGLIATELLE 250G</ProductName>
    <Package>6</Package>
    <category>Ribbon-cut pasta</category>
    <Description>Ribbon style pasta are often rolled flat and then cut. This can be done by hand or
    mechanically.</Description>
  </Pasta>
```

Sample XML data you need to design reports (unparsed).

For you temporary database, create a CSV text database file. You have multiple options:

1. Manually convert your data structure to a CSV text file:



Manually creating temporary CSV text databases.

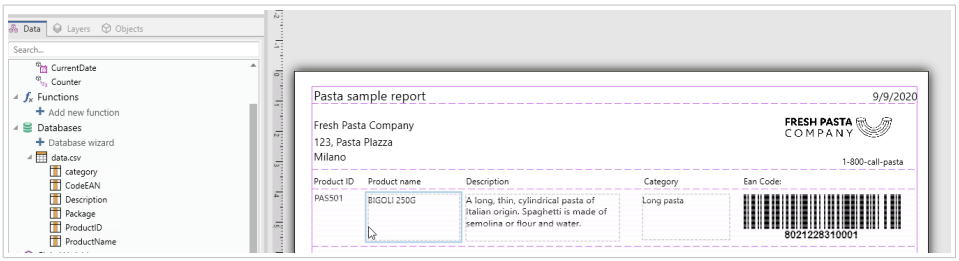
2. Use a common XML to Excel or XML to CSV conversion tool and manually format your data in a spreadsheet.
3. Use an Automation **data filter** to automatically parse your data into a CSV text database (Refer to your included **XML to CSV** sample file for more information).

9.7.2. Designing automated Reports

Open **Designer** to create your **Report** with your temporary CSV text database:

1. **Connect** your temporary CSV text database to your **Report**.
2. **Design** your report using your CSV text data as variables for objects in in your **Repeater Definition**.

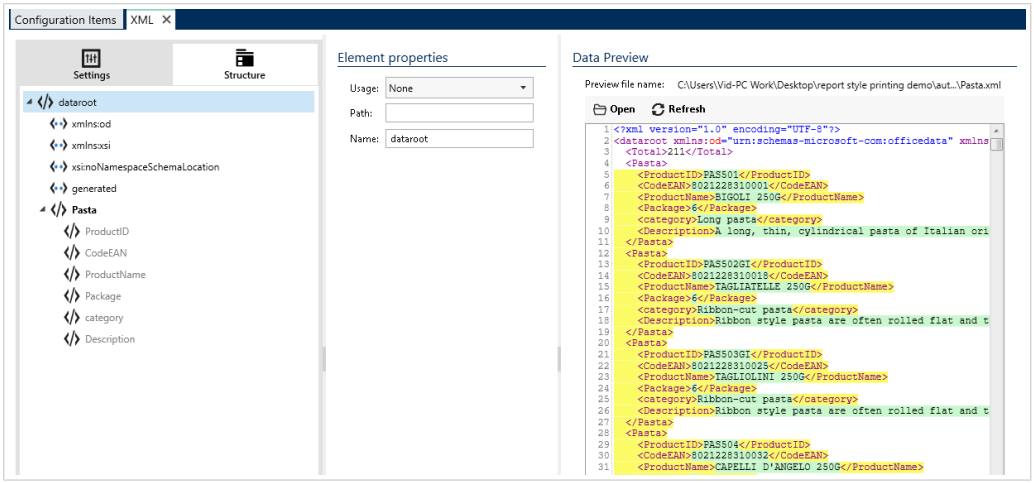
> **NOTE**
> **Match** your **variable names** with **data names** in your data filter to map your data correctly.



Designing reports with your temporary CSV text database.

9.7.3. Creating data filters

Create a **data filter** in Automation to use in your report (Refer to your included example trigger for more information on creating XML filters):



Configuring your XML data filter.

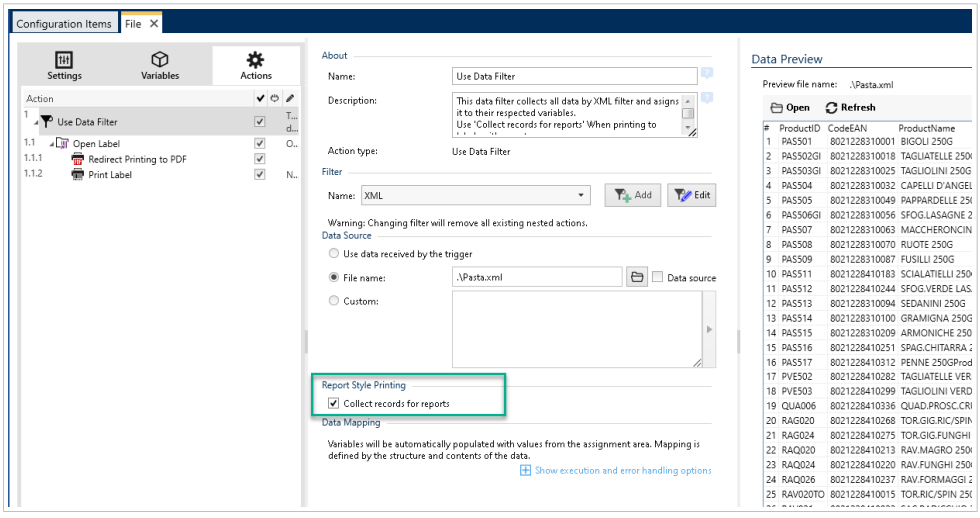
Tips for creating data filters for reports:

