



# *ActionQ* User Manual



Trigger Dynamic SQL, Emails, REST APIs, and Applications (exe, cms,  
PowerShell)  
as Responses To Events  
(Database, Email, Folder, PowerShell, Scheduled Task)

[www.MilletSoftware.com](http://www.MilletSoftware.com)

**Version 3.0.5.3**

(May 2024)

By

Ido Millet

5275 Rome Court, Erie PA 16509

[ido@MilletSoftware.com](mailto:ido@MilletSoftware.com)

(814) 825-6009

Disclaimer: These component and accompanying files are provided "as-is" by Ido Millet without assuming any responsibility for harm to computer systems, software, or data with which these files are used.

INTRODUCTION .....	5
INSTALLATION .....	6
LAUNCH.....	7
<b>INITIAL SETUP.....</b>	<b>8</b>
ENCRYPTED STRINGS.....	9
Using the Encrypted Strings Manager.....	9
DATABASE CONNECTIONS .....	10
Using the Connections Manager .....	10
Using the Encrypted Strings Dialog (old method) .....	11
EMAIL PROFILES .....	12
Using the Email Profiles Manager .....	12
Email Settings for the Service Administrator.....	13
Email_From Accounts for Events as Senders .....	14
EMAIL INBOX SETTINGS.....	15
Using the Email Inboxes (POP3) Manager .....	15
USE THE GLOBAL SETTINGS DIALOG.....	16
Set the ActionQ Service to Run Under a User Account .....	17
<b>SETTING UP EVENTS .....</b>	<b>18</b>
GLOBAL PROPERTIES .....	19
INDIVIDUAL EVENTS PROPERTIES .....	19
Example.....	19
Supported Event Types .....	20
Monitoring Frequency.....	20
Failure Retry Period .....	20
Automatic Snapshot Table .....	20
RESPONSE PROPERTIES .....	21
SQL Response .....	21
EXE or CMD Response .....	22
PowerShell Response .....	22
REST API Call Response.....	23
Email Response .....	25
VIEWING AND EDITING PROPERTIES .....	26
Embed Dynamic Tokens in Response Properties .....	27
Edit Text for Non-Response Properties.....	28
HTML Editor for Email Message Body .....	29
Using {aq_HTML_Table} Token in Email Responses .....	30
Using {aq_HTML_Table} Token in Email Responses .....	32
View Dynamic Tokens for Each Process .....	32
<b>EVENT TYPES.....</b>	<b>33</b>
"FOLDER_NOT_EMPTY" EVENT TYPE .....	33
Dynamic Tokens for Response Logic .....	34
Dynamic Tokens for Renaming Files .....	34
DATABASE EVENTS .....	35
Source_SQL_Template Editor .....	35
Using a Stored Procedure as Data Source .....	36
"DB_LastMaxN" Event Type.....	37
"DB_Flag" Event Type .....	38
"DB_Mirror_Skip_Remove_T1" Event Type .....	39
POWERSHELL "PS_MIRROR_SKIP_REMOVE_T1" EVENT TYPE .....	41
Calling REST APIs such as RabbitMQ.....	41
Typical Properties .....	41
Avoiding Duplicate Processing .....	42

"EMAIL_POP3_T1" EVENT TYPE .....	43
Typical Use Case:.....	43
Setting Default Email Client .....	44
Email Server and Inbox Settings .....	45
Event Settings.....	45
Standard Dynamic Tokens .....	47
Dynamic Tokens By Regular Expressions .....	47
Dynamic Tokens By Brackets .....	47
SQL_After Logic.....	47
<b>SETTING UP ALERTS FOR WINDOWS SCHEDULED TASK FAILURES .....</b>	<b>48</b>
<b>MONITORING SNAPSHOT AND MIRROR TABLES .....</b>	<b>49</b>
<b>OPTIONS .....</b>	<b>50</b>
CUSTOMIZING ABOUT WINDOW IN ACTIONQ MANAGER .....	50
<b>HANDLING FAILURES .....</b>	<b>50</b>
SETTING FAILURE RETRY PERIOD.....	51
<b>TECHNICAL NOTES.....</b>	<b>53</b>
DATABASE CONNECTIVITY .....	53
64 vs 32-Bit DSNs.....	53
NT Authentication.....	53
SQL STATEMENTS .....	54
SQL_After as SELECT Query .....	54
SQL Statement Subquery .....	55
SQL Server (fully qualify table names).....	56
PROGRAMDATA FOLDER .....	57
SETTINGS FILES (ACTIONQ.INI AND ACTIONQ_DATA.INI) .....	57
UNC PATHS RATHER THAN MAPPED DRIVES .....	57
Invisible Mapped Drives .....	57
Batch File to Make Mapped Drives Visible .....	57
QUEUEING EMAILS TO SMTPQ OUTGOING FOLDER.....	58
ACTIONQ SERVICE FAILS TO START .....	58
CUSTOM LOGGING (OVERRIDE APPSETTINGS.JSON) .....	58
<b>UPDATE HISTORY .....</b>	<b>59</b>
VERSION 3.0.5.3 (ENTERED TESTING MAY 12, 2024): .....	59
Mirror Table Features.....	59
Logging Features.....	59
Other Features .....	59
Fixes .....	59
VERSION 2.0.8.01 (FEBRUARY 28, 2024): .....	60
VERSION 2.0.2.01 (AUGUST 1, 2023): .....	60
VERSION 2.0.1.01 (JULY 04, 2023): .....	61
VERSION 2.0.0.01 (JUNE 12, 2023): .....	61
VERSION 1.1.97.02 (NOVEMBER 20, 2022): .....	61
VERSION 1.1.89.02 (JULY 10, 2022): .....	61
VERSION 1.1.0.95 (MARCH 2, 2022): .....	61
VERSION 1.1.0.93 (JANUARY 6, 2022): .....	62
VERSION 1.1.0.87 (NOVEMBER 6, 2021): .....	62
VERSION 1.1.0.83 (SEPTEMBER 19, 2021):.....	62
VERSION 1.1.0.82 (SEPTEMBER 8, 2021):.....	63
VERSION 1.1.0.74 (AUGUST 17, 2021):.....	63

VERSION 1.1.0.71 (JULY 23, 2021): ..... 63

VERSION 1.1.0.63 (JUNE 25, 2021): ..... 63

VERSION 1.1.0.61 (MARCH 29, 2021): ..... 64

VERSION 1.1.0.57 (FEBRUARY 9, 2021): ..... 64

VERSION 1.1.0.52 (JANUARY 24, 2021): ..... 64

VERSION 1.1.0.47 (JANUARY 13, 2021): ..... 64

VERSION 1.1.0.34 (DECEMBER 7, 2020): ..... 64

VERSION 1.1.0.33 (NOVEMBER 30, 2020): ..... 65

VERSION 1.1.0.31 (NOVEMBER 12, 2020): ..... 65

VERSION 1.1.0.22 (OCTOBER 3, 2020): ..... 65

VERSION 1.1.0.21 (SEPTEMBER 23, 2020): ..... 65

VERSION 1.1.0.12 (SEPTEMBER 7, 2020): ..... 65

VERSION 1.1.0.10 (AUGUST 27, 2020): ..... 65

VERSION 1.1.0.9 (AUGUST 25, 2020): ..... 65

VERSION 1.0.0.8 (AUGUST 16, 2020): ..... 66

VERSION 1.0.0.0 (AUGUST 2, 2020): ..... 66

## Introduction

**ActionQ** is an inexpensive BAM (*Business Activity Monitoring*) software. It can trigger, **within seconds**, **dynamic SQL**, **command lines**, **PowerShell scripts**, **REST API calls**, or **emails** in response to **database**, **email**, **PowerShell**, and **folder** events.

It can also alert you when Windows Task Scheduler fails to launch a task.

Typical triggers include new leads, documents, credit holds, inventory shortage, schedule delays, complaints, support requests, pricing exceptions, returns, defects, cancelations, and system exceptions.

**ActionQ** can use ODBC data source/target (including cloud databases). It can call any windows software (EXE, Batch Files, PowerShell, etc.) and pass dynamic arguments and parameters. In the case of *Visual CUT*, this supports reporting, exporting, printing, emailing, uploading/downloading/processing files, capturing/parsing email attachments, and facilitating approval workflows.

[video demo](#) [📺] of **automating responses to database & email events**.

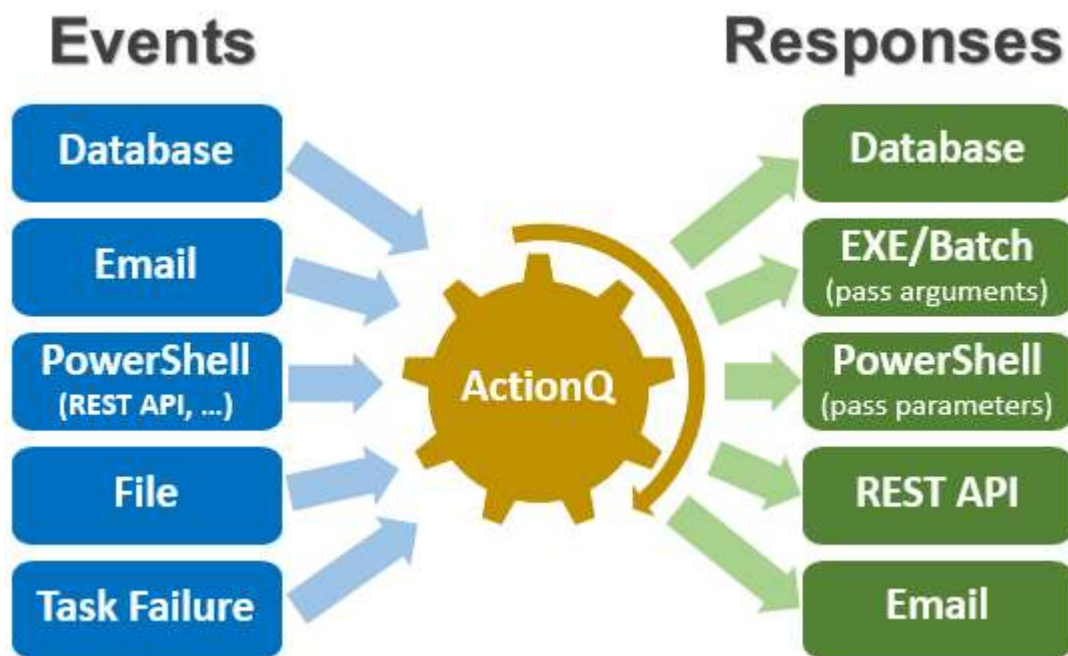
[video demo](#) [📺] of **calling a REST API** (Twilio's SMS service).

[video demo](#) [📺] of **monitoring the health of a SQL Server**.

[video demo](#) [📺] of **PowerShell scripts as events & responses**.

The software includes two programs:

1. **ActionQ\_Service.exe** is a **service**, always running in the background. It responds to events by triggering dynamic SQL statements, emails, command lines, or REST API calls. It can also move files, download email attachments, and download/delete processed emails from monitored inboxes.
2. **ActionQ\_Manager.exe** is a GUI for managing and monitoring the service. The service facilitates monitoring by also sending you emails when starting/stopping or encountering failures.



## Installation

Follow the instruction received via email.

After the install, use file explorer to locate:

**C:\Program Files\Millet Software\ActionQ\ActionQ\_Manager.exe**

Right-click this exe, Properties, Compatibility tab, and set the properties for all users to **run this EXE as administrator**.

If you forget to do so, a message box will remind you when you start the application.

Right-click this exe and '*pin to taskbar*' for easy access.



## Launch

When you start ActionQ\_Manager for the first time, it takes care of a) Copying the sample ActionQ.ini file from the installation folder to: **%APPDATA%\MilletSoftware\ActionQ\ActionQ.ini**  
Make sure users have MODIFY permissions on that folder.

If you get an error about *ChilkatDotNet47.dll*, your machine is probably missing this C++ runtime: [Microsoft Visual C++ Runtime 2017 64-bit](#) . Please download & install.

The initial Screen of ActionQ\_Manager allows you to:



**Manage Events** (create, edit, clone, delete).



Manage global settings such as sleep interval.



**Manage Encrypted String** (create and modify encrypted string for passwords and connection strings).



**Manage Database Connections** (create, edit, clone, delete, and **test** connections).



Manage Email Profiles (create, edit, clone, delete, and **test** email profiles).



Manage Email Inbox (POP3) Settings



**Install/Uninstall** and **Start/Stop** the ActionQ Service.

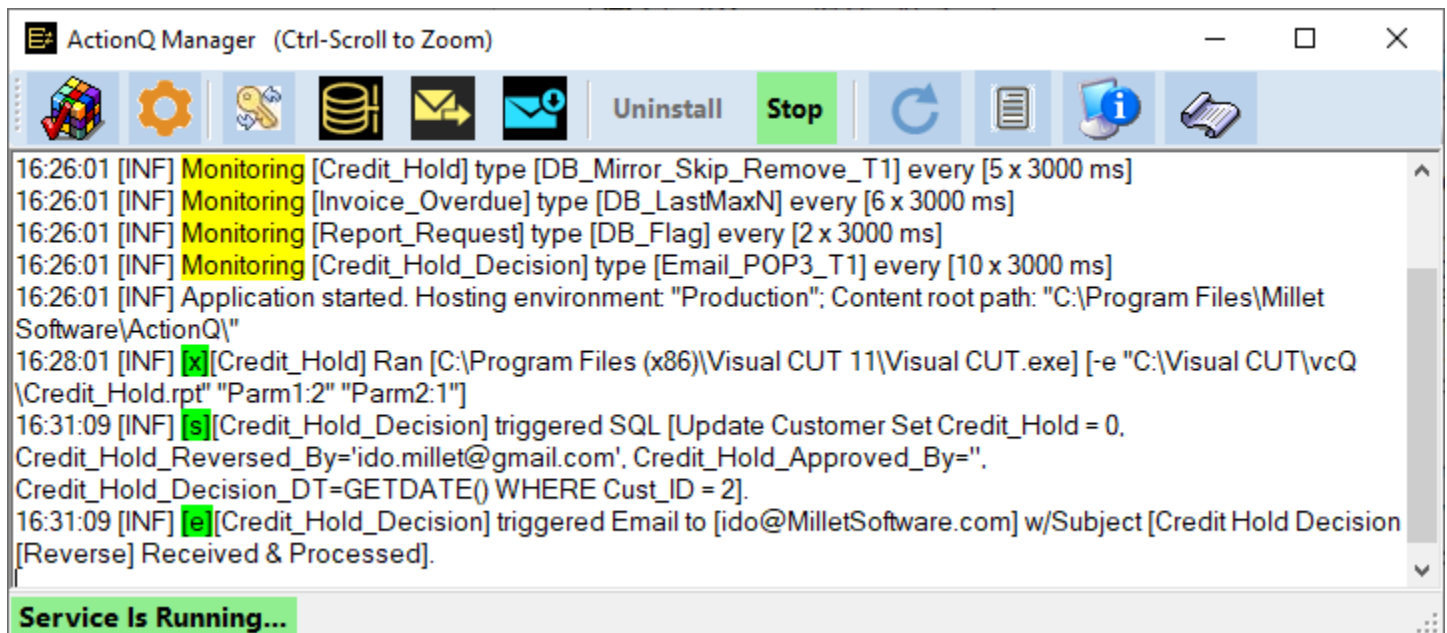


**Refresh the service log** (typically not needed since it auto-refreshes every few seconds).








Open the ActionQ.ini file (typically, not needed)

See  **Version Information** and  **Open the user manual**.



## Initial Setup

Before creating events you should set up:

1.  [Encrypted Strings](#) to protect sensitive information (e.g. passwords).  
in particular, **ES\_Email\_Password** you should select as the password for '**Email**' (the mandatory email profile) used to communicate administrative information (service start/stop, and failures).
2.  [Database Connections](#) to allow the service to connect to source/mirror databases (optional).
3.  [Email Profiles](#) for:
  - a. '**Email**' -- the **ActionQ service administrator** (so the service can communicate with you).
  - b. **Email\_From\_** accounts for events that send email (optional)
4.  Optional: [Email Inbox \(POP3\) Settings](#), for events that monitor **incoming emails**
5.  Optional: [Use the Global Settings Dialog](#),  
including the option to [Set the ActionQ Service to Run Under a User Account](#)

## Encrypted Strings

In the ActionQ.ini, a section called [Encrypted\_Strings] stores a set of named encrypted strings. Other sections can refer to these entries whose names always start with **ES\_** as a prefix. For example, **ES\_Email\_Password** protects a particular email password:

```
[Encrypted_Strings]
ES_Email_Password="C95EA425116ADF35E951A9486FB1B39B40637242E00A2F16DACE67"
ES_ERP_DB="196994C9944EF7124D518634935B53378F7F439AD99E12AA91E629"
ES_AQ_Mirror_DB="196994CB6E0ED179C832B24B9E07AC55B5B580C16298C06C06F28"
ES_Service_Account_Password="A6949FC149F8CB8371063E9953E520A0354AACD32D624AE5"
ES_MS_Invoices_Password="D22E83F37BF870AB4BED33AE3AA0ABE4CD3D731E68C50"
```

## Using the Encrypted Strings Manager



The button starts the dialog shown below. It allows you to create and modify named encrypted strings.

In this example, an encrypted string called **ES\_Email\_Password** being created for an email password.

The **ES Name** drop-down allows you to select and change previously created named encrypted strings.

The dialog box is titled "Encrypt & Save String to ActionQ.ini [Encrypted\_Strings] Section". It contains the following elements:

- ES Name:** A dropdown menu with "Email\_Password" selected.
- String to Encrypt:** A text input field containing a series of asterisks (\*\*\*\*\*).
- Preview:** A large text area on the right showing the resulting encrypted string: "ES\_Email\_Password=C95EA62912C441C425116ADF35E951A9486FB1B39B40637242E00A2F16DACE67".
- Buttons:** "Save to ActionQ.ini" and "Close".

## Database Connections


The service may need to connect to various databases to **detect** database events, **update** a database in response to an event, or **maintain a mirror table** to avoid duplicate processing. Connection properties within event settings simply name a connection. Those connection names are maintained in a [Connection\_Strings] section like this:

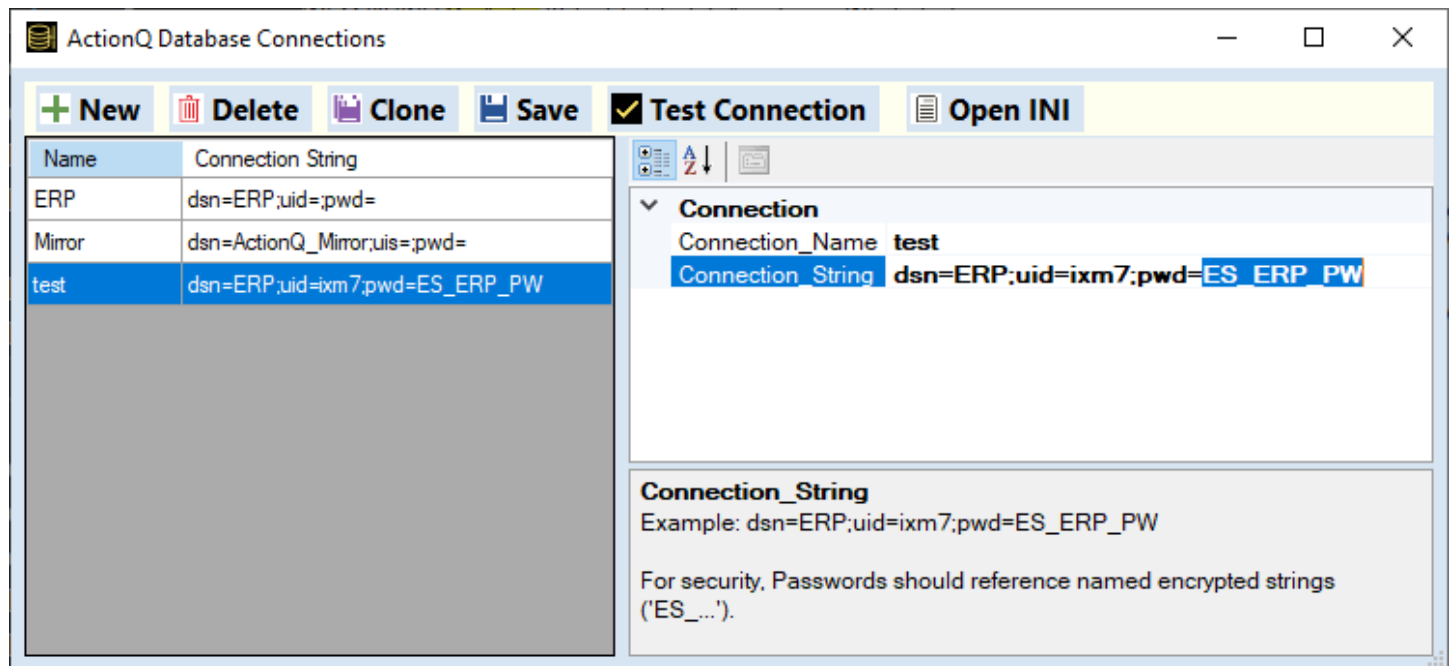
```
[Connection_Strings]
ERP="ES_ERP_DB"
Mirror="dsn=ERP;uid=ixm7;pwd=ES_ERP_PW"
```

If you wish to encrypt the whole connection string, you may do so as demonstrated by the first example above. Otherwise, protect just the password as demonstrated by the 2<sup>nd</sup> example.

You may edit that section manually using Notepad. But it is easier to use the dedicated connections manager.

### Using the Connections Manager

Clicking  starts a connection manager:



The left panel lists the connections (Name and Connection String).

The right panel is a property grid for a selected connection.

As shown in the image above, the connection string should refer to named encrypted strings instead of specifying sensitive information directly.

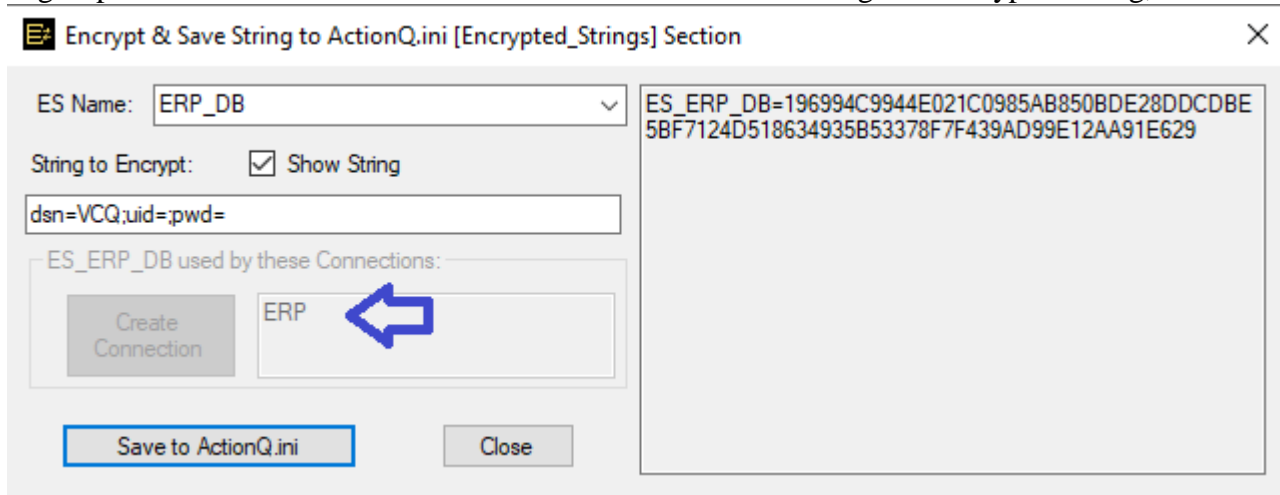
The **Test Connection** button allows you to test the connection. If a problem is found, a detailed error message is provided.

## Using the Encrypted Strings Dialog (old method)

Note: In most cases, avoid this old method. Instead, create connections [Using the Connection Manager](#).

As a somewhat less intuitive option, you may use the *Encrypted Strings* dialog, to also take care of adding a connection string entry to the ini file (where the whole connection string is encrypted).

