AUTOMATIONDIRECT.com

CLICK PLUS

Applications Guide

CLICK PLUS Applications Guide

AUTOMATIONDIRECT.com

Please include the Manual Number and the Manual Issue, both shown below, when communicating with Technical Support regarding this publication.

Manual Number:	C2-APP-GUIDE
Issue:	1st Edition, Rev. A
Issue Date:	02/2025

Publication History		
Issue	**Date**	**Description of Changes**
1st Edition	02/2025	Original
1st Edition Rev. A	02/2025	Moved importable project code to github.

Table of Contents

Top 10 things to know before starting with CLICK ... 1

Common PLC Programming Patterns ... 2

- Flashing Coil ... 2
  - Single Timer Flash ... 2
  - Drum Timers for Advanced Timer control ... 3
- Input Map ... 3
  - Debounce ... 3
  - Analog Input Filter ... 5
- State Machine Programming ... 7
- Motion Control ... 7
- Advanced Math Logic ... 9
  - Min/Max ... 9
  - Average ... 9
  - Convert to Percent ... 10

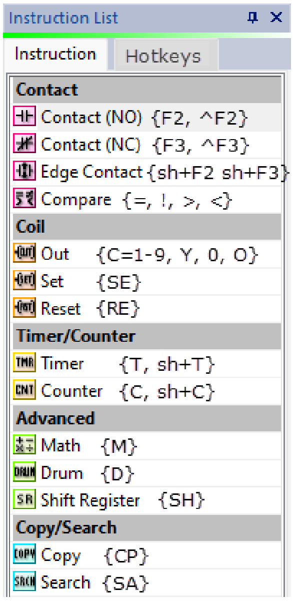
Node-RED Project Examples ... 11

- PROJECT: Initiate a Node-RED flow from a CLICK event ... 11
  - Project Summary: ... 11
  - Importable Project: ... 12
- PROJECT: Find the Max Value ... 13
  - Importable Project: ... 14
- PROJECT: Find your IP Address ... 14
- PROJECT: Connect to a Microsoft SQL Azure Database ... 15
  - SQL Select ... 16
  - SQL Update ... 17
  - SQL Insert ... 18
  - SQL Delete ... 18
- PROJECT: Connect to the local SQLite Database ... 20
  - Pros and Cons of using a local database: ... 20
  - Frequently Asked Questions: ... 20

Minimum configuration to use SQLite 21
PROJECT: Call a Web Service to retrieve data and chart the results....................23
Project Summary:....................23
Importable Project....................24
PROJECT: Local Weather Station 25
PROJECT: Create an MQTT Broker....................28
Importable Project....................28
PROJECT: Send a Barcode value to Node-RED via HTTP Get, parse it into fields on the dashboard, and perform a SQL Lookup 29
Importable Project:....................30
BONUS FUNCTION: Process Modbus - Four message types using JavaScript to create array of values: 31

Top 10 things to know before starting with CLICK

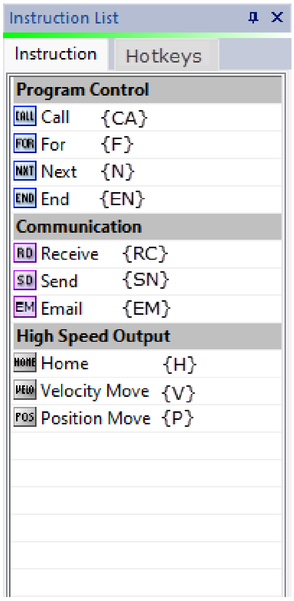
1. Use Hotkeys!



Instruction List

Instruction | Hotkeys

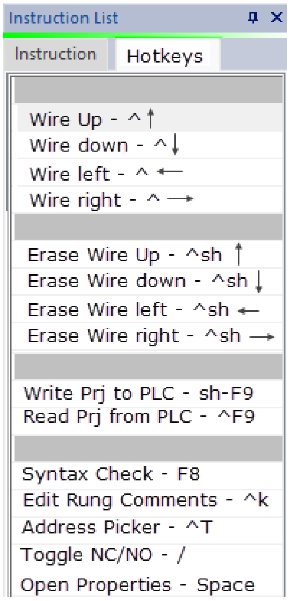
Contact	
Contact (NO)	{F2, ^F2}
Contact (NC)	{F3, ^F3}
Edge Contact	{sh+F2 sh+F3}
Compare	{=, !, >, <}
Coil	
Out	{C=1-9, Y, 0, O}
Set	{SE}
Reset	{RE}
Timer/Counter	
Timer	{T, sh+T}
Counter	{C, sh+C}
Advanced	
Math	{M}
Drum	{D}
Shift Register	{SH}
Copy/Search	
Copy	{CP}
Search	{SA}



Instruction List

Instruction | Hotkeys

Program Control	
Call	{CA}
For	{F}
Next	{N}
End	{EN}
Communication	
Receive	{RC}
Send	{SN}
Email	{EM}
High Speed Output	
Home	{H}
Velocity Move	{V}
Position Move	{P}