- You can use **Structure Import Wizard** to import your data structure.
- Name your elements in data blocks of repeatable elements.
- Set your elements as assignment areas or set sub-items as variables.

9.7.4. Creating triggers for your new data filter

> **NOTE**
>
> For your trigger to work, your data source names in Designer must match the data block field names in your data filter.

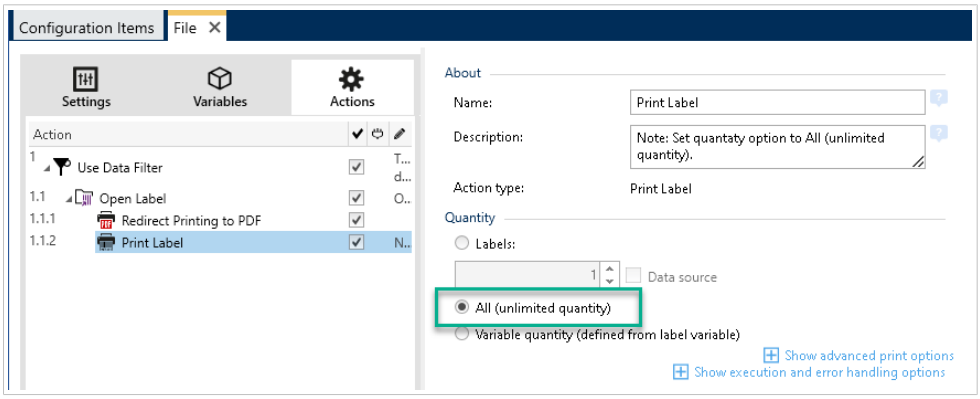
- Include a **Use Data Filter** action to apply your data filter.



- Enable **Collect records for report** to collect all records in a table Automation uses to open your report labels.

> **NOTE**
> Use the same data structure names in your reports, triggers, and filters.

- Include a **Print Label** action.



  - Enable **All (unlimited quantity)**.

Run your trigger. Your report now automatically updates with new data and prints with your trigger.

9.8. Changing Multi-threaded Printing Defaults

> **PRODUCT LEVEL INFO**
> This functionality is available in **LMS Enterprise** and **LMS Pro**.

Every NiceLabel Automation product can take advantage of multiple processor cores – each core runs a print process independently. Half of the number of cores is used for processing concurrent *normal* threads, and the remaining half for processing concurrent *session-print* threads.

> **NOTE**
> Under normal circumstances, you never have to change the default settings. Make sure you know what you are doing when changing these defaults.

To modify the number of the concurrent print threads, do the following:

1. Open file **`product.config`** in text editor.
   The file is located here:

```
%PROGRAMDATA%\NiceLabel\NiceLabel 10\product.config
```

2. Change the values for elements **MaxConcurrentPrintProcesses** and **MaxConcurrentSessionPrintProcesses**.

```
<configuration>
    <IntegrationService>
        <MaxConcurrentPrintProcesses>1</MaxConcurrentPrintProcesses>
        <MaxConcurrentSessionPrintProcesses>1</
MaxConcurrentSessionPrintProcesses>
    </IntegrationService>
</configuration>
```

3. Save the file. NiceLabel Automation automatically updates the service with the new number of print threads.

9.9. Compatibility with NiceWatch Products

NiceLabel Automation allows you to load trigger configurations that were built using legacy NiceWatch products. In the majority of cases, you can run NiceWatch configuration using NiceLabel Automation without any modifications.

NiceLabel Automation products are using a new .NET based print engine optimized for performance and low memory footprint. The new print engine does not support each label design option that is available in the label designer. Each new release of NiceLabel Automation is closing the gap, but you might still come across certain unavailable features.

Resolving Incompatibility Issues

NiceLabel Automation warns you if you try to print existing label templates that contain design features which are not supported with the new print engine.

If there are incompatibilities with NiceWatch configuration file or label templates, Automation notifies you about:

- **Compatibility with trigger configuration:** While opening the NiceWatch configuration (.MIS file), NiceLabel Automation checks it against the supported features. Not all features from NiceWatch are available in NiceLabel Automation. While some are not available at all, some of them are configured differently. If the MIS file contains unsupported features, you will see them listed. Automation removes these features from the configuration.
  If you come across such a case, open the .MIS file in Automation Builder and resolve the incompatibility issues. You will have to use the available NiceLabel Automation features to re-create the removed configuration items.
- **Compatibility with label templates:** If your existing label templates contain unsupported print engine features as provided by NiceLabel Automation, you will see error messages in the **Log** pane. This information is visible in the Automation Builder (when designing triggers) or in Automation Manager (when running the triggers).
  In this case, you have to open the label file in label designer and remove unsupported features from the label.

NOTE

For more information about incompatibility issues with NiceWatch and label designers, see NiceLabel Automation Compatibility with NiceLabel Designers V6 and NiceWatch V5.

Opening NiceWatch Configuration for Editing

Open the existing NiceWatch configuration (.MIS file) in Automation Builder and edit it in Automation Builder. You can save the configuration as a .MISX file only.

To edit the NiceWatch configuration, do the following:

1. Start Automation Builder.
2. Go to **File** > **Open NiceWatch File**.
3. In the **Open** dialog box, browse for the NiceWatch configuration file (.MIS file).
4. Click **OK**.
5. If the configuration contains unsupported features, they are displayed in a list. Automation removes them from the configuration.

Opening NiceWatch Configuration for Execution

You can open NiceWatch configuration (.MIS file) in Automation Manager without conversion to the NiceLabel Automation file format (.MISX file). If the triggers from NiceWatch are compatible with NiceLabel Automation, you can start using them right away.

To open and deploy NiceWatch configuration, do the following:

1. Start Automation Manager.
2. Click **+Add** button.
3. In **Open** dialog box, change the file type into **NiceWatch Configuration**.
4. Browse for the NiceWatch configuration file (.MIS file).
5. Click **OK**.
6. Automation Manager will display the trigger from the selected configuration. To start the trigger, select it and click **Start**.

NOTE

If there are compatibility issues with NiceWatch configuration, open it in Automation Builder and reconfigure it.

9.10. Controlling Automation Service with Command-line Parameters

Read this section to learn how to use command prompt to:

- Start or stop Automation services.
- Control which configurations are loaded.
- Control which triggers are active.

NOTE

Make sure you are running **Command Prompt** in elevated mode (with administrative permissions). Right-click `cmd.exe` and select **Run as Administrator**.

Starting and Stopping the Services

To start both services, use the following commands:

```
net start NiceLabelProxyService2019

net start NiceLabelAutomationService2019
```

If you want to open configuration file when the Service is started, use:

```
net start NiceLabelAutomationService2019 [Configuration]
```

For example:

```
net start NiceLabelAutomationService2019 "c:\Project\configuration.MISX"
```

To stop services, use the following commands:

```
net stop NiceLabelProxyService2019

net stop NiceLabelAutomationService2019
```

Managing the Configurations and Triggers

NiceLabel Automation service can be controlled using Automation Manager command-line parameters. Use the following general syntax:

```
NiceLabelAutomationManager.exe COMMAND Configuration [TriggerName] [/SHOWUI]
```

NOTE

Include full path to the configuration name. Do not use the file name alone.

To ADD configuration

The provided configuration gets loaded into service. No trigger is started. If you include the **/SHOWUI** parameter, Automation Manager UI is started.

```
NiceLabelAutomationManager.exe ADD c:\Project\configuration.MISX /SHOWUI
```

To RELOAD configuration

The provided configuration gets reloaded into service. The running status of all triggers is preserved. Reloading the configuration forces the refresh of all files cached for this configuration. For more information, see section Caching Files. If you include the **/SHOWUI** parameter, Automation Manager UI is started.

```
NiceLabelAutomationManager.exe RELOAD c:\Project\configuration.MISX /SHOWUI
```

To REMOVE configuration

The provided configuration and all of its triggers are unloaded from service.

```
NiceLabelAutomationManager.exe REMOVE c:\Project\configuration.MISX
```

To START a trigger

The referenced trigger is started in the already loaded configuration.

```
NiceLabelAutomationManager.exe START c:\Project\configuration.MISX CSVTrigger
```

To STOP a trigger

The referenced trigger is stopped in the already loaded configuration.