If the **String to Encrypt** starts with ‘**dsn=...**’ ActionQ assumes it is for a connection string. A group box becomes visible and lists the named connections using that encrypted string, like this:



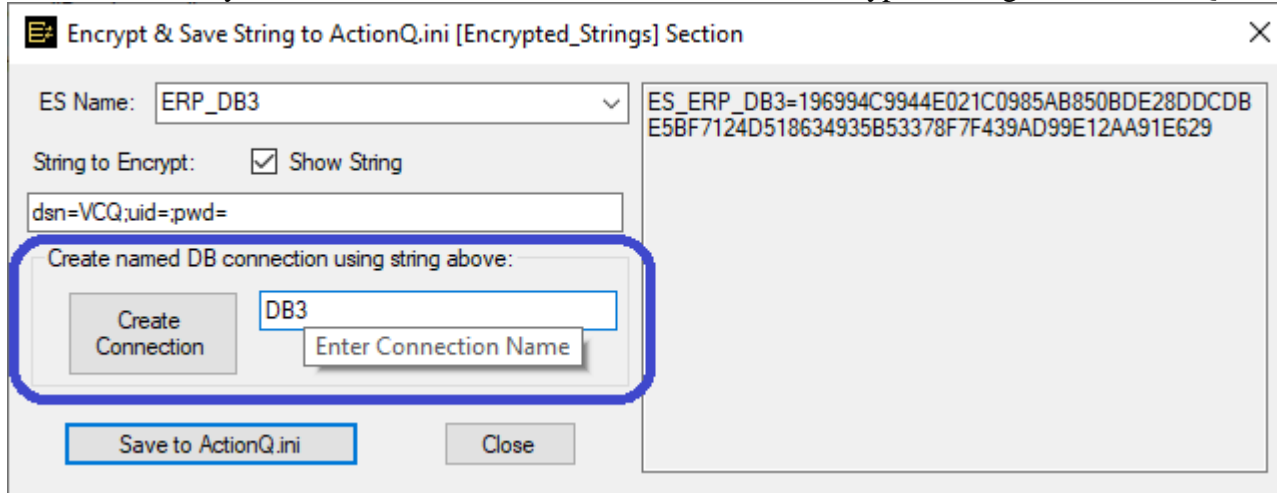
The dialog box is titled "Encrypt & Save String to ActionQ.ini [Encrypted\_Strings] Section". It contains the following elements:

- ES Name:** A dropdown menu with "ERP\_DB" selected.
- String to Encrypt:** A checkbox labeled "Show String" is checked. Below it is a text box containing "dsn=VCQ;uid=:pwd=".
- ES\_ERP\_DB used by these Connections:** A group box containing a "Create Connection" button and a list with the entry "ERP". A blue arrow points to the "ERP" entry.
- Buttons:** "Save to ActionQ.ini" and "Close".
- Encrypted String:** A large text area on the right displaying the encrypted string: "ES\_ERP\_DB=196994C9944E021C0985AB850BDE28DDCDBE5BF7124D518634935B53378F7F439AD99E12AA91E629".

In the case above, the encrypted string is used by a connection called ERP.

For a ‘**dsn=...**’ string that is not yet in use by a named connection, the group box becomes activated. You can then enter a name for the connection and click the ‘Create Connection’ button.

That automatically creates the named connection and saves the encrypted string in the ActionQ.ini file:



The dialog box is titled "Encrypt & Save String to ActionQ.ini [Encrypted\_Strings] Section". It contains the following elements:

- ES Name:** A dropdown menu with "ERP\_DB3" selected.
- String to Encrypt:** A checkbox labeled "Show String" is checked. Below it is a text box containing "dsn=VCQ;uid=:pwd=".
- Create named DB connection using string above:** A group box (highlighted with a blue border) containing a "Create Connection" button, a text box with "DB3", and a label "Enter Connection Name".
- Buttons:** "Save to ActionQ.ini" and "Close".
- Encrypted String:** A large text area on the right displaying the encrypted string: "ES\_ERP\_DB3=196994C9944E021C0985AB850BDE28DDCDBE5BF7124D518634935B53378F7F439AD99E12AA91E629".

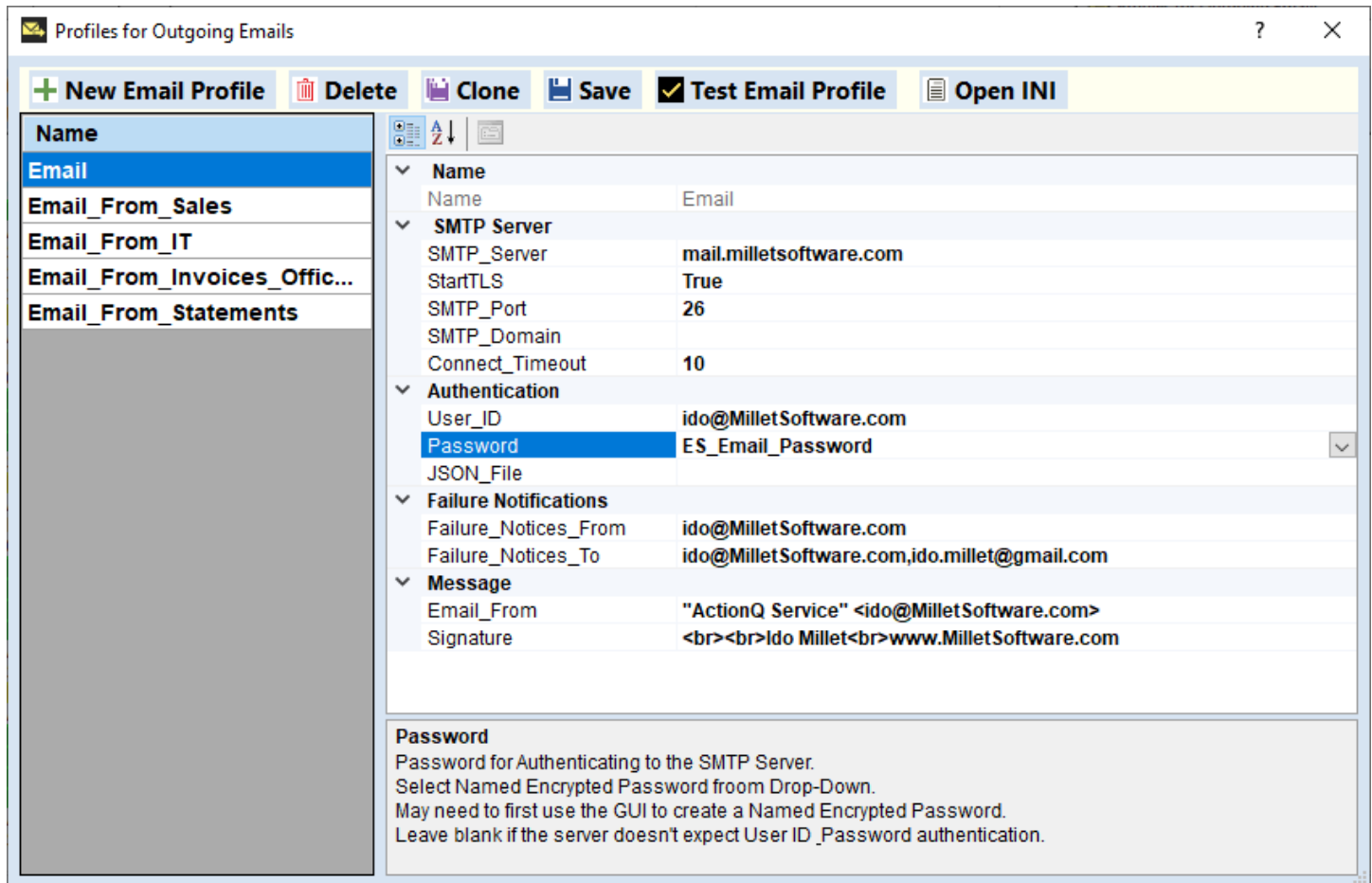
## Email Profiles

You need to set up one email profile called '**Email**' for notifying the administrator about the status of ActionQ. You can set up additional profiles (named as '**Email\_From\_???**') to allow custom email responses to events.

The user manual describes how these settings are saved as sections in the ini file. But instead of editing the ini file manually, you can simply use the **Email Profiles Manager**.

## Using the Email Profiles Manager

Clicking  starts the Email Profiles Manager:



Profiles for Outgoing Emails	
<b>+ New Email Profile</b> <b>Delete</b> <b>Clone</b> <b>Save</b> <b>Test Email Profile</b> <b>Open INI</b>	
<b>Name</b>	
<b>Email</b>	
<b>Email_From_Sales</b>	
<b>Email_From_IT</b>	
<b>Email_From_Invoices_Offic...</b>	
<b>Email_From_Statements</b>	
<b>Name</b>	Name Email
<b>SMTP Server</b>	SMTP_Server mail.milletsoftware.com StartTLS True SMTP_Port 26 SMTP_Domain Connect_Timeout 10
<b>Authentication</b>	User_ID ido@MilletSoftware.com Password ES_Email_Password JSON_File
<b>Failure Notifications</b>	Failure_Notices_From ido@MilletSoftware.com Failure_Notices_To ido@MilletSoftware.com,ido.millet@gmail.com
<b>Message</b>	Email_From "ActionQ Service" <ido@MilletSoftware.com> Signature   Ido Millet www.MilletSoftware.com
<b>Password</b>	Password for Authenticating to the SMTP Server. Select Named Encrypted Password from Drop-Down. May need to first use the GUI to create a Named Encrypted Password. Leave blank if the server doesn't expect User ID_Password authentication.

The left panel lists the email profiles. The right panel is a property grid for a selected profile.

The **Test Email Profile** button allows you to test the email. If a problem is found, a detailed error message is provided. If successful, a test message is sent to the addresses specified in the **Failure\_Notices\_To** property.

## Email Settings for the Service Administrator

The service needs to communicate with an administrator when it starts, stops, or encounters failures.

Open the `%APPDATA%\MilletSoftware\ActionQ\ActionQ.ini` file in Notepad and modify the **[Email]** settings to match yours (note: it is much **easier to use the GUI** to create/edit/clone these settings. See previous section.):

### [Email]

```
Email_SMTTP_Server="mail.milletsoftware.com"
Email_SMTTP_Port="26"
Email_SMTTP_Domain=""
Email_User_ID="ido@milletsoftware.com"
Email_Password="ES_Email_Password"
Email_StartTLS="True"
Email_Connect_Timeout="10"
; Email_From="" "ActionQ Service" <Email_Failure_Notices_From>
Email_Failure_Notices_From="ido@MilletSoftware.com"
Email_Failure_Notices_To="ido@MilletSoftware.com,ido.millet@gmail.com"
Email_Signature="<br><br>Ido Millet<br>www.MilletSoftware.com<br>(814) 825-6009"
Email_Auth_JSON_PATH="C:\ProgramData\MilletSoftware\ActionQ\Client_Secret_SMTTP_Office365.json"
```

Note how the **Email\_Password** entry is referring to an Encrypted String called `ES_Email_Password`. The creation of Encrypted String entries is described in the previous section.

For the service administrator, the **Email\_From** entry is ignored.

Instead, it is set from this logic: **"ActionQ Service" <Email\_Failure\_Notices\_From>**

The **Email\_Signature** option allows you to override the default email signature.

To insert line break, use `<br>` as shown in the example above.

The **Email\_Auth\_JSON\_PATH** is needed only when requiring **OAuth 2.0** via Gmail or Office365.

## Email\_From Accounts for Events as Senders

Each event that responds by sending an email message has an **Email\_From\_<Sender>** property pointing at an ini section that defines email sending properties. Different events may use the same Email\_From\_<Sender> property but each event also has unique properties such as email to/cc/bcc, email subject and message.

Here is an example of the ini section defining email sender properties for the Sales Department:

```
[Email_From_Sales]
Email_SMTP_Server="mail.milletsoftware.com"
Email_SMTP_Port="26"
Email_SMTP_Domain=""
Email_User_ID="ido@milletsoftware.com"
Email_Password="ES_Email_Password"
Email_StartTLS="True"
Email_Connect_Timeout="10"
Email_From="'"Sales Department'" <ido@MilletSoftware.com>'"
Email_Failure_Notices_From="ido@MilletSoftware.com"
Email_Failure_Notices_To="ido@MilletSoftware.com,ido.millet@gmail.com"
Email_Signature="<br><br>VP of Sales<br>www.MilletSoftware.com"
Email_Auth_JSON_PATH=
```

Note that these settings follow the same structure as the email settings for the Service Administrator (as described in the previous section).

The **Email\_Signature** option allows you to specify an email signature.  
To insert line break, use **<br>** as shown in the example above.

The **Email\_Auth\_JSON\_PATH** is needed only when requiring **OAuth 2.0** via *Gmail* or *Office365*.

Note how the **Email\_Password** entry is referring to an Encrypted String called **ES\_Email\_Password**.  
The creation of Encrypted String entries is described in a previous section.

## Email Inbox Settings

Several events may monitor the same inbox. Each event may specify a different filter based on conditions such as the content of the email subject. This is why the general properties of each inbox are maintained in a centralized ini section called **[Email\_Server\_InBoxes]**.

You need to set these entries only if you have events that monitor incoming emails.

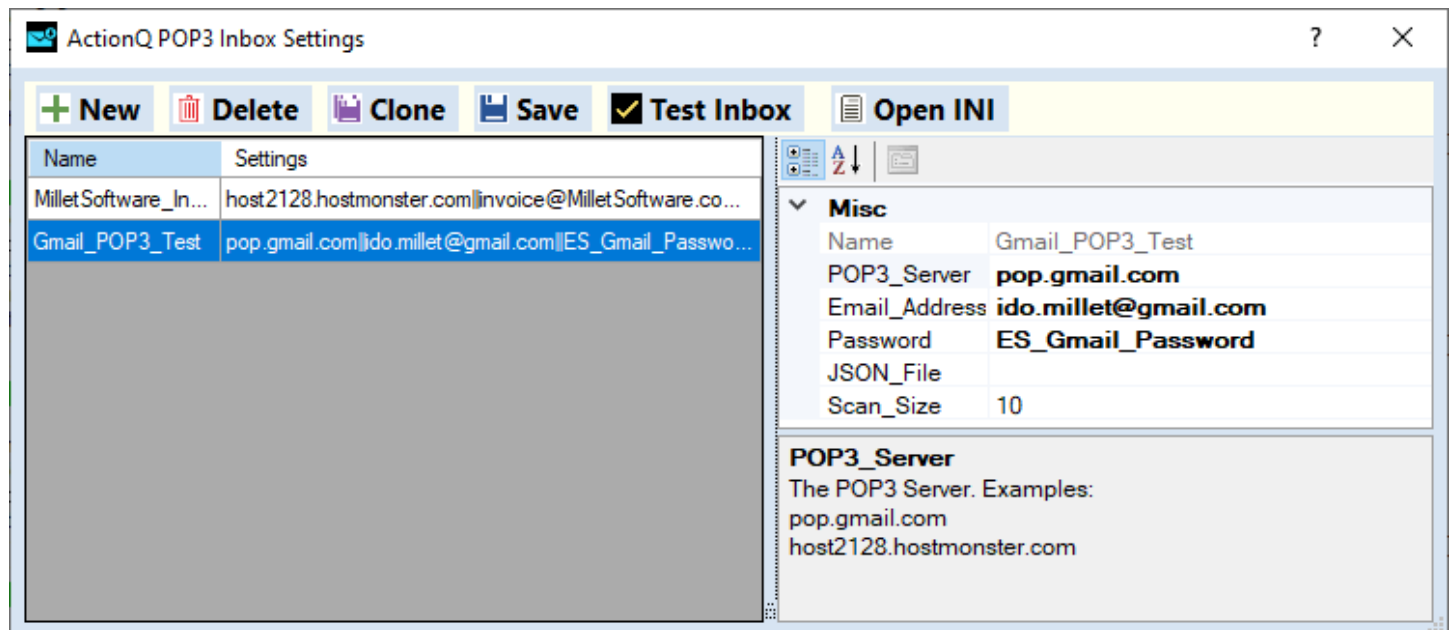
Here is an example of an inbox I monitor for invoice requests to trigger automated invoices via Visual CUT:

```
[Email_Server_InBoxes]
MilletSoftware_Invoice=host4.hostmonster.com||invoice@MilletSoftware.com||ES_MS_Password||10
```

The 4 elements are: a) the POP3 server, b) the inbox email account, c) the encrypted password name, and d) the maximum number of emails to inspect in each scan cycle.

## Using the Email Inboxes (POP3) Manager

Clicking  starts the Email Profiles Manager:



The left panel lists the inbox profiles.

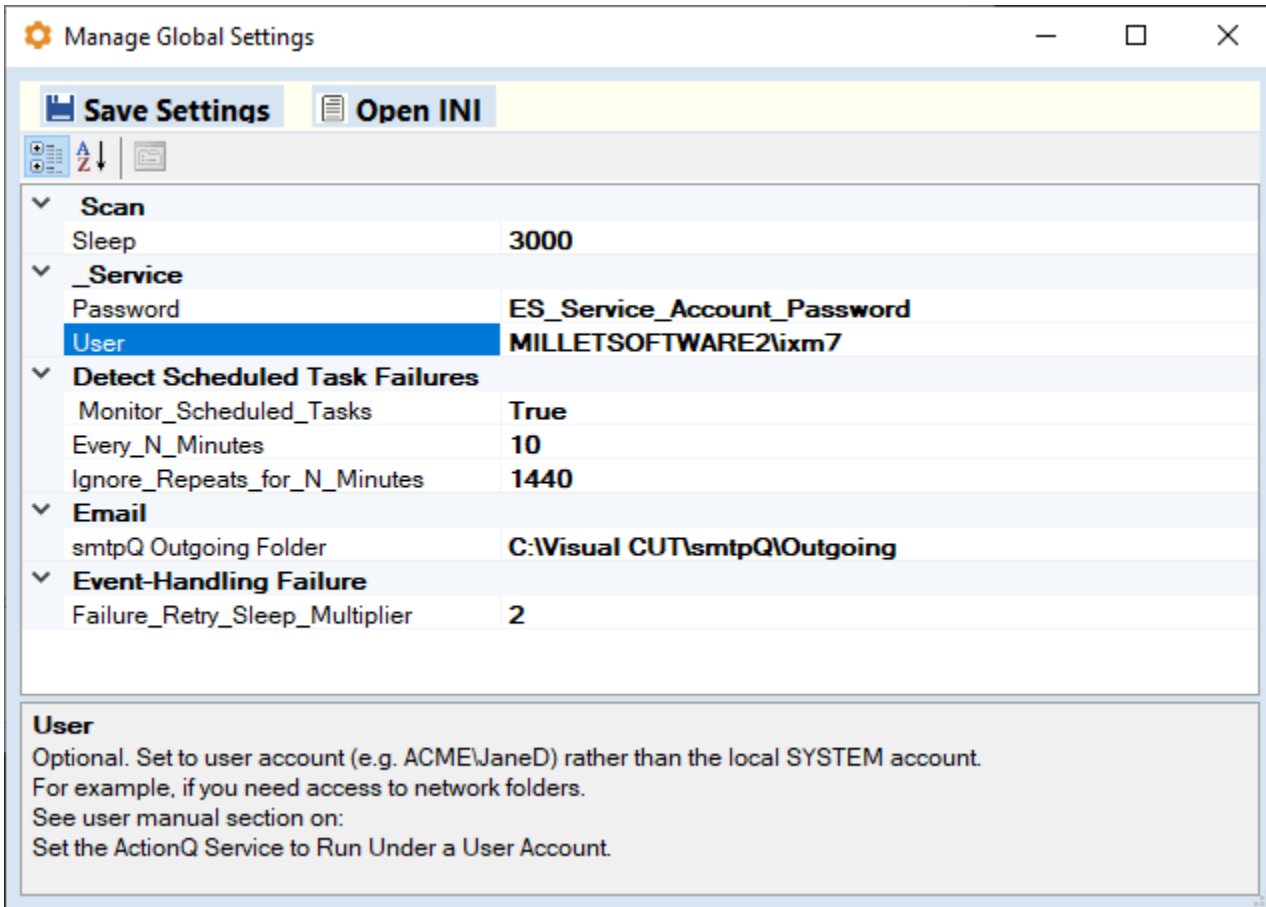
The right panel is a property grid for a selected profile.

The **JSON\_File** property is needed only when requiring **OAuth 2.0** via *Gmail* or *Office365*.

The **Test Inbox** button allows you to test connection and authentication to the inbox. If a problem is found, a detailed error message is provided. If successful, a message box confirms it.

## Use the Global Settings Dialog

Clicking  starts the Global Settings Manager:



Category	Property	Value
Scan	Sleep	3000
_Service	Password	ES_Service_Account_Password
	User	MILLETSOFTWARE2\ixm7
Detect Scheduled Task Failures	Monitor_Scheduled_Tasks	True
	Every_N_Minutes	10
	Ignore_Repeats_for_N_Minutes	1440
Email	smtpQ Outgoing Folder	C:\Visual CUT\smtpQ\Outgoing
	Event-Handling Failure	
	Failure_Retry_Sleep_Multiplier	2

**User**  
Optional. Set to user account (e.g. ACME\JaneD) rather than the local SYSTEM account.  
For example, if you need access to network folders.  
See user manual section on:  
Set the ActionQ Service to Run Under a User Account.

The bottom text area provides explanations for each property.

A typical use for this dialog is to set the ActionQ service to run as a user account (instead of the default local SYSTEM account). This is discussed in more detail in the following section.

## Set the ActionQ Service to Run Under a User Account

By default, the service will be running under the local SYSTEM account. That might mean that:


- Access to network folders might be a problem. For example, rpt files located in shared folders may not be recognized when triggering a Visual CUT process.
- Allowing the SYSTEM account to use NT Authentication to SQL Server might require the procedure described in [NT authentication](#).

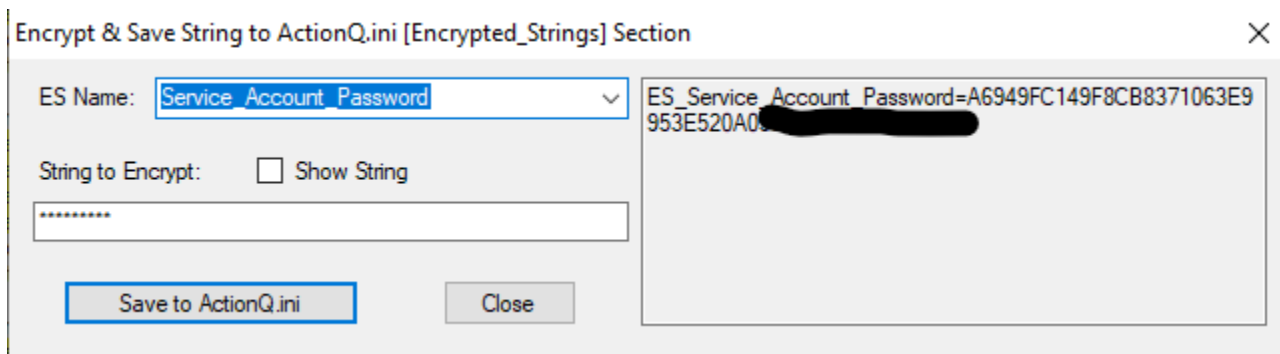
To address such scenarios, you can elect to run the ActionQ Service under a user account.

First, you need to add a Logon as a Service permission to that user account using the procedure described here:

[https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc794944\(v=ws.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc794944(v=ws.10)?redirectedfrom=MSDN)

Here are the steps:

- Start, Control Panel, Administrative Tools, Local Security Policy.**
- double-click **Local Policies**, and then click **User Rights Assignment**.
- In the details pane, double-click **Log on as a service**.
- Click **Add User or Group**, and  
add the user account to the list of accounts that possess the **Log on as a service** right.
- Click on the encrypt & save string button  and  
set a **Service\_Account\_Password** entry to the user's Windows login password.



Encrypt & Save String to ActionQ.ini [Encrypted\_Strings] Section


ES Name: **Service\_Account\_Password**

String to Encrypt: ☐ Show String

\*\*\*\*\*

**Save to ActionQ.ini** Close

ES\_Service\_Account\_Password=A6949FC149F8CB8371063E9953E520A0...