Instruction List

Instruction | Hotkeys

Wire Up - ^↑
Wire down - ^↓
Wire left - ^ ←
Wire right - ^ →

Erase Wire Up - ^sh ↑
Erase Wire down - ^sh ↓
Erase Wire left - ^sh ←
Erase Wire right - ^sh →

Write Prj to PLC - sh-F9
Read Prj from PLC - ^F9

Syntax Check - F8
Edit Rung Comments - ^k
Address Picker - ^T
Toggle NC/NO - /
Open Properties - Space

^ = ctrl and sh=shift

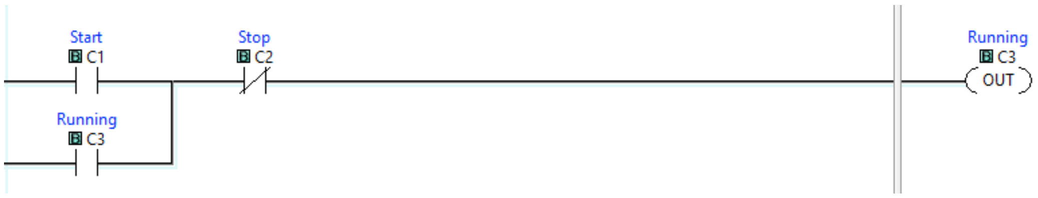
2. The Drum timer can be used as either a multistage timer (Blinker) or as a state machine controller.
3. Use Input maps.
4. Subroutines leave I/O values in their last known state if the subroutine stops running. An output that was on the last time a subroutine runs will stay on when the subroutine is disabled.
5. You can always find the IP address of your CLICK (Ethernet) in the footer of the programming tool.
6. You can always find the subroutine name in the CLICK Program header.
7. CLICK supports low code programming for High-Speed I/O, PID, MQTT, and Drum timers – review these features before you start programming, they can save you a ton of time.
8. XD/YD data types allow an easy way to set or send up to 16 bits at a time.
9. There is a high-speed programmable limit switch under the High-Speed I/O setup – look for "Presets".
10. CLICK is great for motion control and can be used for simple motion AND more complex motion like registration moves, velocity control, linear interpolation, and relative or absolute positioning.

Common PLC Programming Patterns

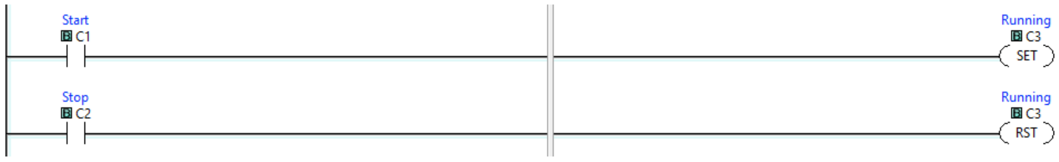
If you are new to ladder logic, the following are several of the most common patterns used. It's a good idea to make sure you understand these before you write your first program.

Latching Coil

The most common circuit is a latching circuit or Start/Stop circuit. This can be accomplished with either a single rung or you can use a latching output.