```
NiceLabelAutomationManager.exe STOP c:\Project\configuration.MISX CSVTrigger
```

Status Codes
Status codes provide feedback about the command-line execution. To enable the status codes return, use the following command-line syntax:

```
start /wait NiceLabelAutomationManager.exe COMMAND Configuration [TriggerName]
[/SHOWUI]
```

The status codes are captured in the **`errorlevel`** system variable. To see the status code, execute the following command:

```
echo %errorlevel%
```

List of status codes:

Status Code	Description
0	No error occurred
100	Configuration file name not found
101	Configuration cannot be loaded
200	Trigger not found
201	Trigger cannot start

Providing User Credentials for Application Authentication
If you have configured the LMS Enterprise or LMS Pro system to use **Application Authentication** (not **Windows Authentication**), you have to provide user credentials with sufficient permissions to manage the configurations and triggers.

There are two command-line parameters you can use:

- -USER:**`[username]`**. Where **`[username]`** is a placeholder for the actual user name.
- -PASSWORD:**`[password]`**. Where **`[password]`** is a placeholder for the actual password.

9.11. Database Connection String Replacement

Configuration file for Automation Service can include database connection string replacement commands.

You can configure the service to replace certain parts of a connection string while the trigger is running. This enables a single instance of Automation to run the same configuration, but actually use a different database server for database related functionality. This allows the user to configure triggers in development environments and run them in production environment without any changes in the configuration.

Connection string replacement logic is defined in the **`DatabaseConnections.Config`** file that is located in the Automation system folder.

```
%PROGRAMDATA%\NiceLabel\NiceLabel 10
```

Configuration file defines from-to pairs using its XML structure. The **`<Replacement>`** element contains one **`<From>`** and one **`<To>`** element. During trigger execution, the "from" string is replaced with the "to" string. You can define as many **`<Replacement>`** elements as necessary.

Configuration file is not installed along with Automation. You can add it yourself using structure from the example below. The same search & replace rules are applied to all triggers running in the Automation Service on this machine.

NOTE

Make sure you restart both Automation services after adding the config file into the Automation System folder.

Example

An existing trigger contains connection to the Microsoft SQL server named **mySQLServer** and the database named **myDatabase**. You want to update the connection string to use the database **NEW_myDatabase** on the server **NEW_mySQLServer**.

Two replacement elements have to be defined – one to change the server name and one to change the database name.