Then, use the Global Setting dialog  (as described in the previous section to set the User and Password for the service. These changes result in entries such as the example below in the ini file:

```
[Service]
User="MILLETSOFTWARE2\ixm7"
Password="ES_Service_Account_Password"
```

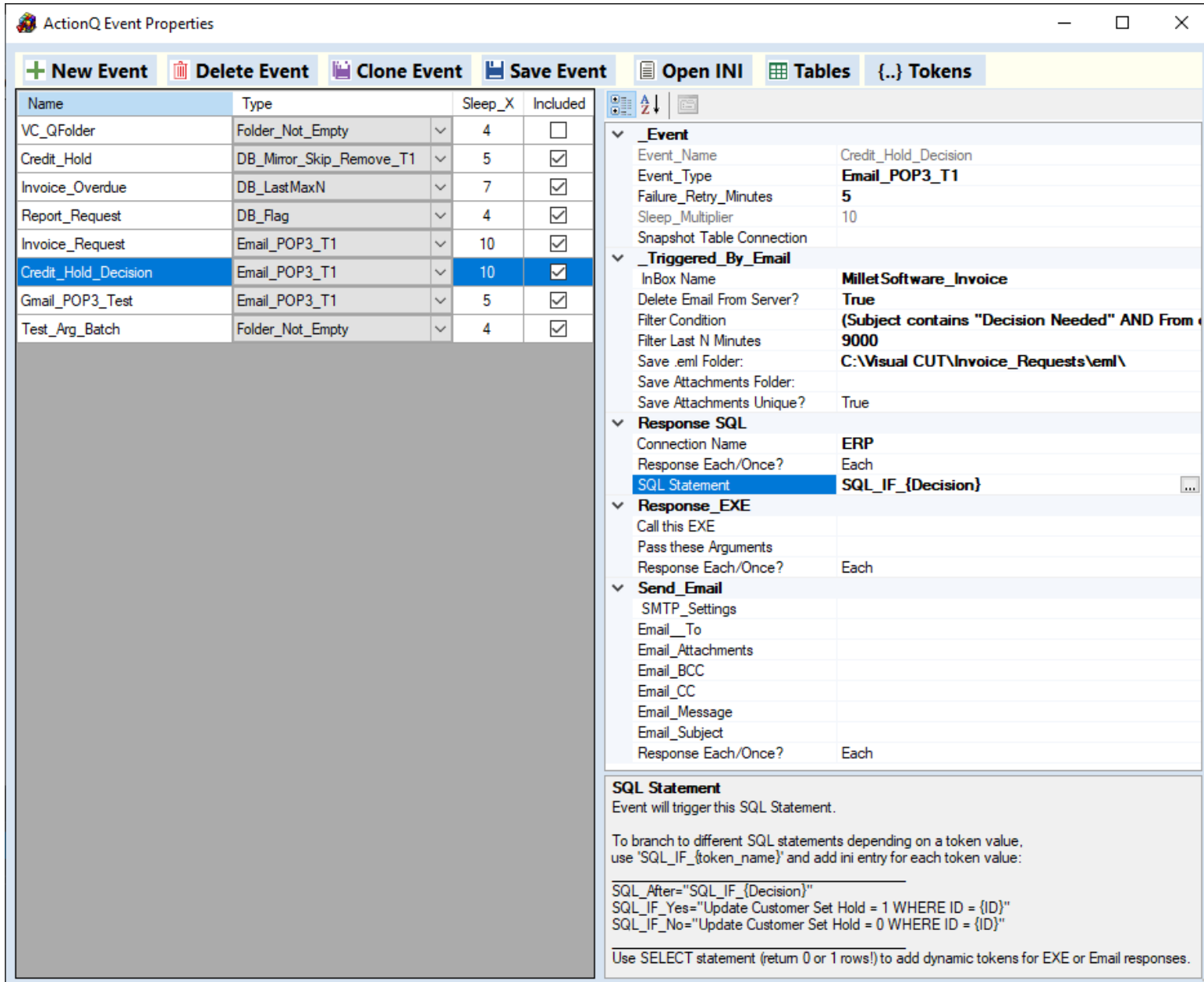
## Setting Up Events



Clicking the  button launches a window that allows you to create, clone, delete, and modify events:

The **event grid** on the left lists all defined events and allows you to set event type, sleep multiplier, and include/exclude the event from scanning by the ActionQ service.

The **Property Panel** on the right allows you to review/change properties for each event. The displayed properties adapt to the type of event. Common and unique event properties are discussed in the sections below.



Name	Type	Sleep_X	Included
VC_QFolder	Folder_Not_Empty	4	<input type="checkbox"/>
Credit_Hold	DB_Mirror_Skip_Remove_T1	5	<input checked="" type="checkbox"/>
Invoice_Overdue	DB_LastMaxN	7	<input checked="" type="checkbox"/>
Report_Request	DB_Flag	4	<input checked="" type="checkbox"/>
Invoice_Request	Email_POP3_T1	10	<input checked="" type="checkbox"/>
Credit_Hold_Decision	Email_POP3_T1	10	<input checked="" type="checkbox"/>
Gmail_POP3_Test	Email_POP3_T1	5	<input checked="" type="checkbox"/>
Test_Arg_Batch	Folder_Not_Empty	4	<input checked="" type="checkbox"/>

**Property Panel**

**\_Event**

Event\_Name: Credit\_Hold\_Decision  
Event\_Type: Email\_POP3\_T1  
Failure\_Retry\_Minutes: 5  
Sleep\_Multiplier: 10  
Snapshot\_Table\_Connection:

**\_Triggered\_By\_Email**

InBox Name: MilletSoftware\_Invoice  
Delete Email From Server?: True  
Filter Condition: (Subject contains "Decision Needed" AND From 9000  
Filter Last N Minutes: 9000  
Save .eml Folder: C:\Visual CUT\Invoice\_Requests\eml\  
Save Attachments Folder:  
Save Attachments Unique?: True

**Response\_SQL**

Connection Name: ERP  
Response Each/Once?: Each  
SQL Statement: SQL\_IF\_{Decision}

**Response\_EXE**

Call this EXE:  
Pass these Arguments:  
Response Each/Once?: Each

**Send\_Email**

SMTP\_Settings:  
Email\_To:  
Email\_Attachments:  
Email\_BCC:  
Email\_CC:  
Email\_Message:  
Email\_Subject:  
Response Each/Once?: Each

**SQL Statement**

Event will trigger this SQL Statement.

To branch to different SQL statements depending on a token value, use 'SQL\_IF\_{token\_name}' and add ini entry for each token value:

```
SQL_After="SQL_IF_{Decision}"
SQL_IF_Yes="Update Customer Set Hold = 1 WHERE ID = {ID}"
SQL_IF_No="Update Customer Set Hold = 0 WHERE ID = {ID}"
```

Use SELECT statement (return 0 or 1 rows!) to add dynamic tokens for EXE or Email responses.

## Global Properties

When the ActionQ service loads, it uses the following ini section in *ActionQ.ini* to detect the events to monitor and set its own sleep period after each monitor cycle.

```
[Events]
EventNames="VC_QFolder, Invoice_Overdue, Report_Request, Credit_Hold"
Sleep="5000"
Failure_Retry_Sleep_Multiplier="10"
```

In the example above, there are 4 events that the service will monitor. After each monitor cycle, the service will *sleep* for 5 seconds (5000 milliseconds).

A value of 10 in *Failure\_Retry\_Sleep\_Multiplier* tells the service that a failing event in a retry mode should be checked at a frequency 10 times lower than normal. For example, if *Failure\_Retry\_Sleep\_Multiplier* is set to 10, global *Sleep* is set to 5 Seconds, and the process has a *Sleep\_Multiplier* of 3, when that process enters failure retry mode, it would be checked every 150 seconds (5 x 3 x 10). If it recovers, it will immediately resume its normal frequency of every 15 seconds (5 x 3).

## Individual Events Properties

Each event has an ini section with its name.

### Example

Common properties for all events include the following example:

```
[Event_Name]
Event_Type="DB_Flag"
Sleep_Multiplier="3"
Failure_Retry_Minutes="5"
Snapshot_Table_Connection="Snapshot"
SQL_After_Connection=""
SQL_After=""
SQL_When="Each"
EXE2Call="C:\Program Files (x86)\Visual CUT 11\Visual CUT.exe"
Arguments_Template="-e "C:\Reports\Credit_Hold.rpt" "Parm1:{Cust_ID}"
Call_When="Once"
SMTP_Settings="Email_From_Sales"
Email_To="ido@MilletSoftware.com"
Email_Attachments="c:\temp\Attendance_{Student_ID}.xlsx; c:\temp\Grades_{Sudent_ID}.pdf"
Email_To_BCC=""
Email_To_CC=""
Email_Subject="Invoice Request"
Email_Message="Invoice Request Received from {Email_From}"
Email_When="Each"
```

## Supported Event Types

Currently, the supported **Event Types** are:

1. **"Folder\_Not\_Empty"** –detect files appearing in a local/cloud folder.
2. **"DB\_LastMaxN"** –detect new records based on an incrementing numeric column
3. **"DB\_Flag"** – detect records and delete or update them in the database to avoid duplication
4. **"DB\_Mirror\_Skip\_Remove\_T1"** – detect records based on comparisons to a mirror table
5. **"PS\_Mirror\_Skip\_Remove\_T1"** – same as above, using *PowerShell* script as the triggering event
6. **"Email\_POP3\_T1"** – detect new emails matching a filter condition. Trigger command lines and database updates. Optionally download the email message and/or attachments, and delete the email from the server.

## Monitoring Frequency

**Sleep\_Multiplier** tells the service to check that event only every Nth cycle. In the case above, a value of 3 indicates that the trigger condition for this event would be checked only every 3<sup>rd</sup> cycle.

## Failure Retry Period

**Failure\_Retry\_Minutes** (if > 0) is an optional setting available to all processes. It tells the service to retry monitoring this event even after a failure (e.g. connectivity to monitored database is down).

For details, see [Setting Failure Retry Period](#).

## Automatic Snapshot Table

**Snapshot\_Table\_Connection** (if specified) directs the service to load the event data (after removing cases due to any Mirror table logic) into a snapshot table. The table is created on the fly at the target database specified by the connection. It is named based on the event name like this: **aqt\_EventName**. (for example **aqt\_CreditHold**).

Besides facilitating debugging, a snapshot table can **expose the event data to downstream processes**.

For example, ActionQ can trigger a Visual CUT process whereby a report consumes the data in the snapshot table. This can remove the need to pass parameters via dynamic tokens in the arguments template.

## Response Properties

Each event can trigger EXE and/or SQL and/or Email responses.

### SQL Response

**SQL\_After\_Connection** and **SQL\_After** specify the connection name and the SQL statement to call.

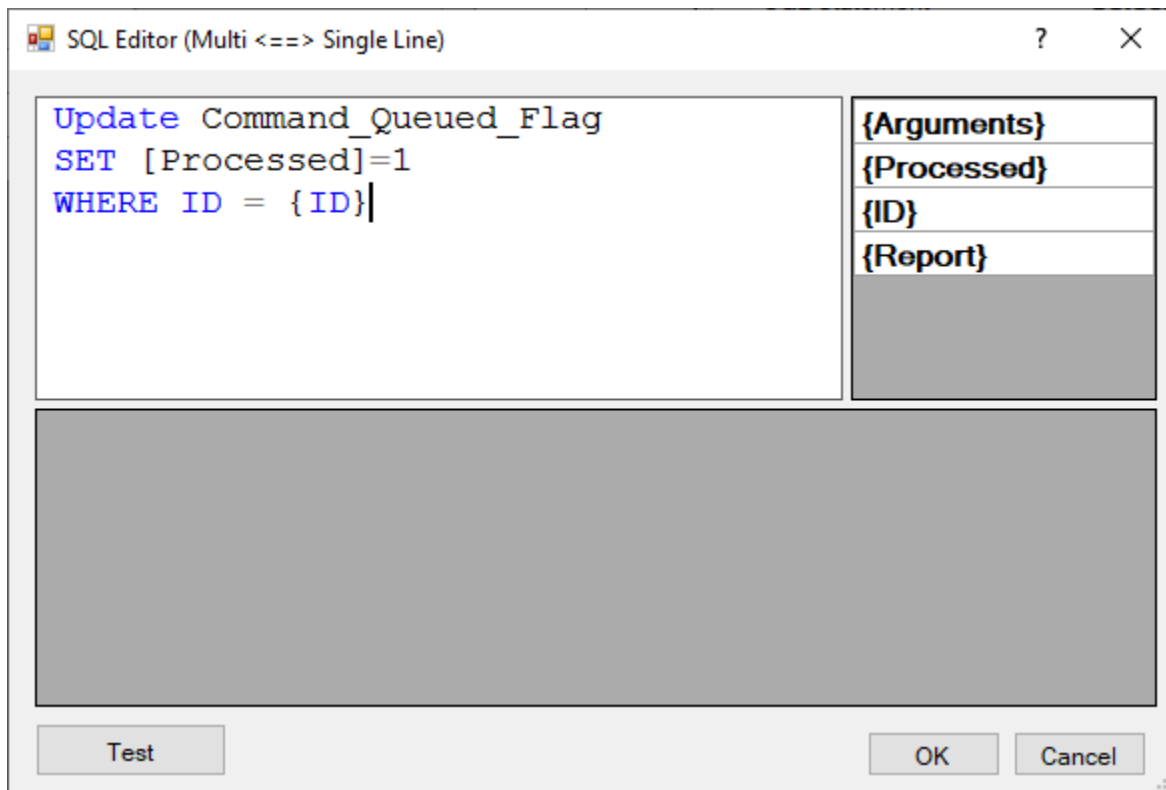
**SQL\_When** specifies if the SQL response is called for **each** Record/File/Email in the event or only **once**.

The SQL statement can use **dynamic tokens** to incorporate dynamic information from the event.

In the example above, these options are left blank, indicating no SQL response is needed.

If the SQL statement starts with SELECT, ActionQ assumed you wish to **add** **dynamic token** information for each column from the query. Make sure the SELECT statement returns only zero or one row. If zero rows are returned the dynamic tokens for each column name is set as blank (“”).

The dialog to edit the SQL Response provides syntax highlighting:



## EXE or CMD Response

**EXE2Call** and **Arguments\_Template** specify the EXE, Batch, or CMD files to call and the arguments to pass. The arguments can use **dynamic tokens** (for example, "Parm1:{Cust\_ID}") to incorporate dynamic information from the event.

**Call\_When** tells the service when to call the EXE2Call. A value of "Each" runs the EXE for each triggering event row (e.g. query row, email, file, PowerShell object). A value of "Once" triggers the response only once.

## PowerShell Response

See [video demo](#) of **PowerShell scripts as events & responses**.

Let's assume your script is at: **C:\PS\Script.ps1**

**Call\_When** tells the service when to call the PowerShell script. A value of "Each" runs the script for each triggering event row (e.g. query row, email, file, PowerShell object) matching the event filter conditions. A value of "Once" triggers the response only once.

Set **EXE2Call** to **Powershell.exe**

Set the **Arguments\_Template** to:

**-NoProfile -ExecutionPolicy Bypass -Command "C:\PS\Script.ps1";**

The **Arguments\_Template** can use **dynamic tokens** to incorporate dynamic information from the event. Typically, you would insert those dynamic tokens as parameters to the PowerShell Script, as discussed below:

### *With Named Parameters:*

If the script uses **named parameters**, add parameter **name** & **value** pairs to the end of the **Arguments\_Template** like this:

**-NoProfile -ExecutionPolicy Bypass -Command "C:\PS\RenameFile.ps1 -newName {FileName}";**

In the example above, the **{FileName}** may have a dynamic value provided by *Folder\_Not\_Empty* event.

### *More Complex Scenarios:*

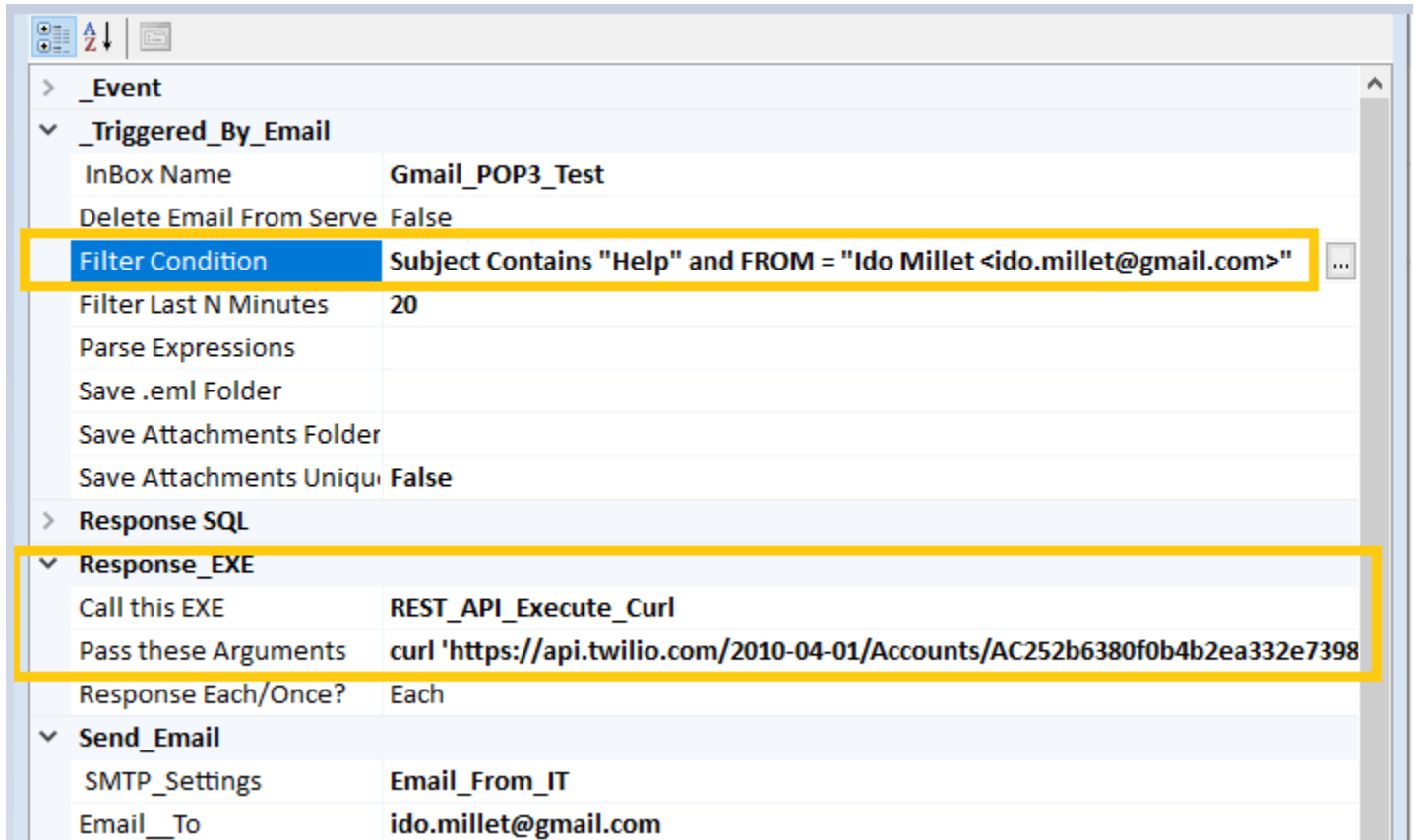
For more complex scenarios, such as running the script with Admin permissions, or handling values with embedded spaces, please refer to [this blog](#).

## REST API Call Response

See [Video demo](#) of **calling a REST API** (Twilio's SMS service).

Set the EXE name as **REST\_API\_Execute\_Curl**. The **Arguments\_Template** specifies the **cURL command**. Most REST API providers offer examples and online help for constructing cURL commands.

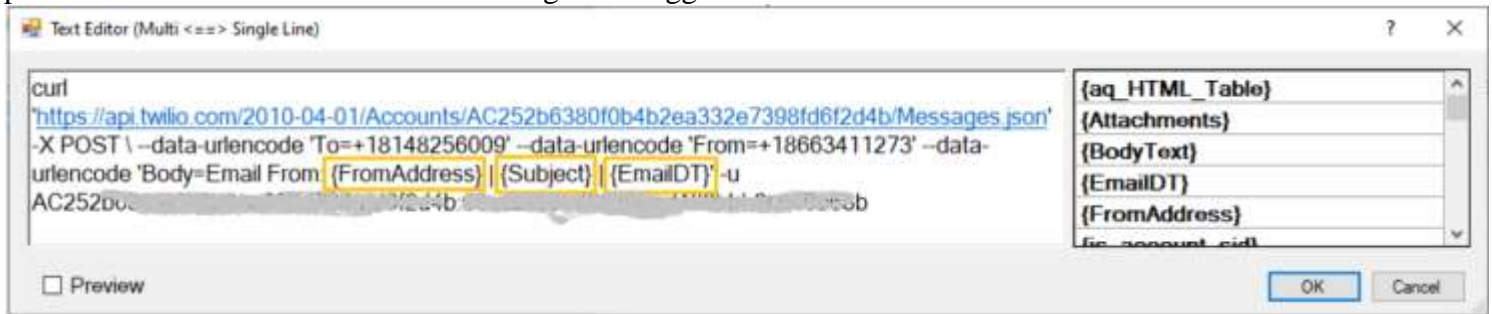
For example, here is an example of calling Twilio's REST API to send an SMS in response to an incoming email:



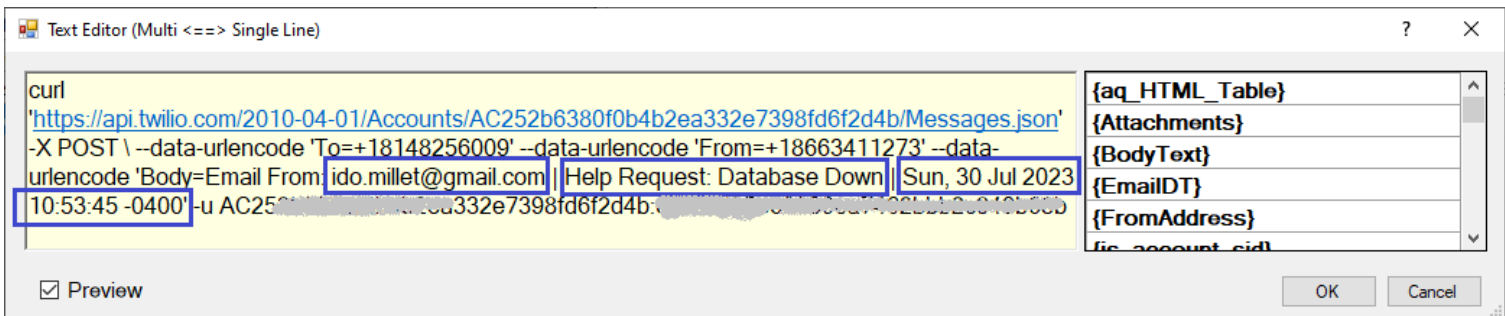
The screenshot shows a configuration window with a tree view on the left and a details pane on the right. The tree view has the following structure:

- > **\_Event**
  - ▼ **\_Triggered\_By\_Email**
    - InBox Name: Gmail\_POP3\_Test
    - Delete Email From Serve: False
    - Filter Condition**: Subject Contains "Help" and FROM = "Ido Millet <ido.millet@gmail.com>"
    - Filter Last N Minutes: 20
    - Parse Expressions
    - Save .eml Folder
    - Save Attachments Folder
    - Save Attachments Unique: False
  - > **Response SQL**
  - ▼ **Response\_EXE**
    - Call this EXE: REST\_API\_Execute\_Curl
    - Pass these Arguments: curl 'https://api.twilio.com/2010-04-01/Accounts/AC252b6380f0b4b2ea332e7398'
    - Response Each/Once?: Each
  - ▼ **Send\_Email**
    - SMTP\_Settings: Email\_From\_IT
    - Email\_To: ido.millet@gmail.com

Here is the ActionQ dialog for **editing the CURL command**. Note the dynamic tokens. In this example, they provide information from the email message that triggered the event.



Here are the dynamic token values shown when you turn on the Preview option:



**Note:** The JSON response is parsed into dynamic tokens that can be used in an email response.

## Email Response

**SMTP\_Settings** specifies the outgoing email profile that can be reused by multiple events to send emails as discussed in [Email\\_From\\_Settings](#).

The rest of the event email properties control the unique aspects (to/cc/bcc/subject/attachments/message) for emails sent by the event.

These properties can refer to **dynamic tokens** to incorporate information from the event.

For example, the **Email\_To** property can be set to **{FromAddress}** in the case of an event triggered by an email or to **{Contact\_Email}** in the case of an event triggered by SQL.

## Viewing and Editing Properties



When clicking the  button the **Common Properties** are in the property groups of: *'Event'*, *'Response\_EXE'*, *'Response\_SQL'*, and *'Send\_Email'* as shown below:

ActionQ Event Properties

[+ New Event](#)
[Delete Event](#)
[Clone Event](#)
[Save Event](#)
[Open INI](#)
[Tables](#)
[{..} Tokens](#)

Name	Type	Sleep_X	Included
VC_QFolder	Folder_Not_Empty	4	<input type="checkbox"/>
Credit_Hold	DB_Mirror_Skip_Remove_T1	5	<input checked="" type="checkbox"/>
Invoice_Overdue	DB_LastMaxN	7	<input checked="" type="checkbox"/>
<b>Report_Request</b>	<b>DB_Flag</b>	<b>4</b>	<input checked="" type="checkbox"/>
Invoice_Request	Email_POP3_T1	10	<input checked="" type="checkbox"/>
Credit_Hold_Decision	Email_POP3_T1	10	<input checked="" type="checkbox"/>
Gmail_POP3_Test	Email_POP3_T1	5	<input checked="" type="checkbox"/>
Test_Arg_Batch	Folder_Not_Empty	4	<input checked="" type="checkbox"/>

**Common Properties**

**\_Event**

Event\_Name: Report\_Request

Event\_Type: **DB\_Flag**

Failure\_Retry\_Minutes: 0

Sleep\_Multiplier: 4

Snapshot Table Connection: **Mirror**

**Trigger-Type Properties**

**\_Triggered\_By\_SQL**

Connection Name: ERP

SQL Statement: **Select \* From Command\_Queue Flag Where [Processed] = 0**

**Response\_SQL**

Connection Name: ERP

Response Each/Once?: Each

**SQL Statement**: Update Command\_Queue Flag SET [Processed]=1 WHERE ID = {ID}

**Response\_EXE**

Call this EXE: C:\Program Files (x86)\Visual CUT 11\Visual CUT.exe

Pass these Arguments: -e {Report} {Arguments}

Response Each/Once?: Each

**Common Response Properties**

**Send\_Email**

SMTP\_Settings: Email\_From\_IT

Email\_To: ido.millet@gmail.com

Email\_Attachments:

Email\_BCC:

Email\_CC:

Email\_Message: Your report request was received AND processed.<br>Contact IT Dept for

Email\_Subject: Report Request processed

Response Each/Once?: Each

**SQL Statement**

Event will trigger this SQL Statement.

To branch to different SQL statements depending on a token value, use 'SQL\_IF(token\_name)' and add ini entry for each token value:

SQL\_After="SQL\_IF(Decision)"

SQL\_IF\_Yes="Update Customer Set Hold = 1 WHERE ID = {ID}"

SQL\_IF\_No="Update Customer Set Hold = 0 WHERE ID = {ID}"

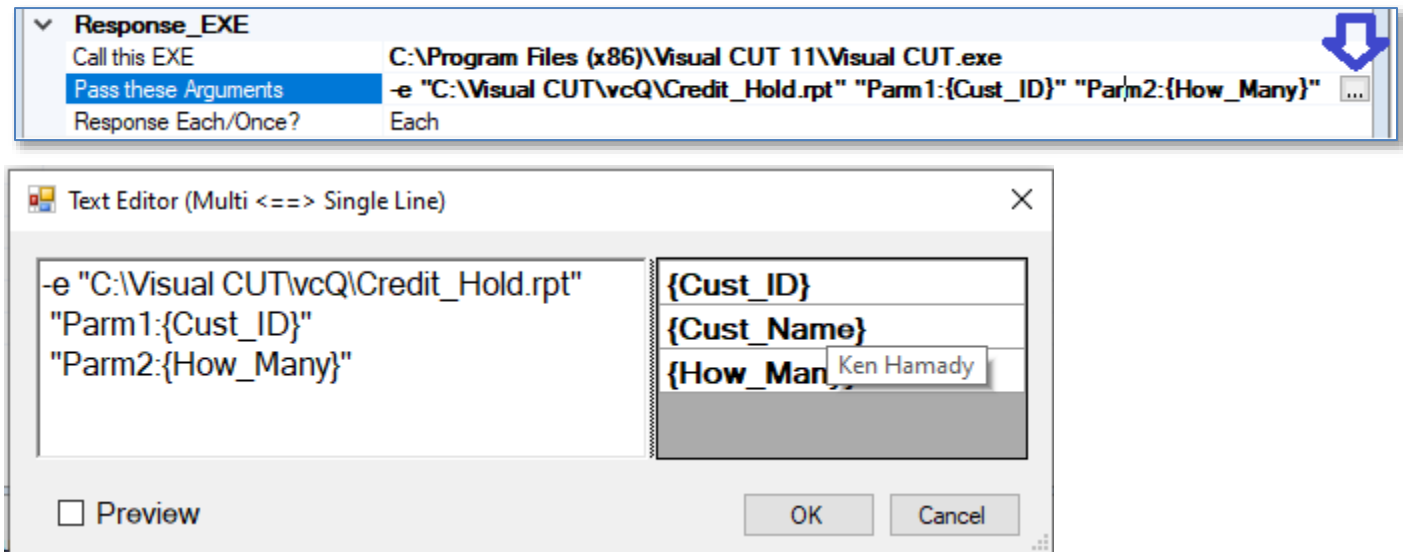
Use SELECT statement (return 0 or 1 rows!) to add dynamic tokens for EXE or Email responses.

As you select a particular event row in the grid on the left, the Property Panel on the right updates to also reflect the unique properties for that event type. In the example above, the *'Triggered By SQL'* property group reflects the unique **Trigger-Type** properties of the 'DB\_Flag' event type.

As demonstrated by the image above, when selecting a property, an ellipsis button on the right edge provides access to a matching property editor. This can be a simple drop-down providing a choice of relevant values. Or it can be a text/html editor for easy editing and embedding dynamic tokens as described in the next sections.

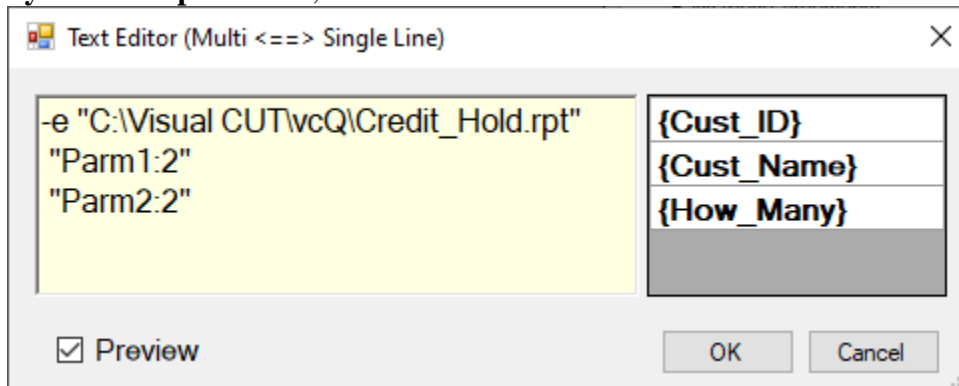
## Embed Dynamic Tokens in Response Properties

The property grid associates a special editor for response properties that may reference dynamic tokens. You open the dialog by clicking the ellipsis button to the right of the text property. For example:



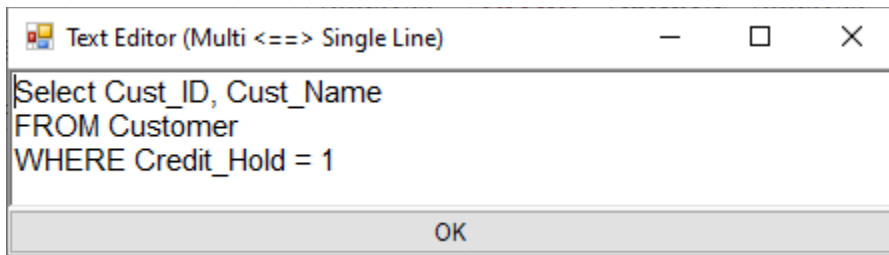
The dialog provides several benefits:

1. It breaks the text into multiple lines using typical key words (such as “Parm” “FROM” “WHERE”) to make it **easier to review and edit the text**. When the user clicks ‘OK’ the dialog re-assembles the content back to a single line.
2. Available **dynamic tokens** are listed in a panel on the right with **tooltips revealing sample values**.
3. **Double-clicking a token on the right inserts it into the cursor location within the editing panel** on the left.
4. **You can change the font size** using Ctrl-Scroll-Up or Ctrl-Scroll-Down.
5. Turning on the ‘Preview’ checkbox **toggles a display of the text with token references substituted by their sample values**, like this:



## Edit Text for Non-Response Properties

Text properties that describe the triggering event (such as source SQL statement for DB events) cannot reference dynamic tokens. Because of this, when clicking the ellipsis button, they are edited in a simple text editor that looks like this:



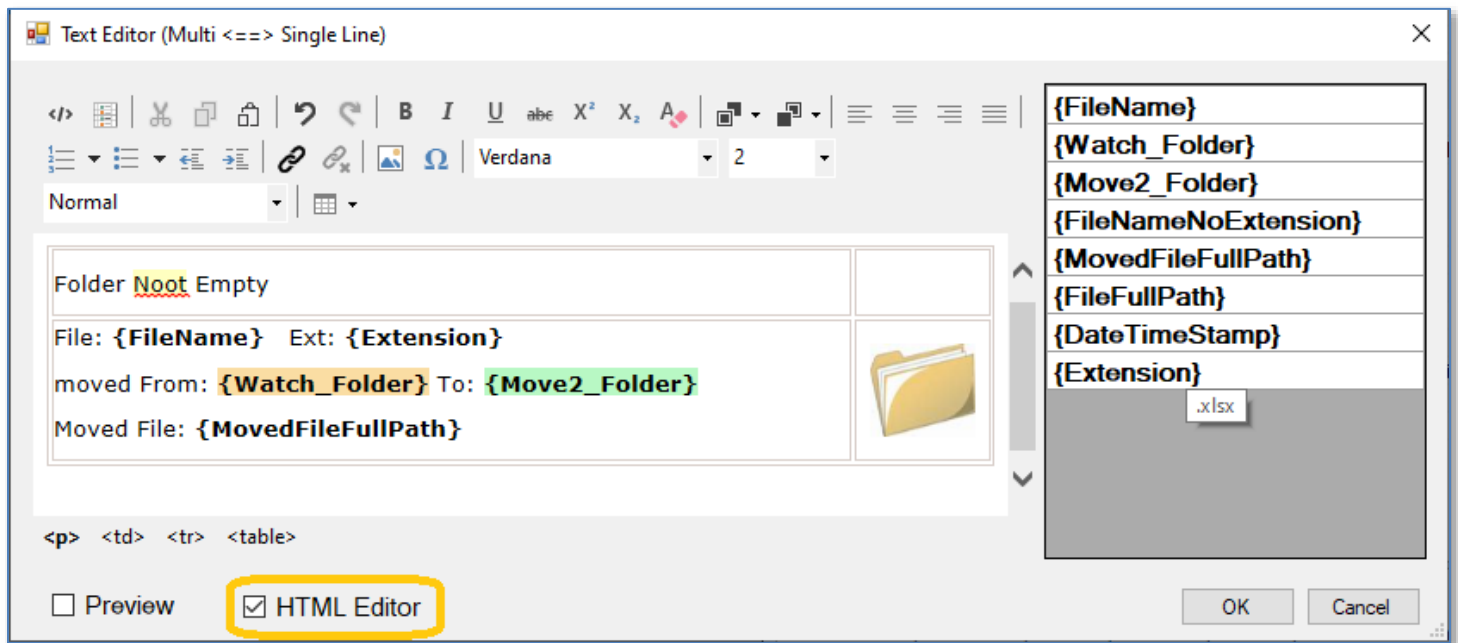
This simpler editor still provides several benefits:

1. It breaks the text into multiple lines using typical key words (such as “Parm” “FROM” “WHERE”) to make it **easier to review and edit the text**.
2. When the user clicks ‘OK’ the dialog re-assembles the content back to a single line.
3. **You can change the font size** using Ctrl-Scroll-Up or Ctrl-Scroll-Down.

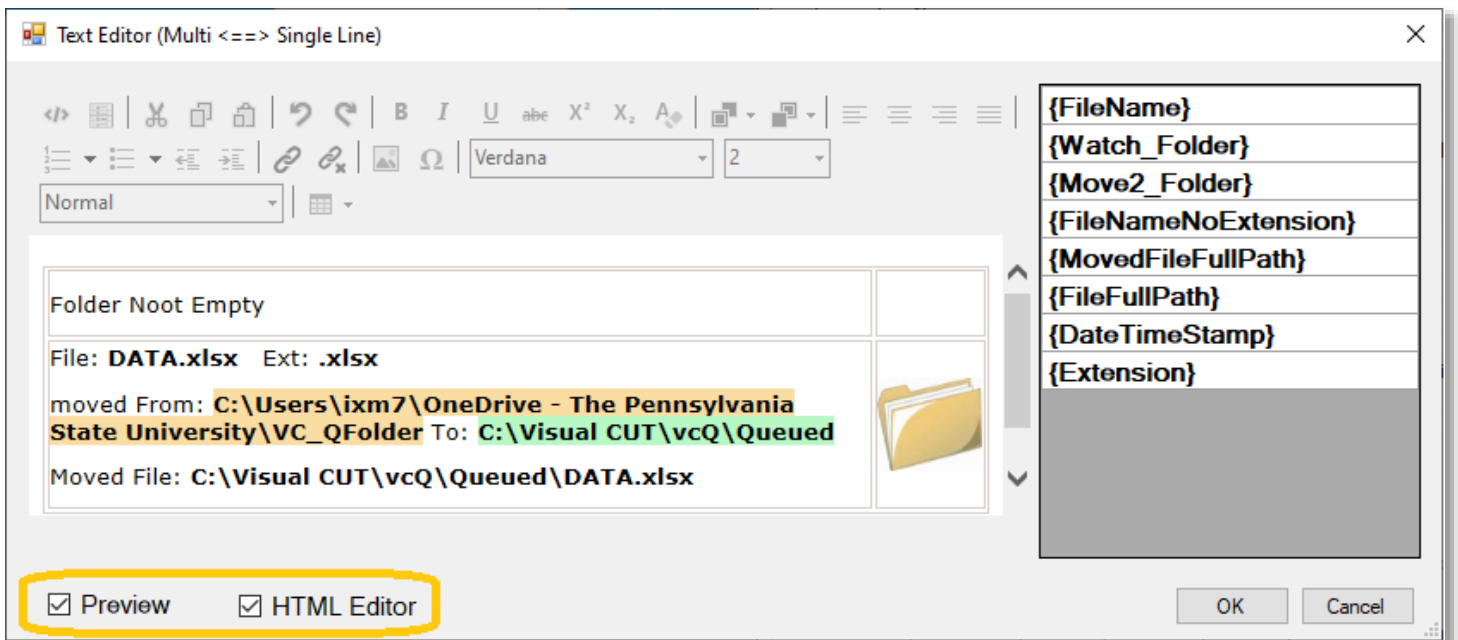
## HTML Editor for Email Message Body

When editing email message body, you can elect to activate an HTML editor as shown below. This allows you to design and preview HTML email messages.

In Windows 8 or higher, the HTML editor also provides a **spell checker** (that is why “Noot” is highlighted):

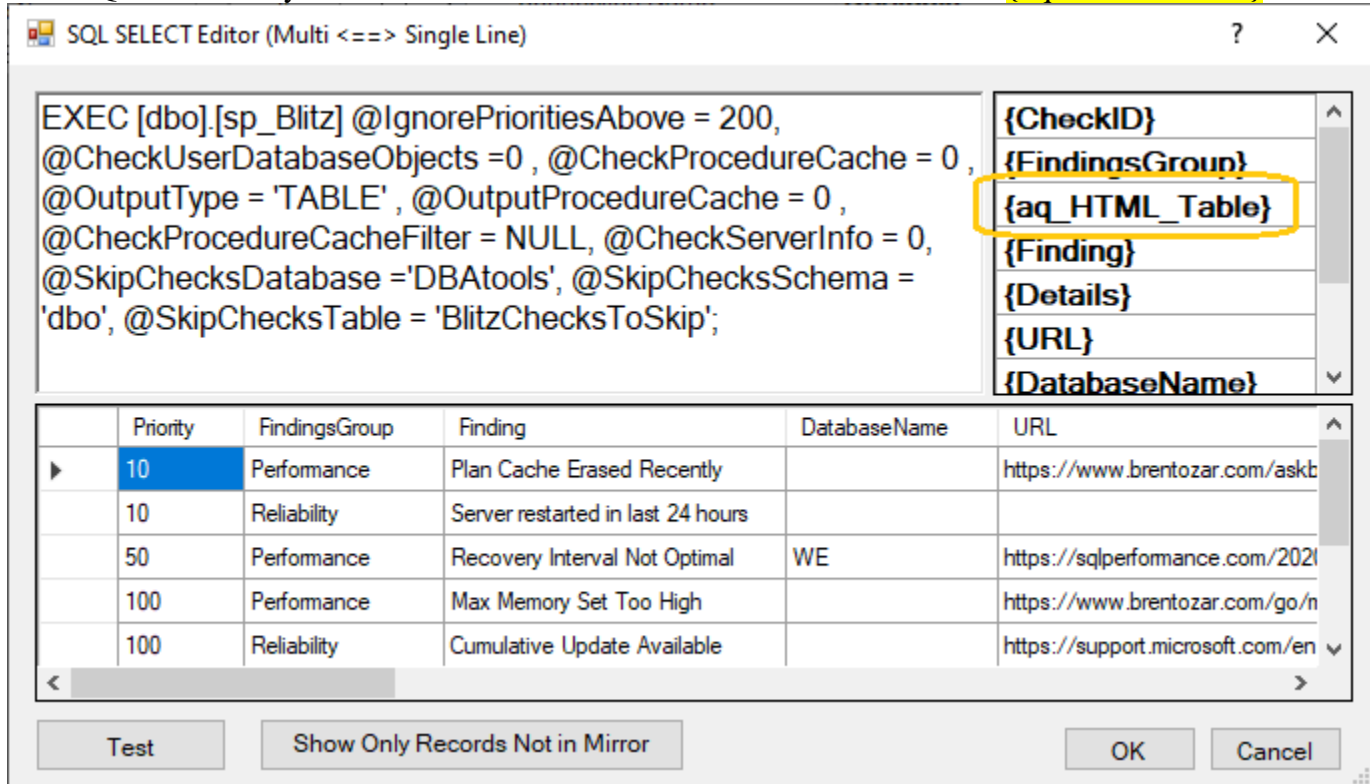


Turning on the Preview checkbox shows a Preview with dynamic token values:

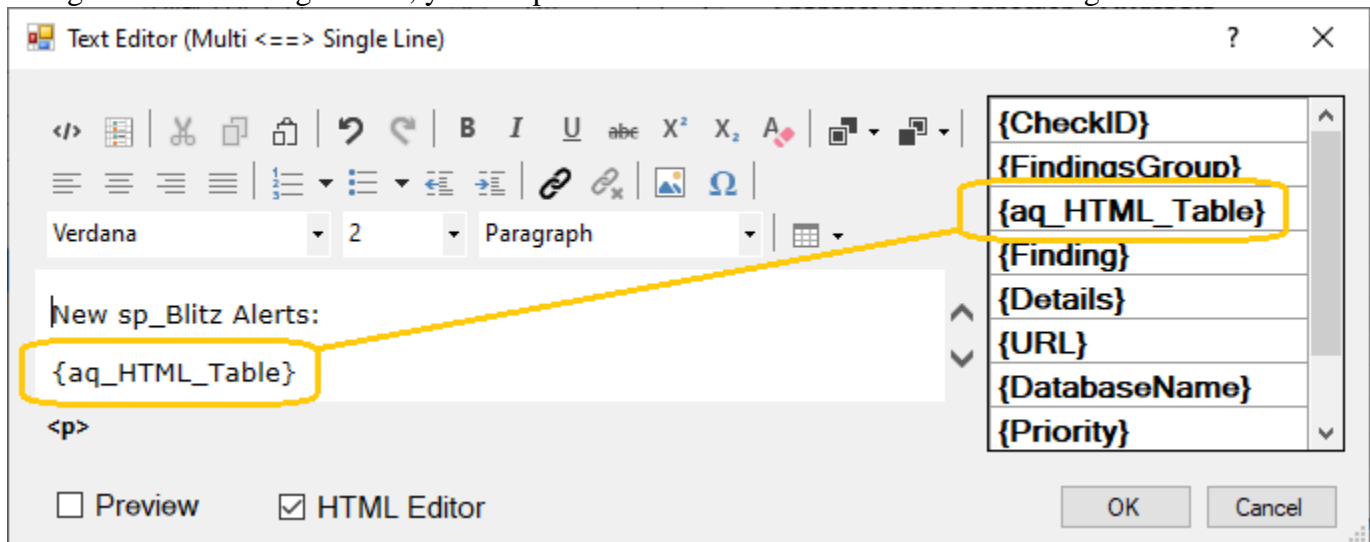


## Using {aq\_HTML\_Table} Token in Email Responses

ActionQ automatically loads event data as an HTML table into a token called {aq\_HTML\_Table}.

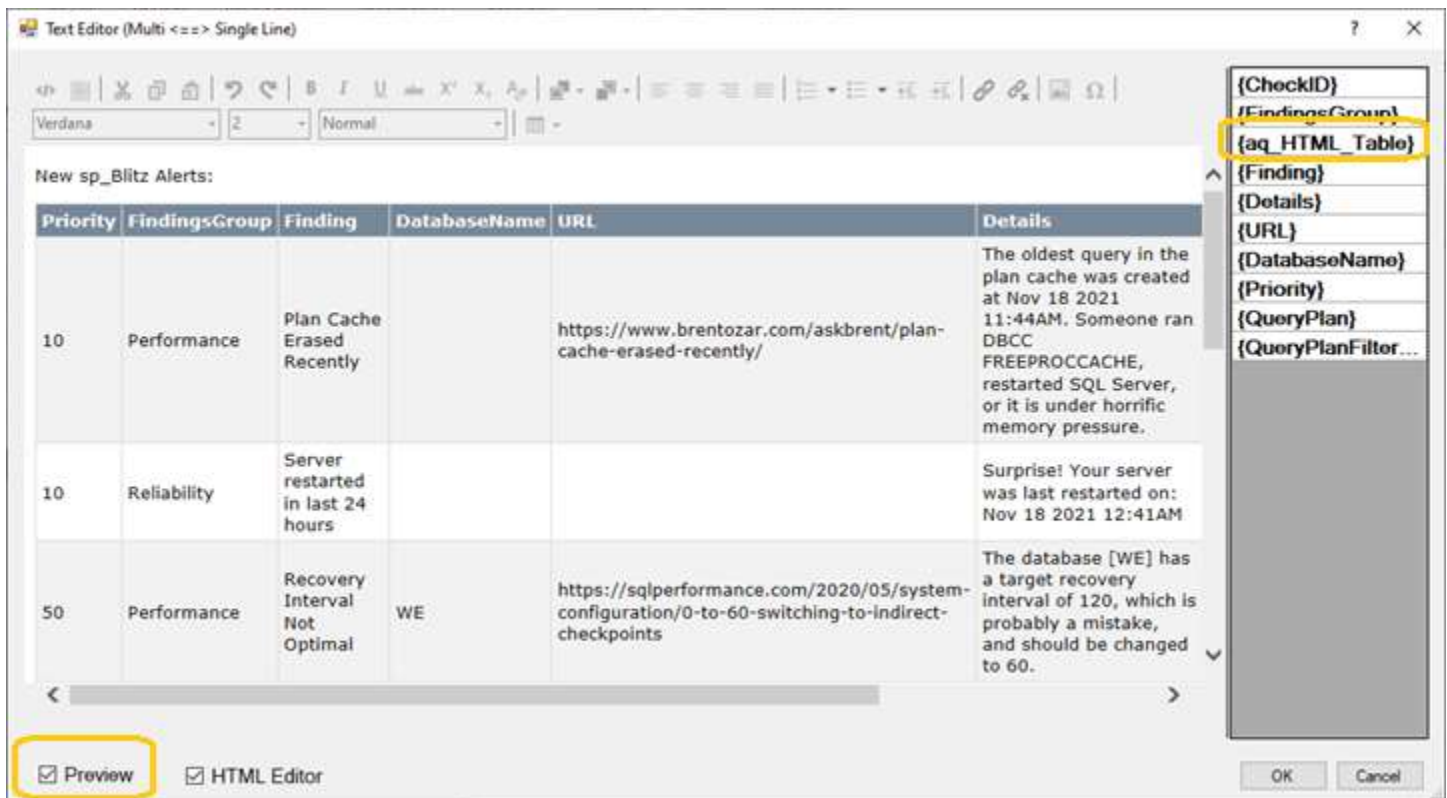


Using the email message editor, you can place that token inside HTML email messages:



Since the HTML table shows all event rows, it makes sense to use it in email responses that are designated to fire ONCE rather than for EACH event row.