```
<?xml version="1.0" encoding="UTF-8"?>
<DatabaseConnectionReplacements>
    <Replacement>
        <From>Data Source=mySQLServer</From>
        <To>Data Source=NEW_mySQLServer</To>
    </Replacement>
    <Replacement>
        <From>Initial Catalog=myDatabase</From>
        <To>Initial Catalog=NEW_myDatabase</To>
    </Replacement>
</DatabaseConnectionReplacements>
```

9.12. Entering Special Characters (Control Codes)

Special characters or control codes are binary characters that are not represented on the keyboard. You cannot type them the way normal characters are because they must be encoded using a special syntax. You would need to use such characters when communicating with serial-port devices, receiving data on TCP/IP port, or when working with binary files, such as print files.

There are two methods of entering special characters:

- **Enter the characters manually** using one of the described syntax examples:
  - Use syntax **<special_character_acronim>**, such as **<FF>** for FormFeed, or **<CR>** for CarriageReturn, or **<CR><LF>** for newline.
  - Use syntax **<#hex_code>**, such as **<#0D>** (decimal 13) for CarriageReturn or **<#00>** for null character.

  For more information, see section List of Control Codes.
- **Insert the listed characters.** Objects that support special characters as their content have an arrow button on their right side. The button contains a shortcut to all of the available special characters. When you select a character in the list, it is added to the content. For more information, see section Using Compound Values.

9.13. List of Control Codes

ASCII Code	Abbreviation	Description
1	SOH	Start of Heading
2	STX	Start of Text
3	ETX	End of Text
4	EOT	End of Transmission
5	ENQ	Inquiry
6	ACK	Acknowledgment
7	BEL	Bell
8	BS	Back Space
9	HT	Horizontal Tab
10	LF	Line Feed
11	VT	Vertical Tab
12	FF	Form Feed
13	CR	Carriage Return
14	SO	Shift Out
15	SI	Shift In
16	DLE	Data Link Escape
17	DC1	XON - Device Control 1
18	DC2	Device Control 2
19	DC3	XOFF - Device Control 3
20	DC4	Device Control 4
21	NAK	Negative Acknowledgment
22	SYN	Synchronous Idle
23	ETB	End Transmission Block
24	CAN	Cancel
25	EM	End of Medium
26	SUB	Substitute
27	ESC	Escape
28	FS	File Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator
188	FNC1	Function Code 1
189	FNC2	Function Code 2

190	FNC3	Function Code 3
191	FNC4	Function Code 4

9.14. Licensing and Printer Usage

Depending on the license type, your NiceLabel product might be limited to a number of printers you can use simultaneously. In case of a multi-user license NiceLabel keeps track of the number and names of different printers you have used for printing on all NiceLabel clients in your environment. The unique printer identifier is a combination of printer driver name (not printer name), printer location and port.

"To use a printer" means that one of the below listed actions has been taken within the Automation configuration:

- Print Label
- Send Data To Printer
- Preview Label
- Set Print Parameter

Each of these actions signalizes that a printer has been used. The associated printer is added to the list of used printers and remains listed for 7 days since last usage. To remove a printer from the list, do not use it for a period of 7 days and it will be removed automatically. The software displays the **Last Used** information so you know when the 7-day period passes for each printer. You can bind a printer seat to a specific printer by clicking the **Reserved** check box. Reservation ensures printer availability at all times – even if a printer has been idle for more than 7 days.

WARNING

If you exceed the number of seats defined by your license, the software enters the 30-day grace period. While in this mode, the number or allowed printers is temporarily increased to twice the number of printers allowed by license.

Grace period provides plenty of time to resolve licensing issues without any printing downtime or loss of ability to design labels. Exceeded number of allowed printers is usually an effect of printer replacement in your environment. It happens if old and new printers are used simultaneously, or if you add new printers. If you do not resolve license violation within the 30-day grace period, the number of available printers is reduced to the number of purchased seats starting with the last used printer on the list.

TIP

To learn more about NiceLabel 10 licensing, read the dedicated document – NiceLabel 10 Licensing.

9.15. Running in Service Mode

NiceLabel Automation runs as a Windows service and is designed not to require any user intervention while processing data and executing actions. The service is configured to start when the operating system is booted and runs in the background for as long as Windows is running. NiceLabel Automation remembers the list of all loaded configurations and active triggers. The last-known state is automatically restored when the server restarts.

The service runs with privileges of the user account selected during the installation. The service inherits all access permissions of that user account, including access to network shared resources, such as network drives, and printer drivers. Use the account of an existing user with sufficient privileges, or even better, create a dedicated account just for NiceLabel Automation.

You can manage the service by launching the Services from Windows Control Panel. In a modern Windows operating system, you can also manage the services using Services tab in Windows Task Manager. You would use Services to execute tasks such as:

- Starting and stopping the service.
- Changing the account under which the service logs on.

Good Practices Configuring the User Account for Service

- While being possible, it is considered a bad practice to run the service under the Local System Account. This is a predefined local Windows account with extensive privileges on the local computer, but is usually without privileges to access network resources. NiceLabel Automation requires full access to the account's `%temp%` folder, which might not be available for Local System Account.
- If creating a **new user account** for NiceLabel Automation service, make sure that you log in Windows with this new user for at least once. This makes sure that the user account is fully created. E.g., when you log in, the temporary `%temp%` folder is created.
- Disable the requirement to occasionally change password for this user account.
- Make sure the account has permissions to **Log on as service**.
- Run the Service in x64 (64-bit) mode.

Accessing Resources

NiceLabel Automation inherits all privileges from the Windows user account under which the service runs. The service executes all actions under that account name. Label can be opened if the account has permissions to access the file. Label can be printed if the account has access to the printer driver.

If using revision control system and approval steps inside Document Storage in Control Center, you have to make the service's user account a member of the 'Print-Only' profile, such as **Operator**. When done, configure access permissions for the specific folder to **read-only** mode or profile Operator. This makes sure that NiceLabel Automation only uses the approved labels, not drafts.

For more information, see section Access to Network Shared Resources.

Service Mode: 32-bit vs 64-bit

NiceLabel Automation can run on 32-bit (x86) and 64-bit (x64) systems natively. The execution mode is auto-determined by the Windows operating system. NiceLabel Automation runs in 64-bit mode on 64-bit Windows, and in 32-bit mode on 32-bit Windows.

- **Printing:** There are benefits of running Automation as a 64-bit process, such as direct communication with the 64-bit printer Spooler service on 64-bit Windows. This eliminates the well-known issue caused by the SPLWOW64.EXE, which is a 'middleware' that enables 32-bit applications to use the 64-bit printer spooler service.
- **Database access:** While running as a 64-bit process, NiceLabel Automation Service requires the 64-bit version of database drivers to be able to access the data. For more information, see section Accessing Databases.

NOTE

If you don't have 64-bit database drivers for your database, you cannot use NiceLabel Automation in 64-bit mode. You have to install it on a 32-bit system, or force it to run in 32-bit mode.

Forcing x86 Operation Mode on Windows x64

There might be reasons to run NiceLabel Automation as a 32-bit application on 64-bit Windows.

To force NiceLabel Automation into x86 mode on Windows x64, do the following:

- Select Start > Run.
- Type in **regedit** and press Enter.
- Navigate to the key

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\services\NiceLabelAutomationService2019
```

- Change the file name to NiceLabelAutomationService10.x86.exe, while keeping the existing path.
- Restart NiceLabel Automation service.