Clicking the **Preview** checkbox, replaces the token with its dynamic value, displaying a nicely formatted table:



ActionQ formats the HTML table using inline CSS to ensure it renders properly in email clients such as *Gmail*:



## Using {aq\_HTML\_Table} Token in Email Responses

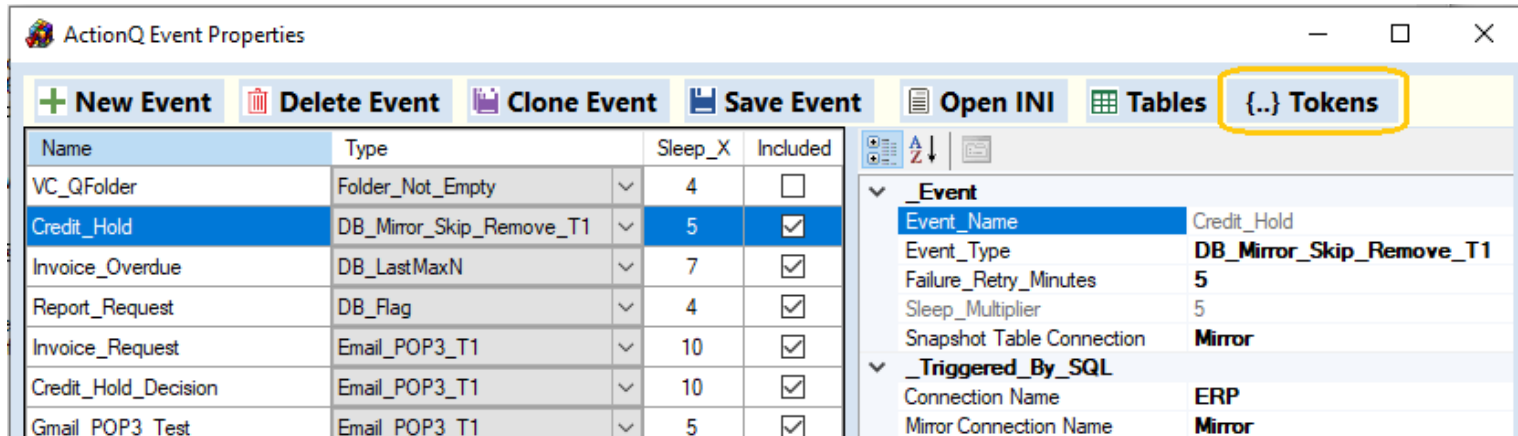
### View Dynamic Tokens for Each Process

When a process is triggered for the first time after the ActionQ service is restarted, a text file is created to document the dynamic tokens for that process.

The text file is named as: **Tokens\_<ProcessName>.txt**

It is created in the ProgramData folder (see next section).

You can open the tokens documentation file for each process by clicking the **Tokens** toolbar button:



Example for an event triggered by email:

```
{EmailDT} <=> Fri, 22 Jan 2021 14:09:06 -0500
{BodyText} <=> test
{Subject} <=> Please Send Sales Report
{ReplyTo} <=>
{FromAddress} <=> ido.millet@gmail.com
{Attachments} <=>
```

Example for an event triggered by SQL

```
{How_Many} <=> 1
{Cust_Name} <=> Dwight Wyse
{Cust_ID} <=> 1
```

Example for an event triggered by Folder event:

```
{Move2_Folder} <=> C:\Visual CUT\vcQ\Queued
{Extension} <=> .cmd
{DateTimeStamp} <=> 20210122135722
{FileNameNoExtension} <=> QFolder_Test
{Watch_Folder} <=> C:\Users\ixm7\OneDrive - MS\VC_QFolder
{FileName} <=> QFolder_Test.cmd
{FileFullPath} <=> C:\Users\ixm7\OneDrive - MS\VC_QFolder\QFolder_Test.cmd
{MovedFileFullPath} <=> C:\Visual CUT\vcQ\Queued\QFolder_Test.cmd
```

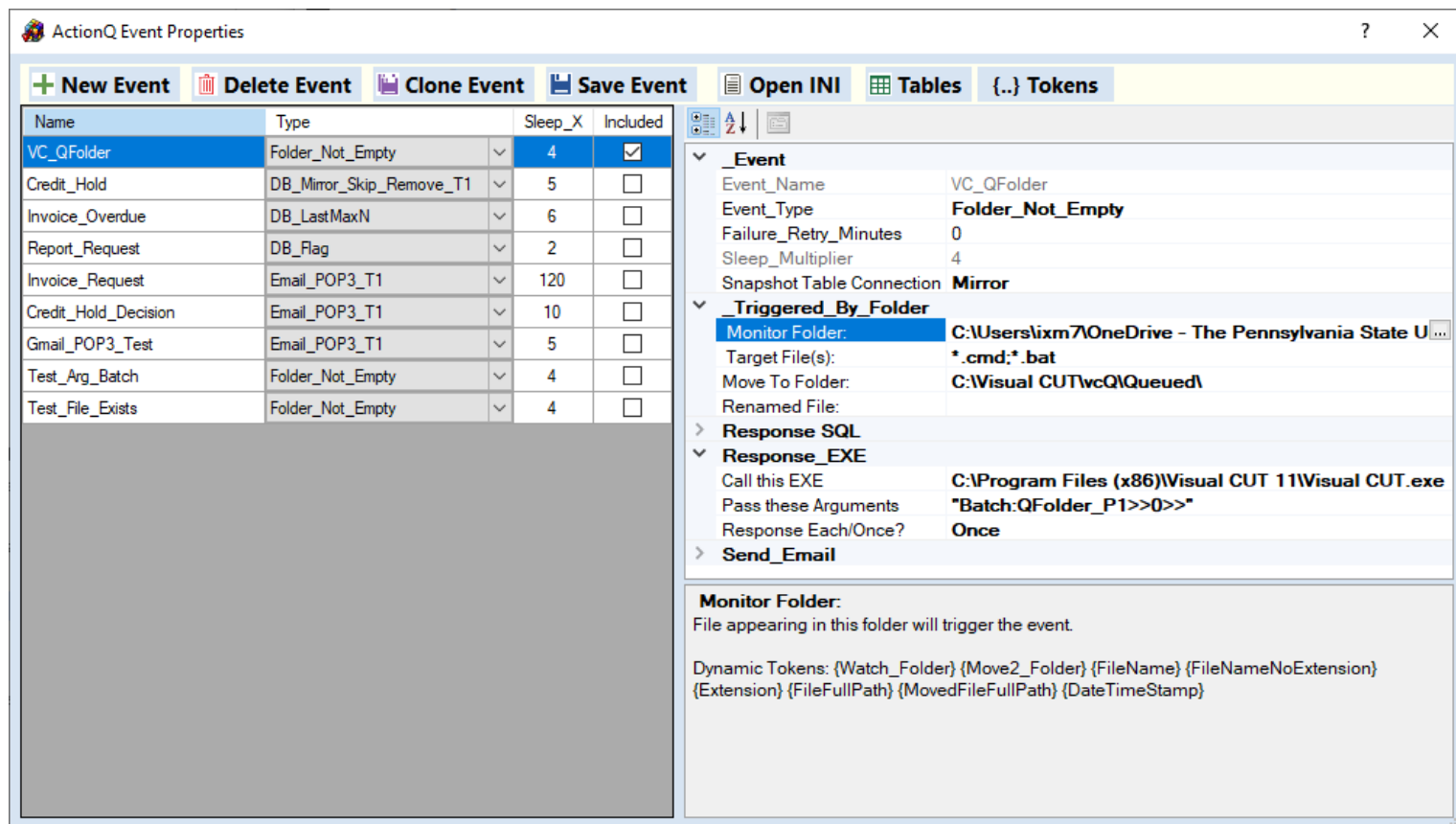
## Event Types

### "Folder\_Not\_Empty" Event Type

Here is an example of options section for this type of event:

```
[VC_QFolder]
Event_Type="Folder_Not_Empty"
Sleep_Multiplier="3"
Watch_Folder="C:\Users\ixm7\OneDrive - The Pennsylvania State University\VC_QFolder\"
Move2_Folder="C:\Visual CUT\vcQ\Queued\"
Target_Files="*.cmd;*.bat"
Renamed_File=""
SQL_After_Connection=""
SQL_After=""
EXE2Call="C:\Program Files (x86)\Visual CUT 11\Visual CUT.exe"
Arguments_Template=""Batch:QFolder_P1>>0>>>""
Call_When="Once"
```

In the example above, ActionQ will move any **.cmd** or **.bat** files in the **Watch\_Folder** to the **Move2\_Folder** and then trigger Visual CUT with a command line argument requesting processing of all batch files in that folder.



The screenshot displays the 'ActionQ Event Properties' window. On the left, a table lists various events. The 'VC\_QFolder' event is selected, showing it is of type 'Folder\_Not\_Empty' with a sleep multiplier of 4 and is included. The right pane details the configuration for this event type.

Name	Type	Sleep_X	Included
VC_QFolder	Folder_Not_Empty	4	<input checked="" type="checkbox"/>
Credit_Hold	DB_Mirror_Skip_Remove_T1	5	<input type="checkbox"/>
Invoice_Overdue	DB_LastMaxN	6	<input type="checkbox"/>
Report_Request	DB_Flag	2	<input type="checkbox"/>
Invoice_Request	Email_POP3_T1	120	<input type="checkbox"/>
Credit_Hold_Decision	Email_POP3_T1	10	<input type="checkbox"/>
Gmail_POP3_Test	Email_POP3_T1	5	<input type="checkbox"/>
Test_Arg_Batch	Folder_Not_Empty	4	<input type="checkbox"/>
Test_File_Exists	Folder_Not_Empty	4	<input type="checkbox"/>

**Event Configuration:**

- Event Name:** VC\_QFolder
- Event Type:** Folder\_Not\_Empty
- Failure\_Retry\_Minutes:** 0
- Sleep\_Multiplier:** 4
- Snapshot Table Connection:** Mirror
- Triggered By Folder:**
  - Monitor Folder:** C:\Users\ixm7\OneDrive - The Pennsylvania State University\VC\_QFolder\
  - Target File(s):** \*.cmd;\*.bat
  - Move To Folder:** C:\Visual CUT\vcQ\Queued\
  - Renamed File:**
- Response SQL:**
- Response\_EXE:**
  - Call this EXE:** C:\Program Files (x86)\Visual CUT 11\Visual CUT.exe
  - Pass these Arguments:** "Batch:QFolder\_P1>>0>>>"
  - Response Each/Once?** Once
- Send\_Email:**

**Monitor Folder:**  
File appearing in this folder will trigger the event.

**Dynamic Tokens:** {Watch\_Folder} {Move2\_Folder} {FileName} {FileNameNoExtension} {Extension} {FileFullPath} {MovedFileFullPath} {DateTimeStamp}

## Dynamic Tokens for Response Logic

This event supports embedding these tokens in the response logic (SQL, App arguments, Email):

Argument\_Template: {Watch\_Folder} {Move2\_Folder} {FileName} {FileNameNoExtension} {Extension}  
{FileFullPath} {MovedFileFullPath} {DateTimeStamp}

## Dynamic Tokens for Renaming Files

If the Mover2\_folder already contains the target file, the event logs & emails a failure message. To avoid such failures, the Renamed\_File property allows you to dynamically rename the incoming file:

1. You can inject into the Renamed\_File property tokens such as:  
{FileNameNoExtension}, {Extension}, and {DateTimeStamp}
2. You can inject current date/time with flexible formatting using {[fdt]DateTimeFormatString} tokens.

The [fdt] prefix instruct ActionQ to format the current date & time according to the formatting string. See Microsoft Date/Time formatting string documentation: <https://bit.ly/3tOW7S0>

For example, if *Renamed\_File* is: {FileNameNoExtension}\_{[fdt]yyMMdd\_HHmm}{Extension}  
an incoming Claims.csv file may be renamed to 'Claims\_210917\_0445.csv'

3. You Can add a counter, using a token such as {[V]N4}, to ensure unique file name.  
This token would be replaced by 4 digits (right-padded with 0's) and incremented to a number ensuring a unique file name.

For example, if *Renamed\_File* is: {FileNameNoExtension}\_{[V]N4}{Extension}  
an incoming Claims.csv file may be renamed to 'Claims\_0001.csv'  
If that file already exists, ActionQ will try 'Claims\_0002.csv' etc.

## Database Events

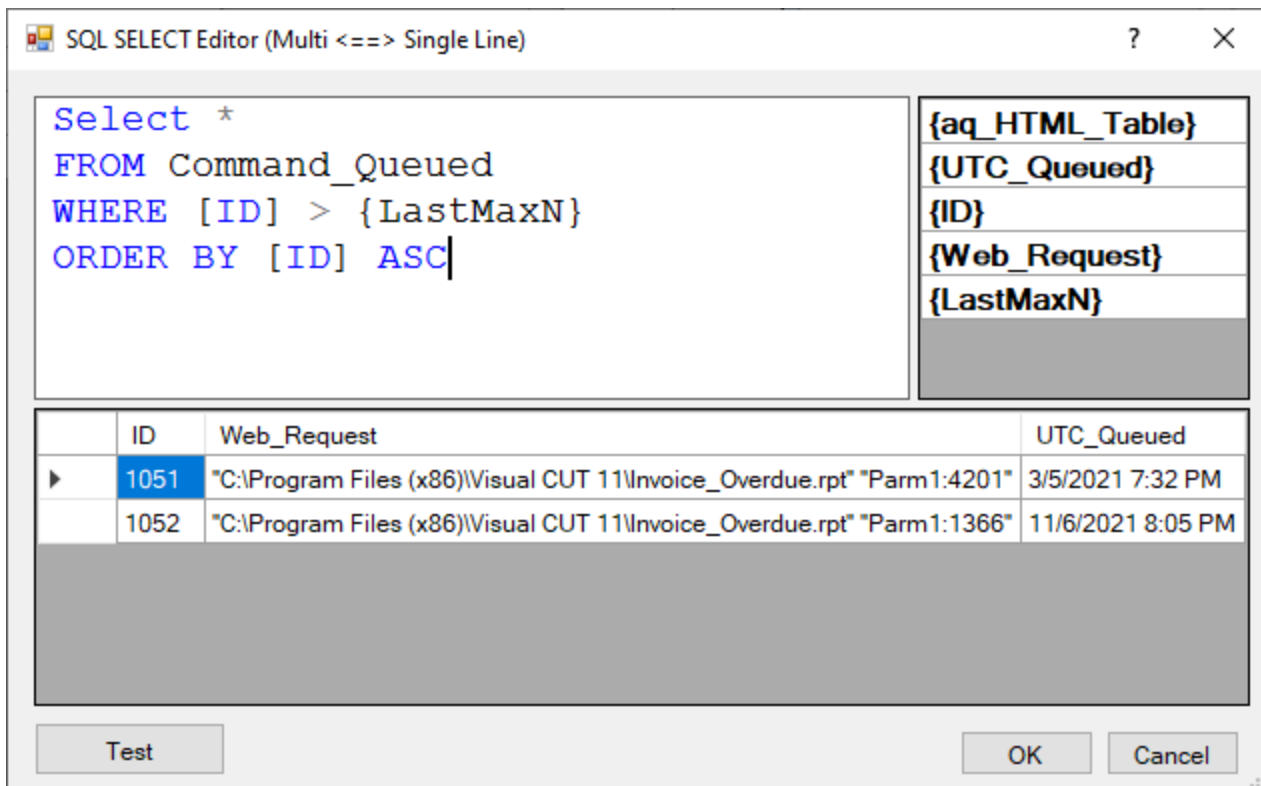
The choice between database event types depends on how you wish to avoid repeat responses:

- **DB LastMaxN** events avoid repeat responses by tracking an auto-increment key and responding only to records a key larger than the max value seen before.
- **DB Flag** events avoid repeat responses by updating the database. For example, they may set a 'Processed' column for the new record(s) to true.
- **DB Mirror Skip Remove T1** events avoid repeat responses by automatically creating & maintaining a Mirror table. If you are not allowed to touch the source database (e.g. an ERP system), you can direct the mirror table to a different database.

## Source\_SQL\_Template Editor

All database events use a SQL statement (SELECT or, for stored procedures, EXEC) to identify new events. When you click to edit the Source\_SQL\_Template property, a special editor helps you test the SQL by displaying the result set.

Upon a test, the editor also populates dynamic tokens based on the column names and values found in the first row. This facilitates constructing dynamic event responses incorporating these tokens.



SQL SELECT Editor (Multi <=> Single Line)

```
Select *  
FROM Command_Queued  
WHERE [ID] > {LastMaxN}  
ORDER BY [ID] ASC
```

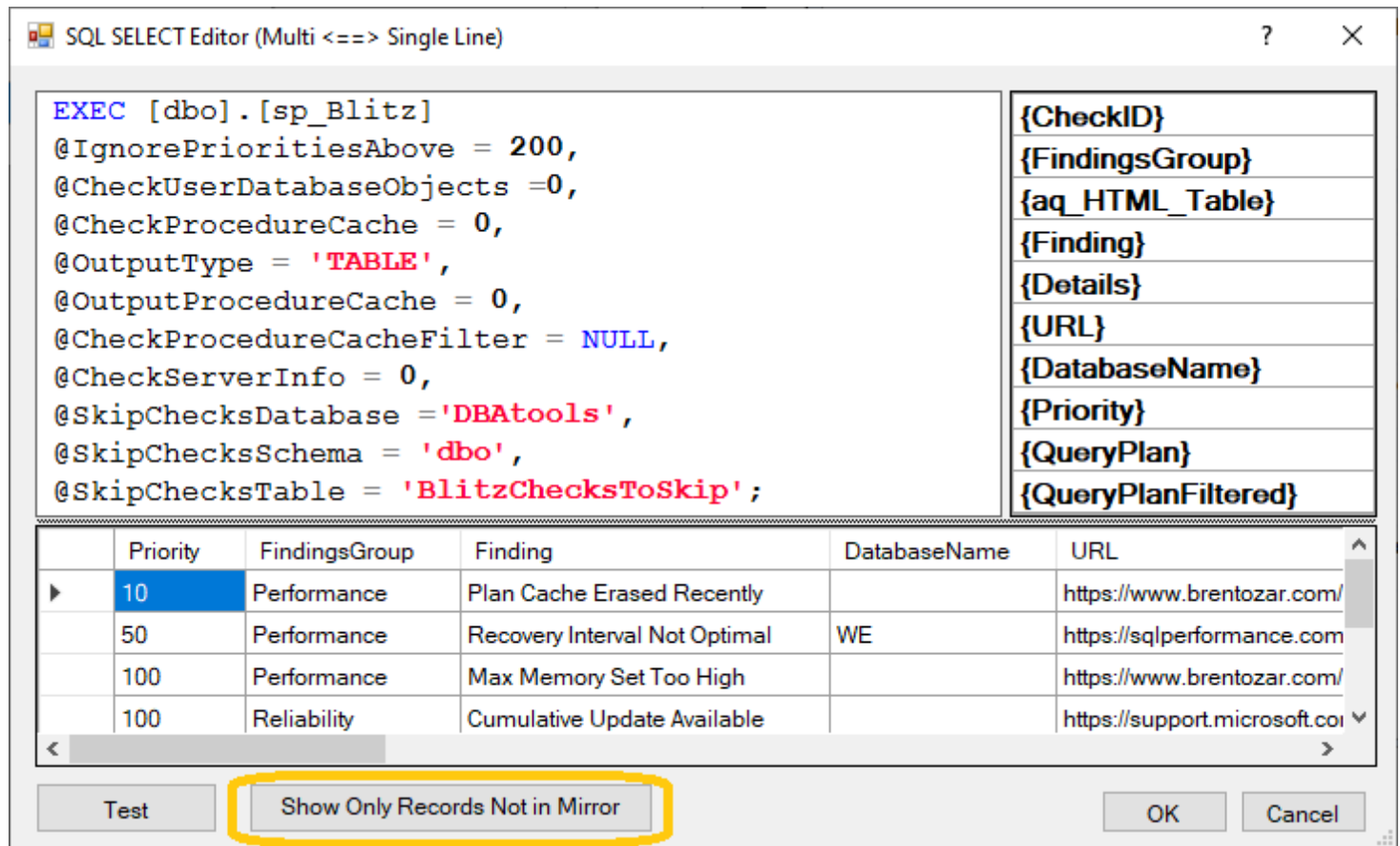
ID	Web_Request	UTC_Queued
1051	"C:\Program Files (x86)\Visual CUT 11\Invoice_Overdue.rpt" "Parm1:4201"	3/5/2021 7:32 PM
1052	"C:\Program Files (x86)\Visual CUT 11\Invoice_Overdue.rpt" "Parm1:1366"	11/6/2021 8:05 PM

Test OK Cancel

## Using a Stored Procedure as Data Source

The image below demonstrates using a Stored Procedure as the data source for an event.

This happens to be a *DB\_Mirror\_Skip\_Remove\_T1* event type. So, after populating the data, a button becomes visible, allowing you to display only rows that are not in the mirror table. See [video demo](#).



## "DB\_LastMaxN" Event Type

Here is an example of options section for this type of event:

```
[Invoice_Overdue]
Event_Type="DB_LastMaxN"
Sleep_Multiplier="6"
Source_Connection="ERP"
Source_SQL_Template="Select [Web_Request] From Command_Queued Where [ID] > {LastMaxN} ORDER
BY [ID] ASC"
SQL_Statement_Subquery=
LastMaxN="1023"
LastMaxN_ColumnName="ID"
SQL_After_Connection=""
SQL_After=""
EXE2Call="C:\Program Files (x86)\Visual CUT 11\Visual CUT.exe"
Arguments_Template="-e {Web_Request}"
Call_When="Each"
```

{LastMaxN} is a dynamic token replaced with the maximum value seen by ActionQ for the LastMaxN\_ColumnName in the query result set returned by the Source\_SQL\_Template. That maximum value is maintained for each event in memory as well as in the LastMaxN entry in the event's ini section. That is why it can be used in the SQL statement to restrict the returned rows to only new ones.

The value in any column returned by the source query, Subquery, and SQL\_After query can be used as a dynamic token in any later actions. In the example above, the argument template passed the value found in {Web\_Request} to Visual CUT for processing. **This allows any application to trigger reports and emails by simply inserting a new record into a database table.**

### Notes:

- The LastMaxN entry in ActionQ.ini is read and used only for initialization purposes. Once a new LastMaxN value is established, it is written to ActionQ\_Data.ini file. From that point on, it is read and managed by the service only at that location. That ensures the ActionQ service never attempts to write to the ActionQ.ini file (which might be open for editing by you at the same time). So:  
ActionQ.ini is for you to change settings  
ActionQ\_Data.ini is for the service to persist values in case the service gets stopped.
- You can use any column names in the SQL result set as dynamic tokens.
- Source\_Connection refers to a connection string entry in the [Connection\_Strings] ini section. To protect passwords, those entries can refer to encrypted strings as explained in [Managing Encrypted Strings](#).

## "DB\_Flag" Event Type

Here is an example of options section for this type of event:

```
[Report_Request]
Event_Type="DB_Flag"
Sleep_Multiplier="4"
Source_Connection="ERP"
Source_SQL_Template="Select * From Command_Queued_Flag Where [Processed] = 0"
SQL_Statement_Subquery=
SQL_After_Connection="ERP"
SQL_After="Update Command_Queued_Flag Set [Processed]=1 WHERE ID = {ID}"
EXE2Call="C:\Program Files (x86)\Visual CUT 11\Visual CUT.exe"
Arguments_Template="-e {Report} {Arguments}"
Call_When="Each"
```

**SQL\_After** is executed for each row in the result set. In this case, it uses a token of one of the columns in the result set (**{ID}**) to update a Status to 'Processed.' This avoids duplicate processing. In a similar way, it uses two other columns from the source query (**{Report}** and **{Arguments}**) to pass dynamic arguments to the called executable.

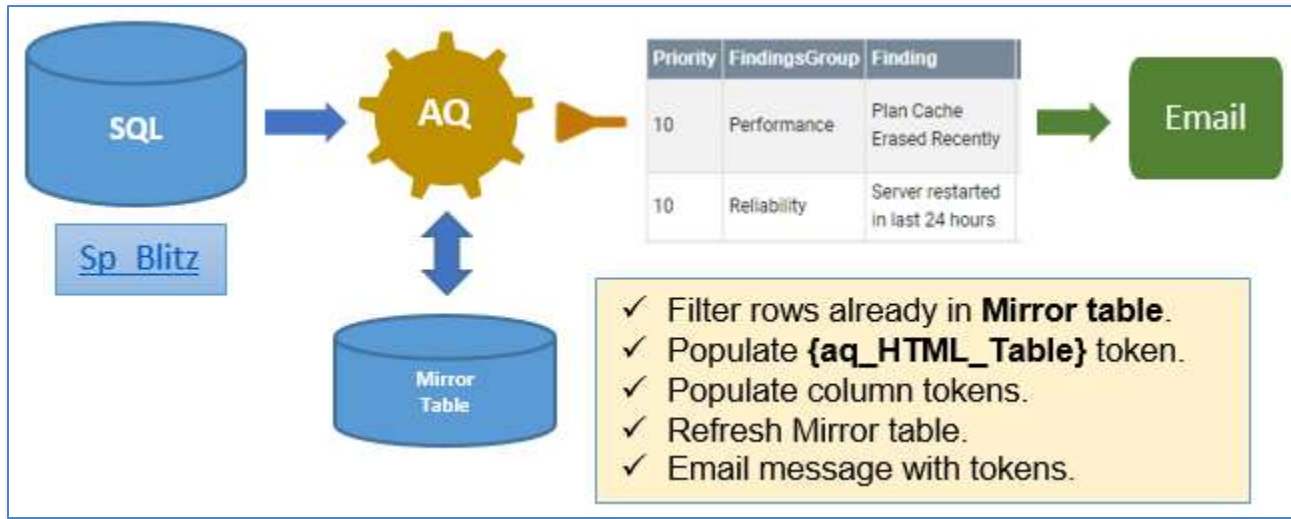
Notes:

- You can use any column names in the SQL result sets as dynamic tokens.
- **Source\_Connection** and **SQL\_After\_Connection** refer to a connection string entry in the [Connection\_Strings] ini section. To protect passwords, those entries can refer to encrypted strings as explained in [Managing Encrypted Strings](#).

## "DB\_Mirror\_Skip\_Remove\_T1" Event Type

[Video demo](#) of using *ActionQ* to monitor the health of a SQL Server using this type of event.

This diagram shows the general logic:



Here is an example of options section for this type of event:

```
[Credit_Hold]
Event_Type="DB_Mirror_Skip_Remove_T1"
Mirror_Connection="SQLExpress_Mirror"
; mirror table always named as 'aqm_<Event_Name>'. so for this event: aqm_Credit_Hold).
Sleep_Multiplier="4"
Source_Connection="ERP"
Source_SQL_Template="Select Cust_ID From Customer WHERE Credit_Hold = 1"
SQL_Statement_Subquery="Select Cust_ID,Date,Amount,Balance_After FROM Payments
WHERE Cust_ID = {Cust_ID}"
EXE2Call="C:\Program Files (x86)\Visual CUT 11\Visual CUT.exe"
Arguments_Template="-e 'C:\Visual CUT\vcQ\Credit_Hold.rpt' 'Parm1:{Cust_ID}'"
Call_When="Each"
Failure_Retry_Minutes="5"
```

This type of event avoids duplicate processing by maintaining a mirror table with rows retrieved last time by the source Query. The mirror table is created automatically by ActionQ based on the column names and data types in the source query. It is named as **aqm\_<Event\_Name>**.

so for the event above the mirror table name is **aqm\_Credit\_Hold**.

In the source query, any row that matches an existing row in the Mirror table gets **skipped**.

**Column names starting with a space are not included in this logic.** That allows you to include extra data for event responses, but treat only some columns as keys in determining if a new event is a duplicate.

In the Mirror table, any row that is no longer in the source query gets **removed**. For example, if the customer is no longer on credit hold, removing that row from the Mirror table allows ActionQ to recognize a case when the same customer is placed on a new credit Hold.

This event type is particularly useful for cases where the source SQL comes from an ERP or Cloud database where you have no Modify permissions. Setting the **Mirror\_Connection** to another database where you have full permissions, allows you to maintain the Mirror table in the database of your choice.

Besides using column names that start with a space to indicate they are not key columns for detecting new events, another option is to include only key columns that identify true duplicates. For example, in the case above, the source query returns just the Cust\_ID column. The **SQL\_Statement\_Subquery** is used to access more database details. As demonstrated above, the **SQL\_Statement\_Subquery** can be **correlated** to the main query or the main event (File or email) by referencing tokens (e.g. {Cust\_ID}) from the main event.

The **SQL\_Statement\_Subquery** can also be used in cases where the event data has **one to many relationship to desired database information**. For example, in the case above, the database event identifies customers placed on hold. But for each such customer you wish to email a manager an HTML table containing that customer's payment and balance history.

#### Notes:

- ActionQ facilitates using any property name in the result set as a dynamic token in responses.
- **Source\_Connection** and **Mirror\_Connection** refer to a connection string entry in the [Connection\_Strings] ini section. To protect passwords, those entries can refer to encrypted strings as explained in [Managing Encrypted Strings](#).
- **Failure\_Retry\_Minutes** is an optional setting available to all processes. See [Setting Failure Retry Period](#).
- **Column names starting with a space** are not included in detecting new events by comparing to the mirror table. You can think of them as non-keys. To save space, increase speed, and improve security, only key columns are stored in the mirror table. This also improves resiliency because you can add/remove/change non-key columns to the event design without breaking the mirror table logic.

## PowerShell "PS\_Mirror\_Skip\_Remove\_T1" Event Type

See [video demo](#) of **PowerShell scripts as events & responses**.

This is very similar to DB\_Mirror\_Skip\_Remove\_T1 except that the source event is a PowerShell script rather than an SQL statement. The PowerShell script (.ps1 file) returns a collection of PowerShell objects as a result. ActionQ parses that result into a data table, similar to how it parses the data returned by an SQL event.

### Calling REST APIs such as RabbitMQ

PowerShell is very powerful. You can leverage this event type to get data from many types of data sources. For example, you can call any **REST API** and not only return data but also "shape" (filter columns/rows, rename columns, add columns). For a good introduction to how you can call REST APIs from PowerShell see [this blog](#). For discussion of how you can call **RabbitMQ** via PowerShell, see [this blog](#).

### Typical Properties

Here is an example of options section for this type of event:

```
[Disk_Full]
Event_Type="PS_Mirror_Skip_Remove_T1"
PS2Call="C:\PS\Disks_Getting_Full.ps1"
Mirror_Connection="Mirror"
Sleep_Multiplier="5"
EXE2Call="PowerShell.exe" ; call any EXE, batch file, REST API, or PowerShell as in this example
Arguments_Template= "-NoProfile -ExecutionPolicy Bypass -Command 'C:\PS\my.ps1 -log {Name}.txt';"
Call_When="Each"
Failure_Retry_Minutes="0"
SQL_After="Insert into [dbo].[ActionQ_PowerShell_Log] values (GetDate(), 'Low Disk Space',
'{DeviceID}{ VolumeName}', Size: {Size(GB)}GB, { FreeSpace(%)})% free.');"
SQL_When="Each"
SQL_After_Connection="ERP"
SMTP_Settings="Email_From_IT"
Email_To="ido.millet@gmail.com"
Email_Subject="Disk Full Alert "
Email_Message="<HTML> ...disks are getting full<p>{ps_HTML_Table}</p> ... </HTML>"
Email_When="Once"
```

## Avoiding Duplicate Processing



This type of event avoids duplicate processing by maintaining a mirror table with rows of results returned last time by the triggering PowerShell script. The mirror table is created automatically by ActionQ based on the properties of the PowerShell objects collection returned by the script. It is named as **aqm\_<Event\_Name>**. so for the event above the mirror table name is **aqm\_Disk\_Full**.

In the source query, any row that matches an existing row in the Mirror table gets **skipped**.

**Column names starting with a space are not included in this logic.** That allows you to include extra data for event responses, but treat only some columns as keys in determining if a new event is a duplicate.

In the Mirror table, any row that is no longer in the source query gets **removed**. For example, if the disk is no longer full, removing that row from the Mirror table allows ActionQ to recognize a case when the same disk is again getting full.

Note: to allow ActionQ to parse the results of the PowerShell script, let the script return PowerShell's default objects. Do NOT pipe the results into text tables for display.

Notes:

- ActionQ facilitates using any property name in the result set as a dynamic token in responses.
- **Mirror\_Connection** refers to a connection string entry in the [Connection\_Strings] ini section. To protect passwords, those entries can refer to encrypted strings as explained in [Managing Encrypted Strings](#).
- **Failure\_Retry\_Minutes** is an optional setting available to all processes. See [Setting Failure Retry Period](#).
- **Column names starting with a space** are not included in detecting new events by comparing to the mirror table. You can think of them as non-keys. To save space, increase speed, and improve security, only key columns are stored in the mirror table. This also improves resiliency because you can add/remove/change non-key columns to the event design without breaking the mirror table logic.

## "Email\_POP3\_T1" Event Type

This event type responds to incoming emails by triggering dynamic processes or database updates. It can detect and use text tokens within the email subject/body/attachments to trigger different actions.

### Typical Use Case:

Imagine a customer was placed on credit hold (Credit\_Hold column set to True in the database):

Cust_ID	Cust_Name	Credit_Hold	Credit_Hold_Reversed_By	Credit_Hold_Approved_By	Credit_Hold_Decision_DT
1	Dwight Wyse	True		NULL	NULL

A Database event in *ActionQ* detects this and uses *Visual CUT* to email the VP of Sales:



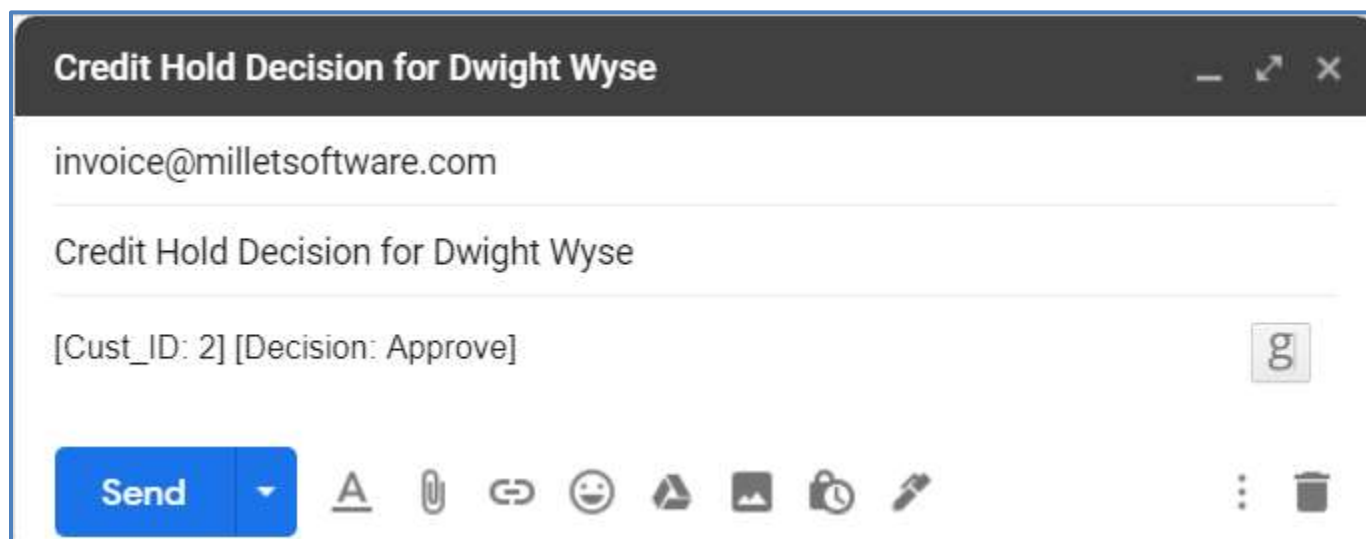
The manager can then manually reply and add the text *Reverse* or *Approve* in a designated text area. Or, better yet, the manager can simply click one of the decision buttons. Those buttons are just images with mailto hyperlinks.

You can use a web site such as [this](#) to generate the **button images** and [this](#) to generate **mailto** syntax as hyperlinks for these images.

For example:

```
<A href="mailto:invoice@milletsoftware.com?subject=Credit Hold Decision for {Customer.Cust_Name}&body=[Cust_ID: 2] [Decision: Reverse]">  
<IMG src="file:///C:/Visual CUT/vcQ/Reverse.png"></A>
```

Clicking on the **Reverse** button triggers the following email message dialog.  
The manager can then simply click **Send**:



Another event in ActionQ (of type **Email\_POP3\_T1**) then:

1. Detects that incoming email
2. Collects data in tokens such as [Cust\_ID: **1**] and [Decision: **Approve**].
3. Responds by updating the database, so the customer record changes to:

Cust_ID	Cust_Name	Credit_Hold	Credit_Hold_Reversed_By	Credit_Hold_Approved_By	Credit_Hold_Decision_DT
<b>1</b>	Dwight Wyse	True		ido@milletsoftware.com	2020-08-25 19:04:04.0000000

If, for another case, the manager indicates **Reverse**, ActionQ would issue a **different SQL statement** resulting in a customer record shown for Ken Hamady:

Cust_ID	Cust_Name	Credit_Hold	Credit_Hold_Reversed_By	Credit_Hold_Approved_By	Credit_Hold_Decision_DT
<b>1</b>	Dwight Wyse	True		ido@milletsoftware.com	2020-08-25 19:04:04.0000000
<b>2</b>	Ken Hamady	False	ido@milletsoftware.com	NULL	2020-08-25 14:40:55.0000000

This demonstrates the ability to **automate workflows by chaining multiple events**.

## Setting Default Email Client

When a user clicks a mailto link, if the wrong email client opens up (e.g. Outlook instead of Gmail), the user can change their default email client.

In Windows, head to Settings > Apps > Default apps. Scroll down and pick **Choose default apps by protocol**. For 'Mailto', choose the client of your choice. For Gmil, select Google Chrome.

## Email Server and Inbox Settings

Multiple events can target the same inbox. Server/Inbox settings are specified like this:

```
[Email_Server_InBoxes]
MilletSoftware_Invoice=host4.hostmonster.com||invoice@MilletSoftware.com||ES_MS_Password||10
```

The 4 elements are: the server, the inbox email account, the encrypted password name, and the maximum number of emails to inspect in each scan cycle.

## Event Settings

Each **Email\_POP3\_T1** event has a section with settings such as these:

```
[Credit_Hold_Decision]
Event_Type="Email_POP3_T1"
Email_Server_InBox="MilletSoftware_Invoice"
Sleep_Multiplier="10"
Filter="(Subject contains "Decision Needed: Credit Hold" AND From contains "ido@milletsoftware.com")"
Message_Sent_in_Last_N_Minutes="9000"
Save_As_EML_To_Folder="C:\Visual CUT\Invoice_Requests\eml\"
Save_Attachments_To_Folder=""
Save_Attachments_To_Unique_File_Names="true"
Delete_Email_From_Server="true"
Parse_Expressions="Cust_ID:::\d{5,6}||Cust_Email:::\S+@\S+\.[a-zA-Z0-9]+"
SQL_After_Connection="ERP"
SQL_After="SQL_IF_{Decision}"
SQL_IF_Reverse="Update Customer Set Credit_Hold = 0, Credit_Hold_Reversed_By='{FromAddress}',
Credit_Hold_Decision_DT=GETDATE() WHERE Cust_ID = {Cust_ID}"
SQL_IF_Approve="Update Customer Set Credit_Hold_Approved_By='{FromAddress}',
Credit_Hold_Decision_DT=GETDATE() WHERE Cust_ID = {Cust_ID}"
Call_When="Each"
EXE2Call=""
Arguments_Template=""
```

The **Email\_Server\_InBox** entry points at the corresponding server and inbox this event monitors.

The **Filter** entry specifies what email messages within that inbox are targeted. Here are some examples:

Subject **like** "Re: Credit Hold\*"

Subject **contains** "Trip Cancellation" and From = "Cruises@Titanic.com"

Any email header field name can be used, case is insensitive.

The "\*" wildcard matches 0 or more occurrences of any character.

Parentheses can be used to group conditions.

The logical operators are: AND, OR, NOT (case insensitive)

Comparison operators are: =, <, >, <=, >=, <>

String comparison operators are: CONTAINS, LIKE (case insensitive)

*Message\_Sent\_in\_Last\_N\_Minutes* entry allows only newer messages to be targeted.

*Save\_As\_EML\_To\_Folder* entry, if populated, directs the process to download the messages as eml files.

*Save\_Attachments\_To\_Folder* and *Save\_Attachments\_To\_Unique\_File\_Names* entries control if, how, and to what folder attachments are downloaded. If attachments are downloaded, ActionQ maintains an **{Attachments}** token with the list of downloaded files separated by '|'|.

*Delete\_Email\_From\_Server* entry controls whether ActionQ deletes a targeted email message from the server after processing it. **Important:** if this option is set to false, ActionQ avoids duplicate processing of old messages by tracking the maximum date & time found in the date header of already-processed messages.

Here is a screenshot showing the properties in the GUI:

The screenshot shows the 'ActionQ Event Properties' window. It has a menu bar with options: + New Event, Delete Event, Clone Event, Save Event, Open INI, Tables, and {...} Tokens. Below the menu is a table of events:

Name	Type	Sleep_X	Included
VC_QFolder	Folder_Not_Empty	4	<input type="checkbox"/>
Credit_Hold	DB_Mirror_Skip_Remove_T1	5	<input type="checkbox"/>
Invoice_Overdue	DB_LastMaxN	6	<input type="checkbox"/>
Report_Request	DB_Flag	2	<input type="checkbox"/>
Invoice_Request	Email_POP3_T1	10	<input type="checkbox"/>
Credit_Hold_Decision	Email_POP3_T1	10	<input type="checkbox"/>
Gmail_POP3_Test	Email_POP3_T1	5	<input type="checkbox"/>
Test_Arg_Batch	Folder_Not_Empty	4	<input type="checkbox"/>
Test_File_Exists	Folder_Not_Empty	4	<input type="checkbox"/>
spBlitz_New_Alert	DB_Mirror_Skip_Remove_T1	5	<input type="checkbox"/>
Export_Open	Folder_Not_Empty	1	<input type="checkbox"/>
Parse_Email_Attachment	Email_POP3_T1	5	<input checked="" type="checkbox"/>
OneDrive_Move_test	Folder_Not_Empty	4	<input type="checkbox"/>
Invoice_Request_No_DB_...	Email_POP3_T1	10	<input type="checkbox"/>
Gmail_to_Twilio	Email_POP3_T1	20	<input checked="" type="checkbox"/>

The right pane shows the configuration for the selected 'Parse\_Email\_Attachment' event:

- \_Event**
  - Event\_Name: Parse\_Email\_Attachment
  - Event\_Type: Email\_POP3\_T1
  - Failure\_Retry\_Minute: 10
  - Sleep\_Multiplier: 5
  - Snapshot Table Conn:
- \_Triggered\_By\_Email**
  - InBox Name: MilletSoftware\_Invoice
  - Delete Email From Se: True
  - Filter Condition: (Subject Contains "Get Attachment Data")
  - Filter Last N Minutes: 9000
  - Parse Expressions: Cust\_ID::\d{5,6} || Cust\_Email::\S+@\S+\.[a-zA-Z0-9]+\
  - Save .eml Folder: C:\Visual CUT\Invoice\_Requests\eml\
  - Save Attachments Fol: C:\Visual CUT\Invoice\_Requests\Attachments\
  - Save Attachments Un: True
- Response SQL**
- Response\_EXE\_or\_REST\_API**
- Send\_Email**

**Parse Expressions**

Tokens by RegEx: extract tokens from Subject, Message Body, and Attachments. each directive is a pair of 'Token\_Name::Regular\_Expression' Separate multiple pairs with '|' like this:  
Cust\_ID::\d{5,6} || Cust\_Email::\S+@\S+\.[a-zA-Z0-9]+\

The 'Cust\_ID' token looks for a 5-to-6 digits number.  
The 'Cust\_Email' token looks for an email address.

Token by Brackets: [Name: Value] patterns are also parsed.  
For example: [Decision: Approve]

ActionQ parses dynamic tokens from each email message using two mechanisms:

1. By Brackets (e.g. **[Name: Value]** patterns located in message **subject** and **body**)
2. By Regular Expressions matching text in message **subject**, **body**, and **attachments**.

## Standard Dynamic Tokens

For email events, ActionQ populates these standard dynamic tokens:

**{EmailDT}**, **{FromAddress}**, **{ReplyTo}**, **{Attachments}**, **{Subject}**, and **{BodyText}**

which can be referenced inside event responses (SQL statements, Command line arguments, Email messages):

## Dynamic Tokens By Regular Expressions

Regular expressions allow you to extract values from text using very powerful and flexible expressions.

You can specify multiple patterns using the Parse Expressions property of email events.

For example:

Parse\_Expressions="**Cust\_ID::\d{5,6}**||**Cust\_Email::\S+@\S+\.[a-zA-Z0-9]+\S+**"

tells ActionQ to parse the email message **subject**, **body**, as well as any **attachments** (with a file type that can be treated as text).

If a pattern with 5-to-6 digits is found, it gets loaded into a **{Cust\_ID}** token.

If a pattern that looks like an email address is found, it gets loaded into a **{Cust\_Email}** token.

This use of such powerful patterns adds a lot of flexibility in automating responses to email messages based on message as well as attachments text.

## Dynamic Tokens By Brackets

In addition, ActionQ automatically scans the **subject** and **body** for tokens that look like this:

**[anyName: anyValue]** and loads them into tokens.

For example, having the following text **[Decision: Approve]** anywhere within the **subject** or **body** of the email would result in a **{Decision}** token with a value of **'Approve'**.

## SQL\_After Logic

You can use the value of a dynamic token to control the SQL statement generated in response to the event.

For example, setting this event property: **SQL\_After="SQL\_IF\_{Decision}"**

indicates that the SQL statement to be triggered depends on the value of the **{Decision}** token.

If the Decision value is **Reverse**, the statement specified by **SQL\_IF\_Reverse** is executed.

If the Decision value is **Approve**, the statement specified by **SQL\_IF\_Approve** is executed.

You can use any token(s) and any number of alternatives with this logic.

If no matching ini entry is found, the SQL step is skipped.

In the example above, the EXE2Call is left blank, but you are free to trigger just SQL, just EXE, both, or even none (for example, if you wish to use the process just for downloading targeted email attachments).