WARNING

It is not recommended to change the NiceLabel Automation service mode. If you decide to change it anyway, make sure you perform an extensive trigger testing before deploying the configuration in production environment.

9.16. Search Order for Requested Files

When loading a specified label or image file, NiceLabel Automation tries to locate the requested file on multiple predefined locations.

NiceLabel Automation locates the file in the following order:

1. Check if the file exists on the location as specified by the action.
2. Check if the file exists in the same folder as the configuration file (.MISX).
3. Check if the label file exists in .\Labels folder (for graphic files check .\Graphics folder).
4. Check if the label file exists in ..\Labels folder (for graphic files check ..\Graphics folder).
5. Check if the file exists in the global Labels folder (Graphics folder for graphics files) as configured in the options.

If the file is not found on any of these locations, the action fails and the error is raised.

9.17. Securing Access to your Triggers

Certain deployments require secured access to the triggers. NiceLabel Automation allows you to enable security measures that grant trigger access only to trustworthy network devices. Security configuration depends on the trigger type. Some of the trigger types allow configuration of security access by design. For all triggers that are based on TCP/IP protocol, you can further define all details within the Windows Firewall.

Configuring Firewall

When using TCP/IP based triggers, such as TCP/IP Server Trigger, HTTP Server Trigger or Web Service Trigger make sure you allow external applications to connect to the triggers. Each trigger runs within NiceLabel Automation service, to which access is governed by Windows Firewall.

NOTE

By default, the Windows Firewall is configured to allow all inbound connections to the NiceLabel Automation service. This makes it easier for you to configure and test triggers, but can be susceptible to unauthorized access.

If the NiceLabel Automation deployment in your company is a subject to strict security regulations, you must update the firewall rules according to them.

For example:

- You can fine-tune the firewall to accept incoming traffic from well-known sources only.
- You can allow inbound data only on pre-defined ports.
- You can allow connection only from certain users.
- You can define on which interfaces your will accept incoming connection.

To make changes in the Windows Firewall, open the **Windows Firewall with Advanced Security** management console from **Control Panel** > **System And Security** -> **Windows Firewall** > **Advanced Settings**.

NOTE

If NiceLabel Automation is linked to NiceLabel Control Center products, make sure you enable inbound connection on port **56415/TCP**. If you close this port, you won't be able to manage NiceLabel Automation from Control Center.

Allowing Access Based on the File Access Permissions

File trigger executes upon the time-stamp-change event in the monitored file or files. You must place the trigger files in a folder, which the NiceLabel Automation service can access. The user account running the Service must be able to access the files. Simultaneously, access permissions to the location also determine, which user and/or application can save the trigger file. You should set up access permissions in a way that only authorized users can save the files.

Allowing Access Based on the IP Address & Hostname

You can protect access to TCP/IP Server trigger with two lists of IP addresses and host names.

- The first list **'Allow connections from the following hosts'** contains IP addresses or host names of devices that can send data to the trigger. If a device has its IP address listed here, it is allowed to send data to the trigger.
- The second list **'Deny connections from the following hosts'** contains IP addresses or host names of devices that are not allowed to send data. If a device has its IP address listed here, it is not allowed to send data to the trigger.

Allowing Access Based on User names & Passwords

You can protect access to HTTP Server trigger by enabling the user authentication. When enabled, each HTTP request sent to the HTTP Server trigger must include the **'user name & password'** combination that matches the defined combination.

Allowing Access Based on Application Group Membership

You can protect access to HTTP Server trigger adding users to an application group in Control Center. With this option enabled, only authenticated members of this group are allowed to access the trigger.

9.18. Session Printing

Session printing enables printing of multiple labels using a single print job. If session printing is enabled, the printer receives, processes and prints all labels in the print job at once. As a result, printing speed increases due to continuous process of bundled label printing.