## Setting Up Alerts for Windows Scheduled Task Failures

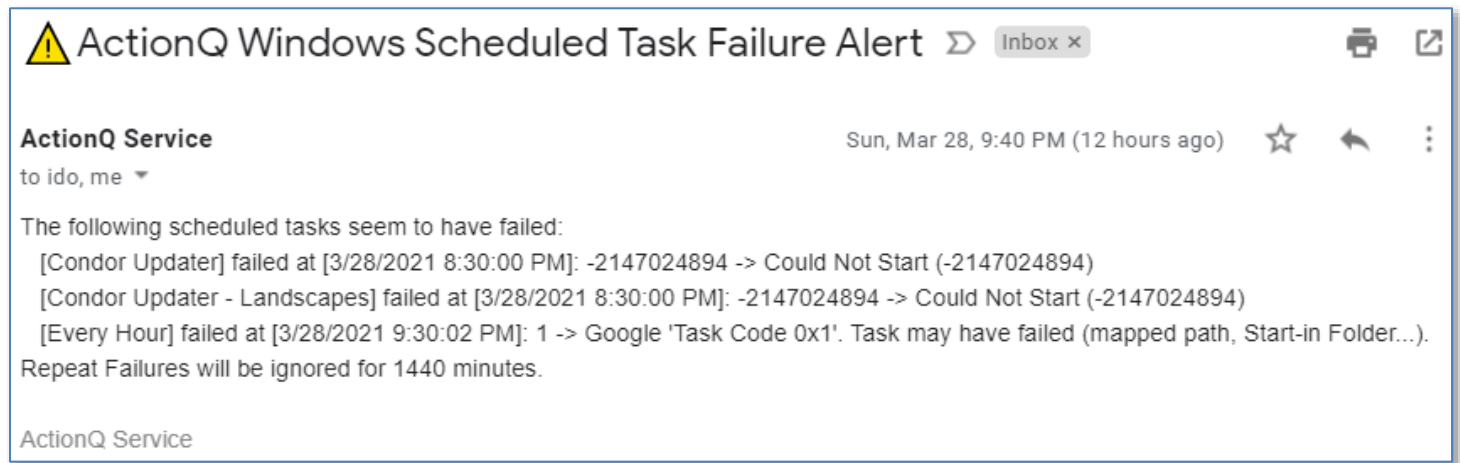
ActionQ can alert you when *Windows Task Scheduler* fails a task. For example, when a scheduled task designed to trigger a batch file fails to locate the batch file.

Simply add the following section to the ActionQ.ini file:

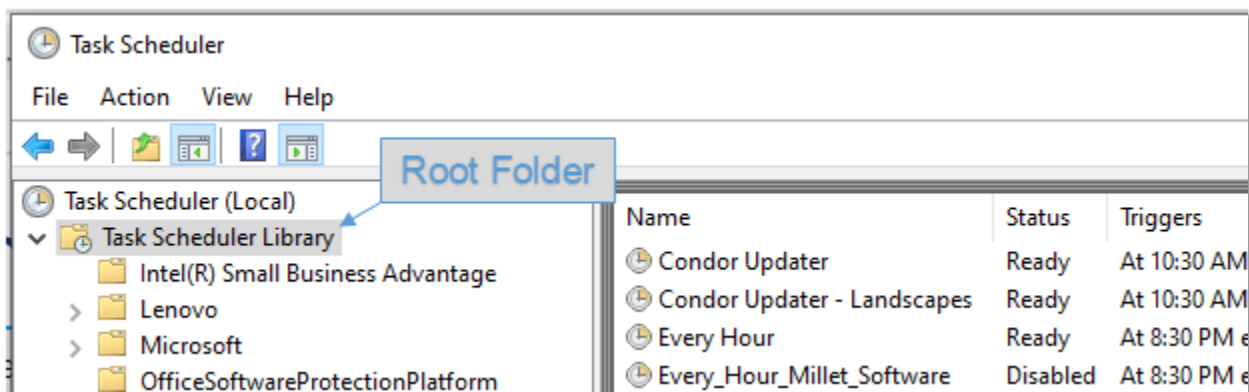
```
[Scheduled_Tasks]
Monitor_Scheduled_Tasks=True
Every_N_Minutes=10
Ignore_Repeats_for_N_Minutes=1440
```

In the example above, ActionQ would check for failures every 10 minutes and would avoid duplicate alerts (same task name & error code) for 1440 minutes (24 hours).


When failures are detected, the ActionQ administrator would receive an email alert such as this:



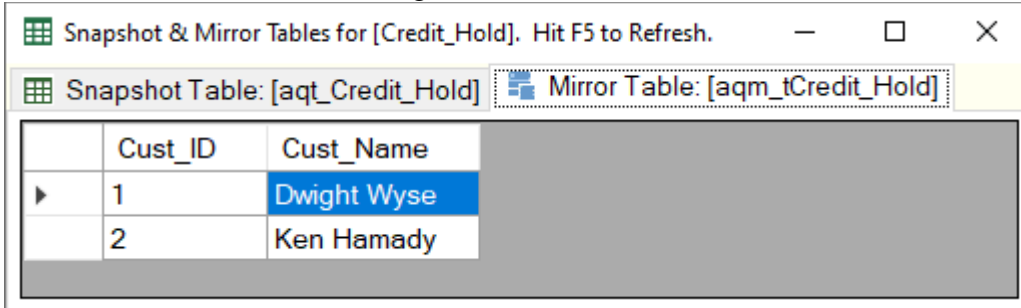
ActionQ monitors only **enabled** scheduled tasks under the **root folder** of task scheduler:



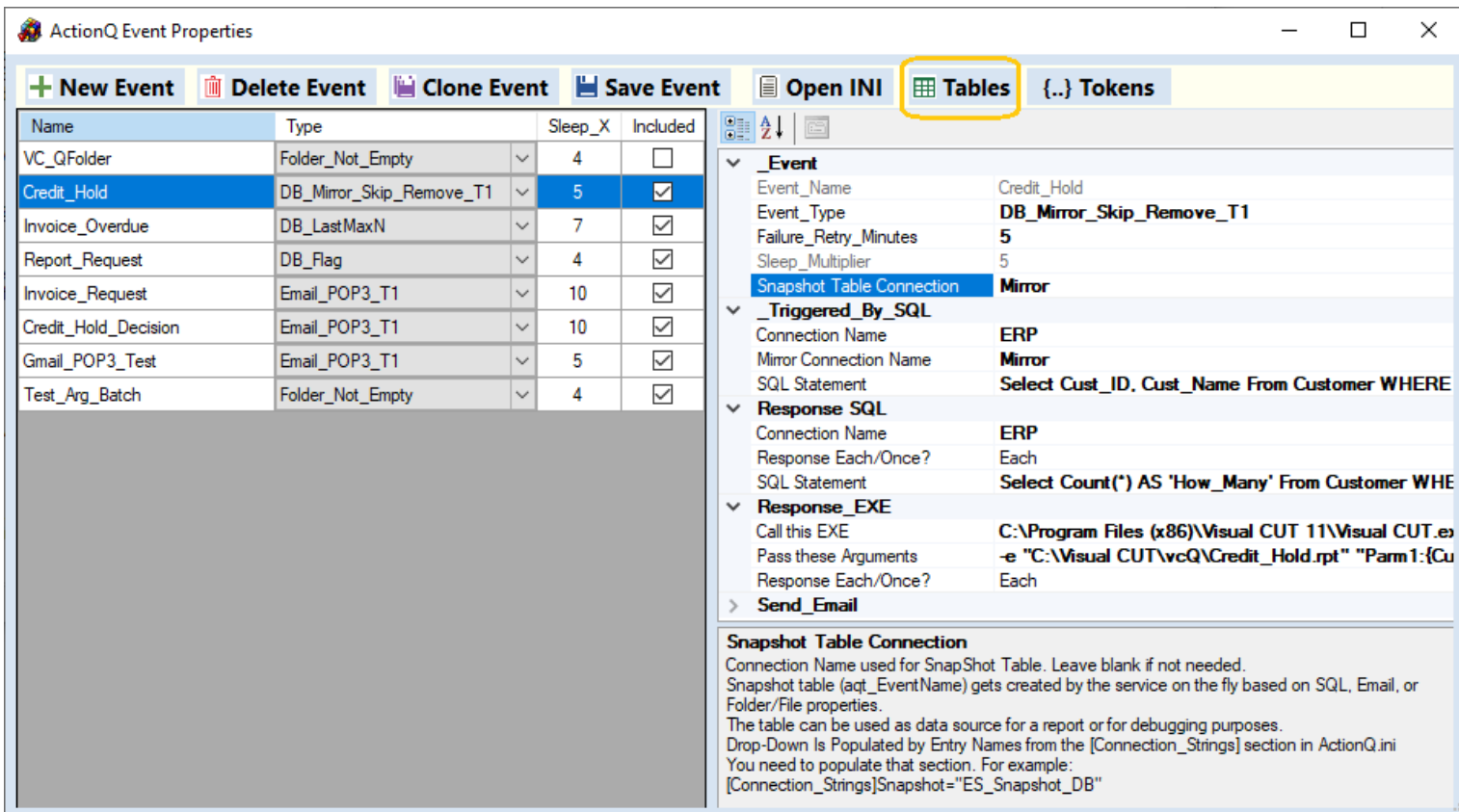
## Monitoring Snapshot and Mirror Tables

The  **Tables** button in the Action Manager menu launches a window displaying the *Snapshot* and/or *Mirror* tables for the currently selected event. Here is an example for an event that has both tables.

This allows you to monitor that information for debugging and training purposes.  
Hit F5 to refresh the data in the grids.



	Cust_ID	Cust_Name
▶	1	Dwight Wyse
	2	Ken Hamady



**ActionQ Event Properties**

**Tables** (highlighted)

Name	Type	Sleep_X	Included
VC_QFolder	Folder_Not_Empty	4	<input type="checkbox"/>
Credit_Hold	DB_Mirror_Skip_Remove_T1	5	<input checked="" type="checkbox"/>
Invoice_Overdue	DB_LastMaxN	7	<input checked="" type="checkbox"/>
Report_Request	DB_Flag	4	<input checked="" type="checkbox"/>
Invoice_Request	Email_POP3_T1	10	<input checked="" type="checkbox"/>
Credit_Hold_Decision	Email_POP3_T1	10	<input checked="" type="checkbox"/>
Gmail_POP3_Test	Email_POP3_T1	5	<input checked="" type="checkbox"/>
Test_Arg_Batch	Folder_Not_Empty	4	<input checked="" type="checkbox"/>

**Event Properties:**

- \_Event**
  - Event\_Name: Credit\_Hold
  - Event\_Type: DB\_Mirror\_Skip\_Remove\_T1
  - Failure\_Retry\_Minutes: 5
  - Sleep\_Multiplier: 5
  - Snapshot Table Connection: Mirror
- \_Triggered\_By\_SQL**
  - Connection Name: ERP
  - Mirror Connection Name: Mirror
  - SQL Statement: Select Cust\_ID, Cust\_Name From Customer WHERE
- Response\_SQL**
  - Connection Name: ERP
  - Response Each/Once?: Each
  - SQL Statement: Select Count(\*) AS 'How\_Many' From Customer WHE
- Response\_EXE**
  - Call this EXE: C:\Program Files (x86)\Visual CUT 11\Visual CUT.e
  - Pass these Arguments: -e "C:\Visual CUT\vcQ\Credit\_Hold.rpt" "Parm1:{Cu
  - Response Each/Once?: Each
- Send\_Email**

**Snapshot Table Connection**

Connection Name used for SnapShot Table. Leave blank if not needed.  
Snapshot table (aqt\_EventName) gets created by the service on the fly based on SQL, Email, or Folder/File properties.  
The table can be used as data source for a report or for debugging purposes.  
Drop-Down Is Populated by Entry Names from the [Connection\_Strings] section in ActionQ.ini  
You need to populate that section. For example:  
[Connection\_Strings]Snapshot="ES\_Snapshot\_DB"

## Options

### Customizing About Window in ActionQ Manager

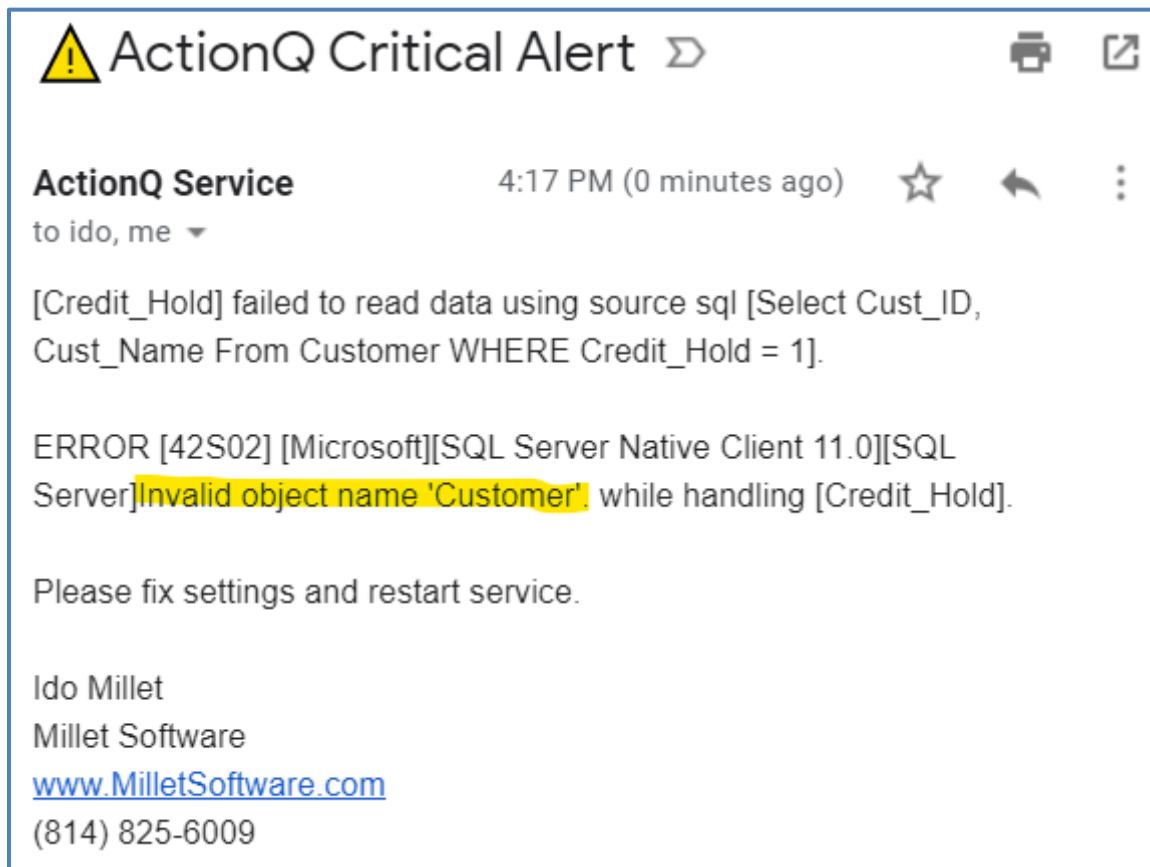
Set the following lines in the ini file:

```
[Options]
About_Line1="email: ido@MilletSoftware.com"
About_Line2="Skype: ido_millet"
About_Line3=www.MilletSoftware.com
```

## Handling Failures

When a failure occurs, the service emails a message to the addresses specified in *Email\_Failure\_Notices\_To* option as described in [Setting Email Options](#).

Here is an example of such an email triggered when a database table used in a Source\_SQL\_Template is not found:



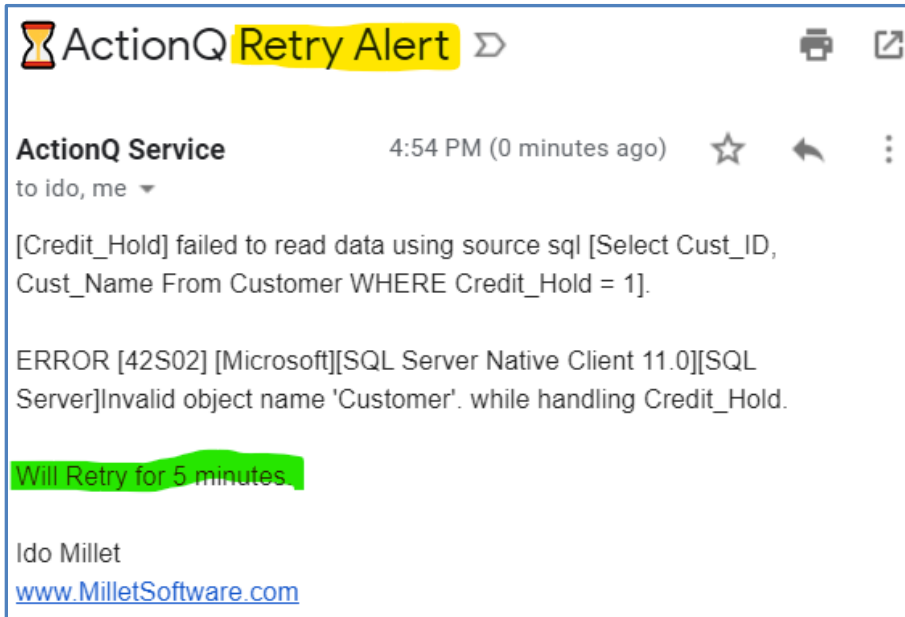
As stated in the email message, once an event fails, the service will no longer monitor it. You need to fix the problem and restart the service.

## Setting Failure Retry Period

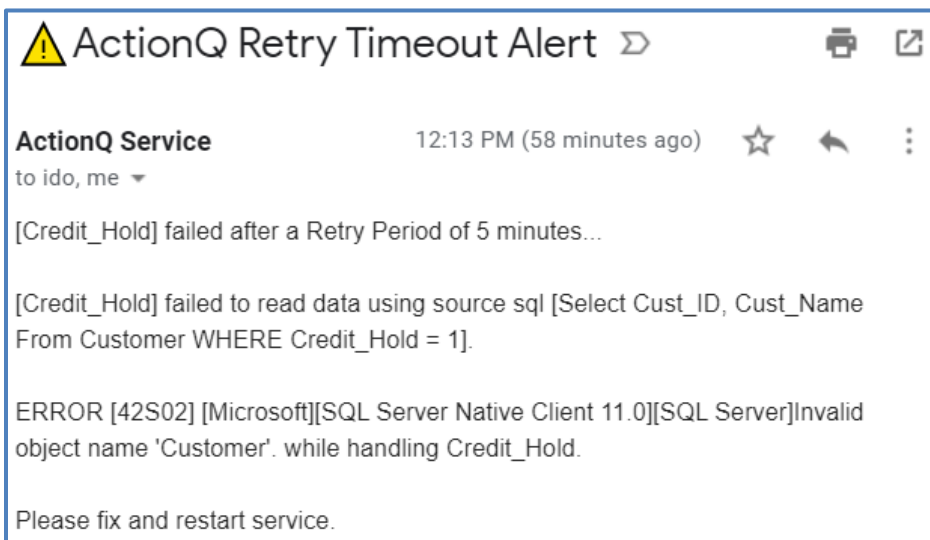
To handle transitory problems such as loss of connectivity to a database or to a cloud folder, you can set an optional entry in the specific event properties:

`Failure_Retry_Minutes="5"`

This would place a failing event in a Retry mode for 5 minutes.  
And the user would receive an email message such as this:



If after 5 minutes the process still fails, it is disabled and a final alert email is sent:



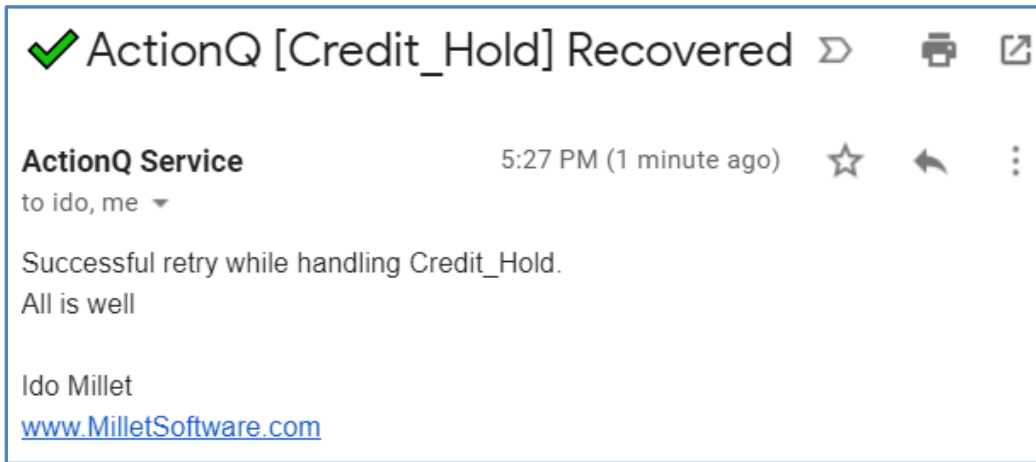
The event recovers if, before getting disabled as above, it:

a) experiences a success (triggering an EXE or moving a file)

\* Or \*

b) stops experiencing failures during one more retry period beyond the initial retry period.

An email confirms the good news like this:



# Technical Notes

## Database Connectivity

### 64 vs 32-Bit DSNs

The 64-bit version of ActionQ can only use 64-bit ODBC DSNs.

A 32-bit version of ActionQ is available upon request if you can only use 32-bit ODBC DSNs.

### NT Authentication

If you are using SQL Server with NT Authentication, unless you [Set the ActionQ Service to Run Under a User Account](#) (and that user account has login permissions to SQL Server), you need to map the SYSTEM account to have Login permissions. Go to SQL Server >> Security >> Logins and set the User Mapping for NT AUTHORITY\SYSTEM as shown below:

Login Properties - NT AUTHORITY\SYSTEM

Select a page

- General
- Server Roles
- User Mapping
- Securables
- Status

Script ? Help

Users mapped to this login:

Map	Database	User	Default Schema
<input checked="" type="checkbox"/>	ActionQ_Mirror	NT AUTHORITY\SYSTEM	dbo
<input type="checkbox"/>	master		
<input type="checkbox"/>	model		
<input type="checkbox"/>	msdb		
<input type="checkbox"/>	tempdb		
<input checked="" type="checkbox"/>	vcQ	NT AUTHORITY\SYSTEM	dbo
<input type="checkbox"/>	WE		

< >

☐ Guest account enabled for: ActionQ\_Mirror

Database role membership for: ActionQ\_Mirror

- ☐ db\_accessadmin
- ☐ db\_backupoperator
- ☐ db\_datareader
- ☐ db\_datawriter
- ☐ db\_ddladmin
- ☐ db\_denydatareader
- ☐ db\_denydatawriter
- ☒ db\_owner
- ☐ db\_securityadmin
- ☒ public

OK Cancel

## SQL Statements

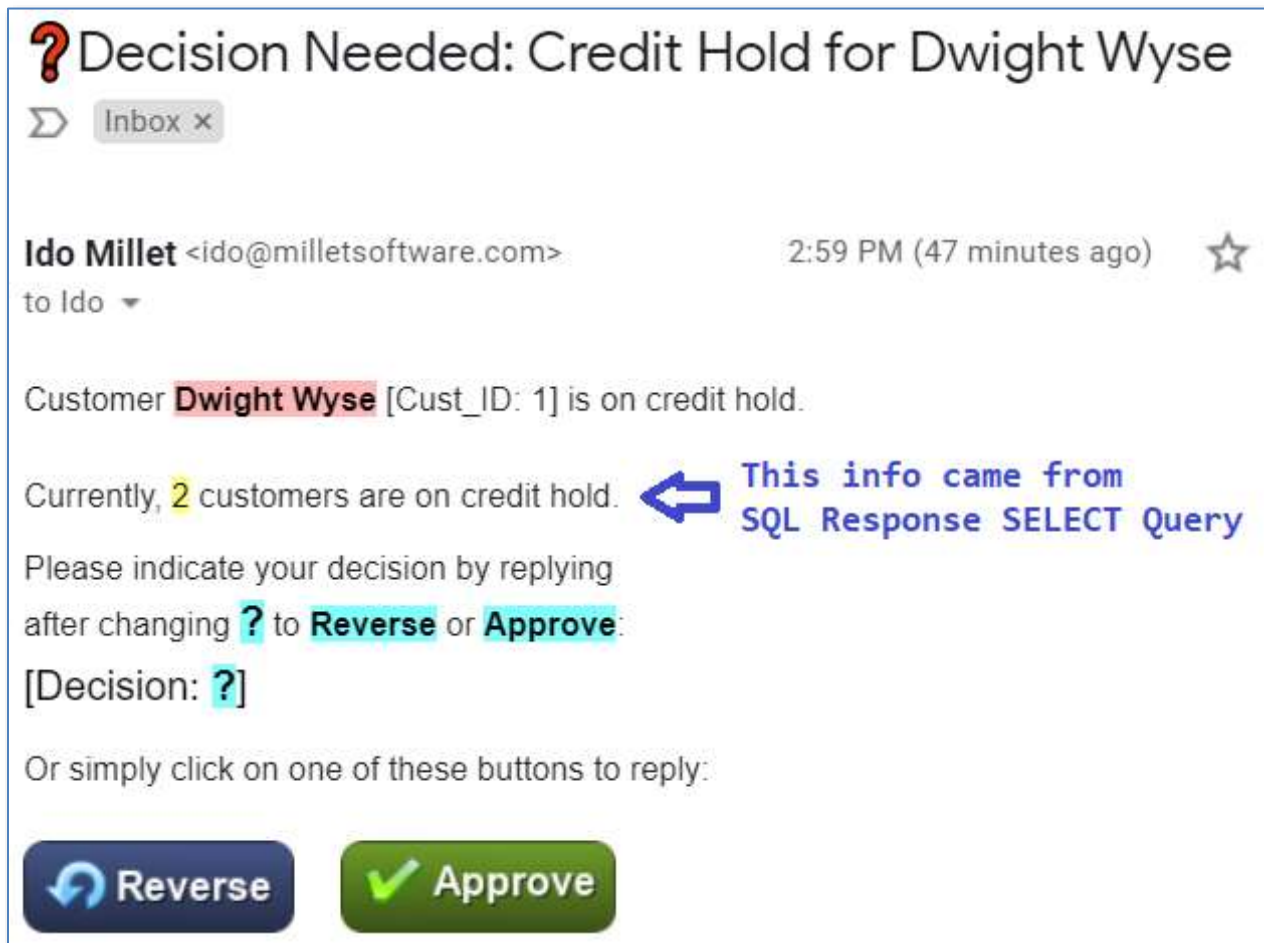
### SQL\_After as SELECT Query

If the **SQL\_After** is a SELECT statement, ActionQ adds extra **{af\_<columnName>}** tokens as well as **{aq\_HTML\_Table\_After}** token for each row in the main event tokens.

The information in **these extra dynamic tokens can be used in the Email or EXE responses** because those responses are always triggered AFTER the SQL response.

**SQL\_After** statements are now (Version 2.0) allowed to return multiple rows. **{af\_<colName>}** tokens are generated from the first returned row. The **{aq\_HTML\_Table\_After}** token is generated from all the rows. Note: When set to run for **Each** row in the main event (rather than **Once**), the **SQL\_After** logic fires and repopulates these tokens for each row in the main event.

Here is an example of an email from Visual CUT (EXE response) where information from a SELECT statement in the SQL response (how many customers are on credit hold) was used as a dynamic token in the command line arguments within the call to the Visual CUT EXE.



## SQL Statement Subquery

If the SQL Statement Subquery is a SELECT statement, ActionQ adds extra **{sb\_<columnName>}** tokens as well as **{aq\_HTML\_Table\_Subquery}** token for each row in the main event. The information in these extra dynamic tokens can be used in any follow-up actions/responses, including in the **SQL\_After** statement

▼ <b>_Event</b>	
Event_Name	Credit_Hold
Event_Type	DB_Mirror_Skip_Remove_T1
Failure_Retry_Minutes	5
Sleep_Multiplier	5
Snapshot Table Connection	Mirror
▼ <b>_Triggered_By_SQL</b>	
Connection Name	ERP
Mirror Connection Name	Mirror
SQL Statement	Select ID, Cust_Name, Balance FROM Customer WHERE Credit_Hold = 1
SQL Statement Subquery	Select ID, Date, Amount, Balance_After FROM Payments WHERE ID = {ID}
▼ <b>Response SQL</b>	
Connection Name	ERP
Response Each/Once?	Each
SQL Statement	Select Count(*) AS 'How_Many' FROM Customer WHERE Credit_Hold = 1
▼ <b>Response_EXE</b>	
Call this EXE	C:\Program Files (x86)\Visual CUT 11\Visual CUT.exe
Pass these Arguments	-e "C:\Visual CUT\vcQ\Credit_Hold.rpt" "Parm1:{Cust_ID}" "Parm2:{How_Many}"
Response Each/Once?	Each
▼ <b>Send_Email</b>	
SMTP_Settings	Email_From_Sales
Email_To	ido.millet@gmail.com

**SQL Statement Subquery**  
Optional! The Correlated Subquery to run for each main row.  
Generates {aq\_HTML\_Table\_Subquery} from all rows and {sb\_<colName>} tokens for 1st row.  
  
For example, main query may return several new order rows.  
And for each row, you want to email an HTML table showing the order's line items.  
Another use case is to Mirror only Key Columns and get more columns via this Subquery.  
  
Sample: Select Date,Amount,Balance from Payments where Cust\_ID = {Cust\_ID}

## SQL Server (fully qualify table names)

To avoid errors such as:

[SQL Server]Invalid object name 'dbo.Email'

use the fully qualified table name (including the database name) in SQL statements.

Good example (where **AQ** is the database name):

```
INSERT INTO AQ.dbo.Email ("Email_DT", "Email_From", "Email_Subject", "Email_BodyText",  
"Attachments", "Answer", "Note") Values ({EmailDT}', '{FromAddress}', '{Subject}', '{BodyText}',  
'{Attachments}', '{Answer}', '{Note}');
```

Bad example:

```
INSERT INTO dbo.Email ("Email_DT", "Email_From", "Email_Subject", "Email_BodyText", "Attachments",  
"Answer", "Note") Values ({EmailDT}', '{FromAddress}', '{Subject}', '{BodyText}', '{Attachments}',  
'{Answer}', '{Note}');
```

## ProgramData Folder

The settings file (**ActionQ.ini**), the service private ini file (**ActionQ\_Data.ini**) and the daily log files are maintained at:

**%APPDATA%** \MilletSoftware\ActionQ\

For example, on a Windows 10 machine, the ActionQ.ini file is at:

C:\ProgramData\MilletSoftware\ActionQ\ActionQ.ini

## Settings Files (ActionQ.ini and ActionQ\_Data.ini)

All entry values (even numeric ones) must be enclosed in double quotes. For example:

Sleep\_Multiplier="4"

All comments in the ini file must start with a semi-colon: ;

While ActionQ Service is running it maintains an in-memory copy of the **ActionQ.ini** file as a Read Only version. It never attempts to write to that file. That makes it safe for you to make changes to the settings while the service runs.

Values the service needs to “remember” even if it is stopped or uninstalled (such as LastMaxN) are saved by the service to **ActionQ\_Data.ini** file. Typically, you should not open or edit that file.

## UNC Paths rather than Mapped Drives

Use UNC paths rather than mapped drive letters when referring to resources such as Crystal Reports files in command line arguments.

ActionQ automatically converts mapped drive to UNC paths for the following properties:

*Watch\_Folder, Move2\_Folder, smtpQ\_Folder .*

## Invisible Mapped Drives

ActionQ Manager **runs as administrator** and hence may not display mapped drive letters in folder selection dialogs unless you use one of the solutions discussed in this [Microsoft Note](#).

You can copy & paste UNC paths manually into options for folder and file targets.

## Batch File to Make Mapped Drives Visible

As a solution for the problem above, a batch file is provided in the ActionQ application folder:

**C:\temp\Enable Mapped Drives in Elevated Apps (Right-Click and Run as admin).cmd**

Running that batch file sets a registry entry allowing elevated applications to see mapped drive paths.

A machine restart is needed to apply the change.

## Queuing Emails to smtpQ Outgoing Folder

If you also have Visual CUT installed with smtpQ enabled, you can queue outgoing emails to the Outgoing folder by setting this entry in the ActionQ ini file under the [Events] section:

```
[Events]
smtpQ_Folder="C:\Visual CUT\smtpQ\Outgoing"
```

## ActionQ Service Fails to Start

If you get the following error:

**Error 1053: the Service Did Not Respond to the Start or Control Request in a Timely Fashion**

Try to:

- Reboot the computer
- Review the suggestions in this [blog](#).

## Custom Logging (override appsettings.json)

To override the custom logging options, which by default are specified in the *appsettings.json* file, create a *logsettings.json* file and place it in the same folder.

For example, this can easily allow you to use a cloud logging service such as *Seq*.

# Update History

## Version 3.0.5.3 (entered testing May 12, 2024):

- **New response type of calling a PowerShell script.** You can pass dynamic tokens to the PowerShell script. For detail, see [PowerShell Response](#). See [video demo](#) [📺]
- **New event type of PowerShell script** (PS\_Mirror\_Skip\_Remove\_T1). See [video demo](#) [📺]  
This is similar to the Database event, except that the data comes from a PowerShell script. For example, you may want to get data via **REST API** calls (e.g. **RabbitMQ**). Or monitor system health. Typical responses are email alerts and updating databases.  
For detail, see [PowerShell "PS\\_Mirror\\_Skip\\_Remove\\_T1" Event Type](#).

## Mirror Table Features

- **Mirror Table logic now ignores column names that start with a space** (e.g. ' Balance ').  
This allows including useful data (e.g. balance of account placed on credit hold) in event data, yet **avoiding triggering another response when such non-key information changes**.  
For example, once a customer is placed on credit hold, we may not want to trigger another alert just because their balance changed.
- Newly created **mirror tables store only key columns** (column names that do not start with a space).  
This saves space, increases speed, and improves security. It also improves resilience because you can add/remove/change non-key columns to the event design without breaking the mirror table logic.

## Logging Features

- **Added dynamic logging as an event response** (typical targets are Seq or RabbitMQ).  
This is not yet documented yet. Contact Millet Software for testing this new feature.
- Added support for logging to Serilog Seq and Serilog RabbitMQ.
- Added more structured logging properties, including the event name.
- Added support for [custom logging](#) settings (*logsettings.json*)

## Other Features

- You can now **use both 64-bit and 32-bit versions on the same machine**.  
The new 32-bit version uses a service called ActionQ32 instead of ActionQ.  
It automatically renames & uses separate files (ActionQ32.ini, ActionQ32\_Data.ini, ActionQ32.log).
- The ActionQ Service was upgraded to .NET 8.0
- When the service fails to install/start/stop/uninstall the service, a message explains the reason.
- Fixed a problem with how the Encrypted Strings editor displays previously saved settings
- Installation now includes a sample ini file with many examples of events, encrypted strings, database connections, mailboxes, etc.
- Installation now includes a sample json file for Office365 OAuth 2.0 email authentication.
- Added a **batch file to automate the installation process** (ActionQ\_Install\_RClick\_Run\_As\_Admin.cmd).
- When adding a database connection profile, a helpful text of **dsn=?;uid=?;pwd=ES\_?** is provided.


## Fixes

- Fixed a problem in Mirror table handling of Boolean values.
- Fixed a problem in how the 64-bit version shows the connection string for database connections.
- **Fixed a problem with how the service handled OAUTH for SMTP via Office365.**

### Version 2.0.8.01 (February 28, 2024):

- Added support for **OAuth2** when sending/receiving emails (SMTP & POP3) via Office365 and Gmail. Contact Millet Software for detailed instructions.
- **Fixed multi-row SQL population of dynamic tokens.**
- **Fixed {aq\_HTML\_Table} bug** (introduced in V2.0.5, and remained in V2.0.6)
- Removed several CSS style defaults for tables in email messages (email clients require inline styling).
- Fixed display of object properties (event, email, inbox, database connections) after cloning the object.
- Fixed a bug in adding a new inbox profile.

### Version 2.0.2.01 (August 1, 2023):

- **ActionQ can now [call a REST API](#) as a response to an event.** See [video demo](#) 
- Returned JSON is parsed to dynamic tokens that can be embedded in a follow-up email message. For example, an incoming email event can trigger an SMS message via Twilio's REST API. Twilio's JSON response is parsed into tokens that can be embedded in an email as another response.

### Version 2.0.1.01 (July 04, 2023):

- Fixed sanitizing of connection strings when logging connection failures.

### Version 2.0.0.01 (June 12, 2023):

- **Breaking Change!** *SQL After* statements tokens names are **{af\_<colName>}** instead of **{<colName>}**. For example, **{af\_ID}** instead of **{ID}**.  
This solves cases where other SQL statements for the same event have columns with the same name. If you have an event that uses tokens from an SQL After statement, be sure to update that event.
- *SQL After* statements are now allowed to return multiple rows. For SELECT statements, this provides a new **{aq\_HTML\_Table\_SQL\_After}** token for embedding in email responses.  
Note: When set to run for **Each** row in the main event (rather than **Once**), the *SQL After* logic fires and repopulates these tokens for each row in the main event.
- **Added SQL Statement Subquery property to database events.** *ActionQ* runs that correlated subquery for each row in the main query. For SELECT statement, it populates new **{aq\_HTML\_Table\_Subquery}** and **{sb\_<colName>}** tokens. For example, a backorder event can easily embed line items detail as HTML table in an email response. Thanks to Corey Durthaler for the feature idea.
- Dynamic tokens are now sorted alphabetically in the token panel.
- Fixed double-prompting for unsaved changes.
- Added green highlights for action descriptions (e.g. 'emailed to', 'Ran', 'Subquery') in the log display.
- The SQL editor now starts Cast() expressions on new lines for improved readability.
- The SQL editor's data grid now right-aligns date and number columns.
- Auto-generated HTML tables now handle Date values as dates rather than DateTime.
- The log display automatically abbreviates content of HTML tables to `<table ... </table>`.

### Version 1.1.97.02 (November 20, 2022):

- Updated internal component to a later version.
- Bug fix (null object reference).

### Version 1.1.89.02 (July 10, 2022):

- Fixed a problem in testing email profiles.
- Fixed an emailing problem from the service.
- Added debug option to review mirror table vs fresh data.
- Fixed Mirror Table comparisons for Decimals values with different number of trailing 0's.
- Fixed tooltip issue for the {aq\_HTML\_Table} token.
- Updated the email message HTML editor. Preview scrollbars are now enabled.

### Version 1.1.0.95 (March 2, 2022):

- Email events can now **use Regular Expressions** to parse dynamic tokens from the email **subject**, message **body**, and even **file attachments**. See [Dynamic Tokens By Regular Expressions](#).

### Version 1.1.0.93 (January 6, 2022):

- ActionQ now automatically builds an HTML table token called {aq\_HTML\_Table}. You can place that token in email messages to display event data as a nicely formatted table. See [Using {aq\\_HTML\\_Table} Token in Email Responses](#).
- Fixed handling of NULL values in Mirror table logic ('DB\_Mirror\_Skip\_Remove\_T1' event type).
- SQL Query editor now provides a button (only for DB\_Mirror\_Skip\_Remove\_T1 event types) allowing you to display only new cases (rows that are not already in the Mirror table). See [example](#).
- Added an example for [Using a Stored Procedure as Data Source](#).
- Added a [video demo](#).
- Fixed bug in properties grid for email inboxes.
- Fixed a bug in handling of encrypted strings.
- Added syntax highlighting to SQL editor.

### Version 1.1.0.87 (November 6, 2021):

- Added a 32-bit version of ActionQ (to support 32-bit ODBC drivers). Contact Millet Software for detail.
- Fixed support for HTML email signatures.
- Added a special dialog for entering & testing Source\_SQL\_Template statements for database events. When ActionQ displays the result set from the SELECT statement, it also takes care of populating dynamic tokens to facilitate designing dynamic responses. See [Source\\_SQL\\_Template Editor](#).
- When saving a new Folder or Incoming Email event, ActionQ now automatically generates and saves sample tokens to facilitate designing dynamic responses even before the events get triggered.

### Version 1.1.0.83 (September 19, 2021):

- **Folder\_Not\_Empty** events now support an optional **Renamed\_File** property, allowing you to rename incoming files. For in-place renaming, you may leave the **Move2\_Folder** blank.
- **Folder\_Not\_Empty** events now support flexible **DateTime** and **File Counter** tokens to ensure the Renamed\_File is unique. See [Dynamic Tokens for Renaming Files](#).
- Email/SQL/App responses to **Folder\_Not\_Empty** events, which are set to fire ONCE (instead of for EACH incoming file) now indeed fire only once.
- **Ctrl-S** now acts as a Save hot key for event and global properties.
- A **?** top-right menu button in various windows (or F1 key) now launches relevant online help.


### Version 1.1.0.82 (September 8, 2021):

- Added a window for managing global settings via a property grid with help text. This avoids manual editing of the ini file to manage settings such as sleep interval and service account.
- Added a blank option in SMTP\_Settings drop-down to allow removing email response target.
- Events can now use the admin email profile (“Email”) settings to send emails.
- Added right-click menu for the log window with menu options to Refresh (F5) and Clear (F12) the log.
- Changing the ‘Included’ checkbox for an event saves the change (no need to click ‘Save Event’).
- Added alert dialogs when a user is about to lose unsaved changes to property values due to closing a window or navigating to another event. The user can elect to save or discard the unsaved changes.
- A failure message is now logged & emailed when a *Folder\_Not\_Empty* event finds the file already exists in the *Move2\_Folder*.
- Added a [Batch File to Make Mapped Drives Visible](#) .
- Added automatic conversion from mapped drive to UNC paths.
- *Folder\_Not\_Empty* events now support an optional *Target\_Files* property, allowing you to target only specified file names or wildcard patterns. For example: `*.xls;*.xlsx;Claims*.csv`

### Version 1.1.0.74 (August 17, 2021):

- Fixed handling of single quotes when populating Mirror and Snapshot tables
- Fixed saving of Mirror Connection property.
- Fixed property grid initialization when saving a new event.

### Version 1.1.0.71 (July 23, 2021):

- Added an *Email Profiles Manager*. Use it to add, edit, delete, clone, **and test** email profiles. See: [Using the Email Profiles Manager](#).
- Added an *Email Inbox (POP3) Manager*. Use it to add, edit, delete, clone, **and test** settings for email inboxes. See: [Using the Email Inboxes \(POP3\) Manager](#).
- For security reasons, the user can no longer opt to display unobscured encrypted strings.
- Instead of encrypting the whole connection string, you can now refer to a named encrypted string from just the sensitive portions. For example: `dsn=ERP;uid=jsmith;pwd=ES_ERP_PW`  
This facilitates managing connection strings by making the non-sensitive parts visible.
- Increased font size in property grids to improve readability.
- Fixed duplicate key bug in detecting failed schedules tasks.
- Fixed handling of dummy/blank encrypted strings
- Fixed email profile test bug
- Added tooltips to menu buttons
- Removed *Email\_Bounce\_Address* property (by definition, must be the *Email\_From*).
- Added to the main form a button  to open the ActionQ.ini file.
- The User Manual button now opens the user manual in your browser.
- The user manual now provides an [Initial Setup](#) section with 3 required steps and 2 optional steps.

### Version 1.1.0.63 (June 25, 2021):

- Updated HTML editor for email messages.
- Added a *Connections Manager*. Use it to add, delete, clone, **and test** named database connections. See [Using the Connections Manager](#).

### Version 1.1.0.61 (March 29, 2021):

- New Feature: see [Setting Up Alerts for Windows Scheduled Task Failures](#).
- When saving an event that uses the LastMaxN property, a dialog allows the user to avoid overwriting the current value used by the service in a case where the value shown by the manager dialog is old.
- Fixed an issue with showing LastMaxN in event properties.
- Updated HTML editor component for email messages.

### Version 1.1.0.57 (February 9, 2021):

- Added **Attachments** property for email response. Separate multiple attachments with ‘;’.
- Email responses can now have **HTML message bodies**.
- Added an **HTML editor** (including inline spell-checker) for designing HTML email messages. See [HTML Editor for Email Message Body](#).

### Version 1.1.0.52 (January 24, 2021):

- You can now use a SELECT statement in the SQL Response to add extra dynamic tokens for use in the Email or EXE response. For details, see [SQL Response as SELECT Query](#).
- For email events, if Delete\_Email\_From\_Server entry is set to false, ActionQ now avoids duplicate processing of old messages by tracking the maximum date & time found in the date header of messages processed in previous scans.
- Editor window for text properties now allows font size increment/decrement using Ctrl-Scroll Up/Down.
- Editor window for text properties now adds a new line when pressing ‘Enter’ instead of closing.
- Added automated documentation of all dynamic tokens and sample values for each process. A new toolbar button allows you to view this documentation for a selected process. See [View Dynamic Tokens for Each Process](#).
- Added special dialog for editing response properties with dynamic tokens. See [Embed Dynamic Tokens in Response Properties](#).

### Version 1.1.0.47 (January 13, 2021):

- The encrypted string creation dialog now also allows creation of a named connection string referencing the encrypted string. This allows creation of named connection strings without manual editing of the ini file. See [Database Connections](#).
- Added special handling for POP3 (incoming emails) from Gmail.
- Fixed error message when event triggers only an email (no exe and no SQL).
- Fixed saving of SQL\_After\_Connection property from event properties editing window.
- Improved layout (spacing between sections) and comments in Ini file.
- Event properties grid now shows **Response SQL** properties before **Response\_EXE** properties. This reflects the response sequence: SQL response occurs before EXE response.
- Fixed informational logging issue.
- Improved handling of cases where user forgot to activate at least one event.
- Fixed SQL handling of single quotes inside email subject and body.
- Fixed dynamic tokens handling for multiple files in Folder events.
- Added more tokens for Folder events: `{FileNameNoExtension}`, `{Extension}`, `{DateTimeStamp}`

### Version 1.1.0.34 (December 7, 2020):

- Performance tweaks.
- Added message suggesting closing AQ Manager before manually editing the ini file in Notepad.


### Version 1.1.0.33 (November 30, 2020):

- ActionQ can now trigger dynamic email messages in response to events.
- Added SQL\_When ('Once' or 'Each') event property to support cases where the SQL response should be triggered only Once per event as well as for Each Record/File/Email detected within the event.

### Version 1.1.0.31 (November 12, 2020):

- You can now queue outgoing emails using smtpQ. See [Queuing Emails to smtpQ Outgoing Folder](#).
- 'Pass These Arguments' event property now offers a multi-line editor for easier viewing/editing.
- Fixed a bug in saving setting for Move2\_Folder
- Fixed a bug in handling dynamic tokens for SQL\_After statements.
- Fixed a bug in some cases of reading Mirror database connection string.
- Fixed unhandled exception scenario.
- Fixed data type handling issue in creating snapshot/mirror tables.
- Form for viewing snapshot and mirror tables now shows the DSN name used by their connections.

### Version 1.1.0.22 (October 3, 2020):

- Added Snapshot\_Table\_Connection property to activate saving of event details to a table. This facilitates debugging and exposing the event data to downstream processes. For example, a Crystal report can use the snapshot table, removing the need to pass parameters via dynamic tokens in the arguments template.
- Added a  **Tables** button in ActionQ\_Manager to display the Snapshot and/or Mirror tables for the selected event. See [Monitoring Snapshot and Mirror Tables](#).
- Property Grid now provides a multi-line editor for pasting or viewing SQL statements.
- Log file refreshes no longer change user's clipboard.
- Fixed *Email\_POP3\_TI* connection problem.

### Version 1.1.0.21 (September 23, 2020):

- Added window for creating, cloning, and modifying events.
- Merged Install and Uninstall buttons into a single Install/Uninstall button.
- Merged Start and Stop buttons into a single Start/Stop button.
- Tweaked service installation logic

### Version 1.1.0.12 (September 7, 2020):

- Added **ActionQ\_Data.ini** for the service to maintain values in case it is stopped. This ensures the service never writes to the **ActionQ.ini** file, making it safe for you to change settings while the service runs.

### Version 1.1.0.10 (August 27, 2020):

- Enhanced saving of error messages to log and to Status entry in ini file.

### Version 1.1.0.9 (August 25, 2020):

- Added an event type of '*Email\_POP3\_TI*' to detect incoming emails (matching filter criteria), and triggering SQL and/or EXE with dynamic command line arguments in response. This includes automatic extraction of data from [Answer: <...>] and [Note: <...>] tokens in the message subject or body, allowing managers to approve/reject email requests and trigger processes and database updates by responding to an email.

**Version 1.0.0.8 (August 16, 2020):**

- Added options to [Set the ActionQ Service to Run Under a User Account](#).
- Starting ActionQ\_Manager for the first time takes care of copying the sample ini file to the AppData subfolder and setting Modify permissions for users.
- Several enhancements to user interface (e.g. changing color for service status text)
- Added support for running on older operating systems (e.g. Windows 2008 R2)
- Added toolbar button to open user manual.
- Added msi installer

**Version 1.0.0.0 (August 2, 2020):**