TIP

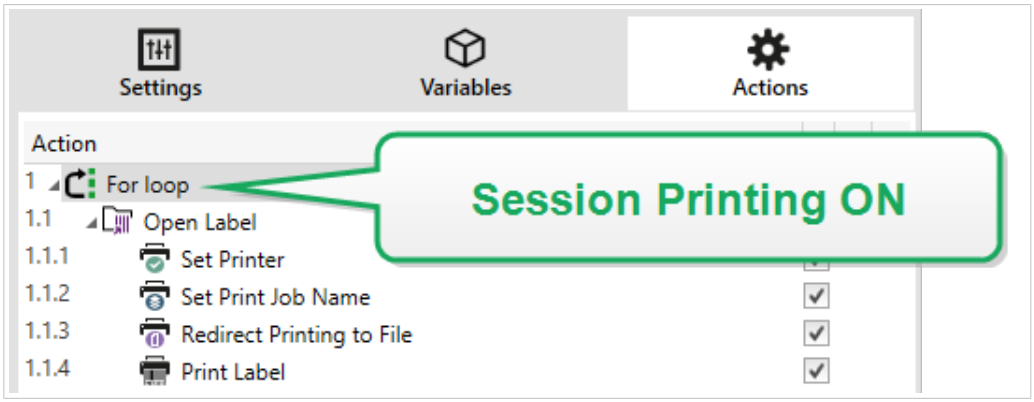
Session printing serves as an alternative to the normally used non-session printing, during which each label is sent to a printer as a separate print job.

NOTE

Automation activates session printing automatically based on the configuration of actions.

How does session printing start?

Session printing automatically starts if For Loop, For Every Record or For Each Line actions are present in the workflow. In such case, the nested Print Label action automatically enables session printing. This means that print actions for all items in the loop are included in a single print job.



How does session printing end?

Each session printing ends either with a finished loop or with Print Label action combined with at least one of the following conditions:

- Printer changes. If you select another printer using the Set Printer action, session printing ends.
- Printer port changes. If you redirect the print job to a file using the Redirect Printing to File action, session printing ends.
- Label changes. If you select another label to be printed using Open Label action, session printing ends.
- Custom command that ends session printing is sent. If you send `SESSIONEND` command using the Send Custom Command action, session printing ends.

  NOTE

  In this case, `SESSIONEND` must be sent as the only item in Send Custom Command action. If you would like to send additional commands, use separate Send Custom Command actions.

NOTE

More complex configurations might have multiple loops nested within each other. In such case, session printing ends when the outermost parent loop exits.

9.19. Tips and Tricks for Using Variables in Actions

If using variables in NiceLabel Automation actions, follow these recommendations.

- **Enclose variables in square brackets.** If you have variables with spaces in their names and refer to variables in actions, such as Execute SQL Statement or Execute Script, enclose the variables in square brackets, as in **`[Product Name]`**. Also use square brackets if variable names are the same as reserved names, e.g. in the SQL Statement.
- **Place colon in front of the variable name.** To refer to variables in Execute SQL Statement action or in Database Trigger, you have to place a colon (**`:`**) in front of the variable name, such as **`:[Product ID]`**. The SQL parser understands it as 'variable value'.

```
SELECT * FROM MyTable WHERE ID = :[ProductID]
```

- **Convert values to integer for computation.** If you want to execute a numeric calculation with variables, make sure that you convert variable values to integers. Defining variables to be numeric only limits the acceptable characters for defining the value, but doesn't change the variable type.

NiceLabel Automation treats all variables of string type. In VBScript, you would use the function `CInt()`.

- **Set default / start up values for scripts.** When using variables in actions, make sure these variables have some default value assigned, or the script checking might fail. You can define default values in variable properties, or in a script (and remove them after testing the script).

9.20. Tracing Mode

By default, NiceLabel Automation logs events into log database. This includes higher-level information gathering, such as:

- action execution logging
- filter execution logging
- logging of trigger status updates

For more information, see section Event Logging Options.

However, default logging doesn't keep track of the deep under-the-hood executions. If troubleshooting is needed on the lower-level of the code execution, you must enable tracing mode. In this mode, NiceLabel Automation logs details about all internal executions that take place during trigger processing. Tracing mode should only be enabled during troubleshooting to collect logs and then disabled to resume normal operation.

WARNING

Tracing mode slows down processing. It should only be used when instructed so by NiceLabel technical support team.

Enabling the tracing mode

To enable the tracing mode, do the following:

1. Navigate to the NiceLabel Automation System folder.

```
%PROGRAMDATA%\NiceLabel\NiceLabel 10
```

2. Create backup copy of **`product.config`** file.
3. Open **`product.config`** file in text editor. The file has an XML structure.
4. Add the element **`Common/Diagnostics/Tracing/Enabled`** and assign value **True** to it.
   The file should have the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <Common>
        <Diagnostics>
            <Tracing>
                <Enabled>True</Enabled>
                <Folder>c:\Troubleshooting\TracingLogs</Folder>
            </Tracing>
        </Diagnostics>
    </Common>
...
</configuration>
```

5. After you save the file, NiceLabel Automation Service automatically applies the setting.
6. By default, tracing files (*.LOG file extension) appear in the same System folder.
   To override the log folder, use **`product.config`** file. Specify the custom log folder in the **`Folder`** element. This element is optional.
7. To confirm that tracing mode is enabled, start the Automation Manager. With tracing mode enabled, it displays text **Tracing has been enabled** in the notification pane above the trigger list.

9.21. Understanding Printer Settings and DEVMODE

NOTE

DEVMODE data structure is part of the GDI Print API structure in Windows. This section includes highly technical content, relevant only for specific requirements.

Whenever you print labels with NiceLabel software (or any document in Windows applications for that matter), the printing application reads printer settings as defined in the printer driver and applies them to the print job. The same label can be printed using different printers just by selecting another printer driver. Each time, printer settings of a newly selected printer apply to the label.

Printing a text document using two different laser printers usually produces the same or at least a comparable result. Printing labels using two different label printers can produce very inconsistent results. To produce comparable results, the same label file might require additional printer driver settings, such as adjustment of offsets, speed, and temperature of printing. NiceLabel also applies printer settings to every printout. By default, printer settings are saved inside the label file for the selected printer.

What is DEVMODE?

DEVMODE is a Windows structure that holds printer settings (initialization and environment information about a printer). It contains two parts: public and private. Public part contains data that is common to all printers. Private part contains data that is specific to a particular printer. Private part can be of variable length and contains all specific manufacturer related settings.

- **Public part:** This part encodes general settings that are exposed in the printer driver model, such as printer name, driver version, paper size, orientation, color, duplex, and similar. Public part remains unchanged any printer driver and does not support specifics related to label printers (thermal printers, industrial ink jet printers, laser engraving machines).
- **Private part:** This part encodes settings that are not available in the public part. NiceLabel printer drivers use this part to store the printer model-specific data, such as printing speed, temperature setting, offsets, print mode, media type, sensors, cutters, graphics encoding, RFID support, and similar. Data structure within the private part of DEVMODE is a stream of binary data defined by driver developers.

DEVMODE Changing

DEVMODE data structure is stored in Windows registry. There are two copies of the structure: default printer settings and user-specific printer settings. You can alter the DEVMODE (printer settings) by changing the parameters in the printer driver. The first two options are Windows related, while the third option is available with NiceLabel software.

- **Default printer settings:** These settings are defined in **Printer properties** > **Advanced tab** > **Printing Defaults**.
- **User specific settings:** These settings are stored separately for each user in the user's HKEY_CURRENT_USER registry key. By default, user specific settings are inherited from the printer's default settings. The user specific settings are defined in **Printer properties > Preferences**. All modifications here only affect the current user.
- **Label specific settings:** Label author that uses NiceLabel software can embed the DEVMODE into the label. This makes printer settings portable. If the label is copied to another computer, printer settings travel with it. To embed printer settings into the label, enable the option **Use custom printer settings saved in the label** using *Document Properties* in Designer Pro. You can change the in-label printers settings by selecting *Printer* panel in *Document Properties*.

Applying custom DEVMODE to printout

In NiceLabel Automation, you can open a label file and apply a custom DEVMODE to it. When printing a label, its design is taken from the .NLBL file, and the DEVMODE applies the specific printer-related formatting. This allows you to have a single label master. In such case, label printout remains the same no matter which printer you use for printing because optimal settings for that printer are applied.

To apply a custom DEVMODE to a label, you can use two options:

1. Using an action, more specifically parameter **Printer settings**.
2. JOB command file, more specifically command **SETPRINTPARAM** with parameter **PRINTERSETTINGS**. For more information, see section Using Custom Commands.

9.22. Using the Same User Account to Configure and Run Triggers

NiceLabel Automation Service always runs under credentials of the user account configured for the service. However, Automation Builder always runs under credentials of the currently logged-in user. The credentials of service account and currently logged-in account might be different.

While you can preview the trigger in Automation Builder without any issues, the Service might report an error message caused by credentials mismatch. While the currently logged-in user has permissions to access folders and printers, the user account that the Service uses might not.

You can test the execution of triggers in Automation Builder using the same credentials as Service does. To do so, run Automation Builder under the same user account as defined for the Service.

To run Automation Builder under a different user account, do the following:

1. Press and hold **Shift** key, then **right click** the Automation Builder icon.
2. Select **Run as different user**.
3. Enter the credentials for the same user, as used in NiceLabel Automation Service.
4. Click **OK**.

If you plan to frequently run Automation Builder under credentials of the other user account, see the Windows-provided command-line utility **RUNAS**. Use the switches `/user` to specify the user account, and `/savecred`. The latter allows you to enter the password only once, and then it is remembered for the next time.