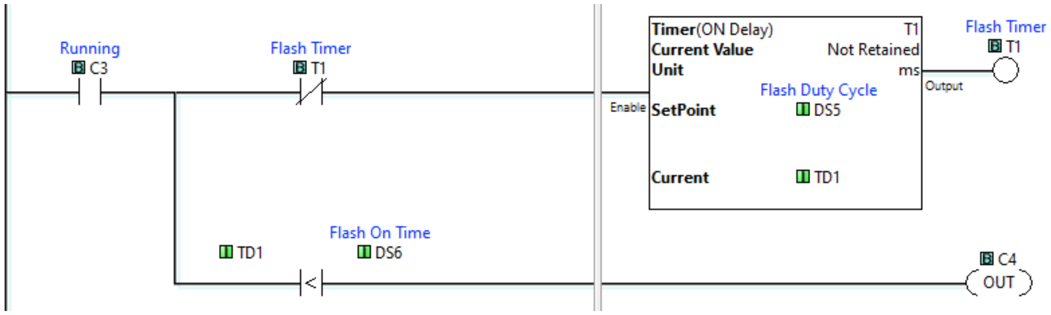
-or-



Flashing Coil

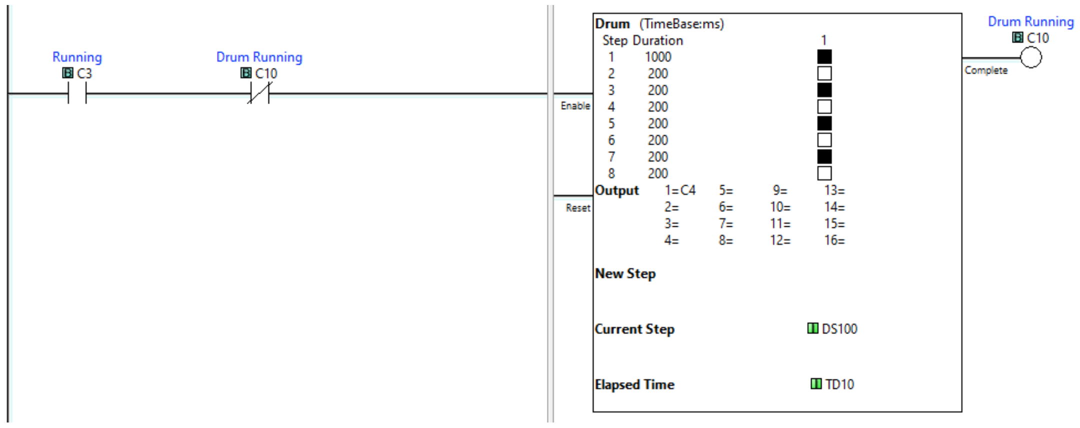
Single Timer Flash

Another one of the most common circuits is to create a flasher or on/off cycle. Here is an example of an easy one timer circuit that allows independent control of the on and off times.



In this example, the timer automatically resets when it is done. As long as the timer value is less than the "On" time (DS6), the output C4 will be on, after that C4 will turn off. So if DS5 is 3s, and DS6 is 1s, C4 will be on for 1 second, then off for the remaining 2 seconds in the timer. At the 3 second mark the timer will reset to 0 and C4 will turn back on. That cycle will repeat until C3 turns off.

Drum Timers for Advanced Timer control



Drum timers make complicated things look easy. The example above shows how to use a drum to create one long blink followed by 3 short blinks. This happens to be the letter "B" in morse code.

Input Map

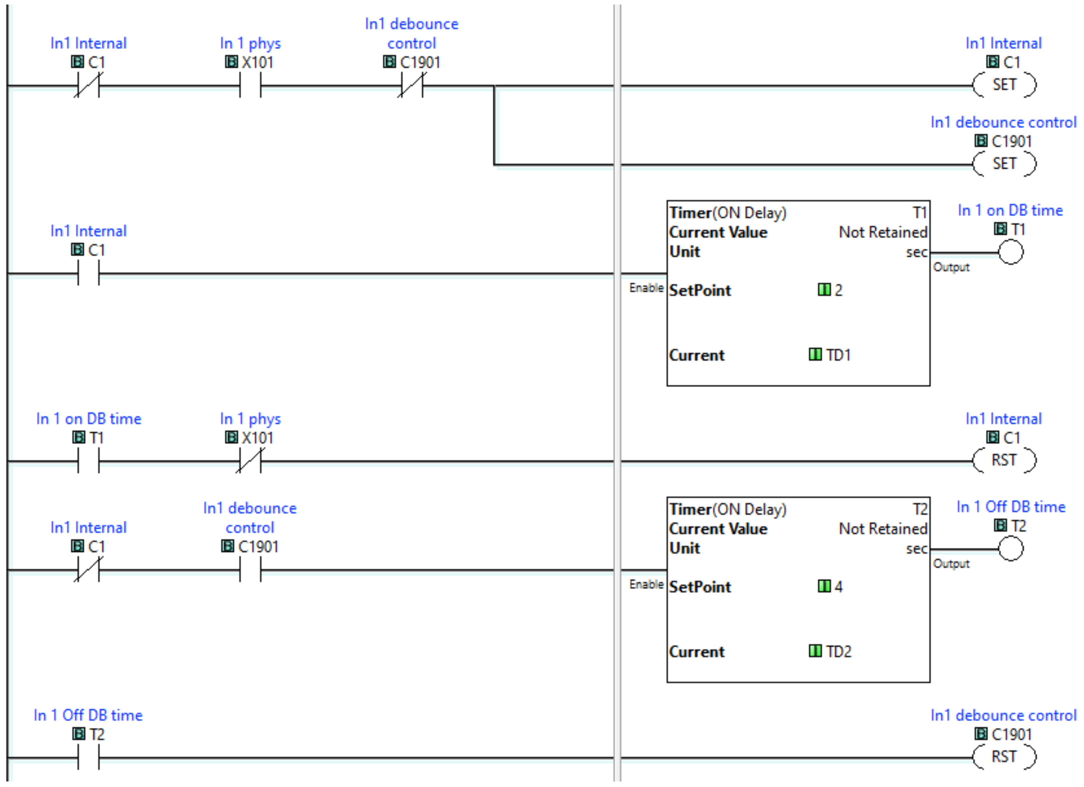
This is not really a circuit, but great programming practice to make your system easier to maintain. It also prevents issues from Asynchronous I/O scans (not an issue with CLICK). If you put this in a subroutine, this also gives you a place to apply debounce timers, and create flashers so they don't interfere with the readability of you main program flow.

Debounce

If your circuit uses physical relays or mechanical switches, you may encounter bouncing. This happens because the physical switch is made from a flexible copper wire that bends when the switch or relay changes state. That bending can allow the wire to bounce off the contact point when it opens or closes. Modern PLCs are fast enough to detect the rapid on-off changes that happen when a switch bounces. You may need to handle this in your program by either latching on a circuit so it stays on through the bounce (fast-acting), or delaying the "On" event until the switch has stabilized (slow-acting).

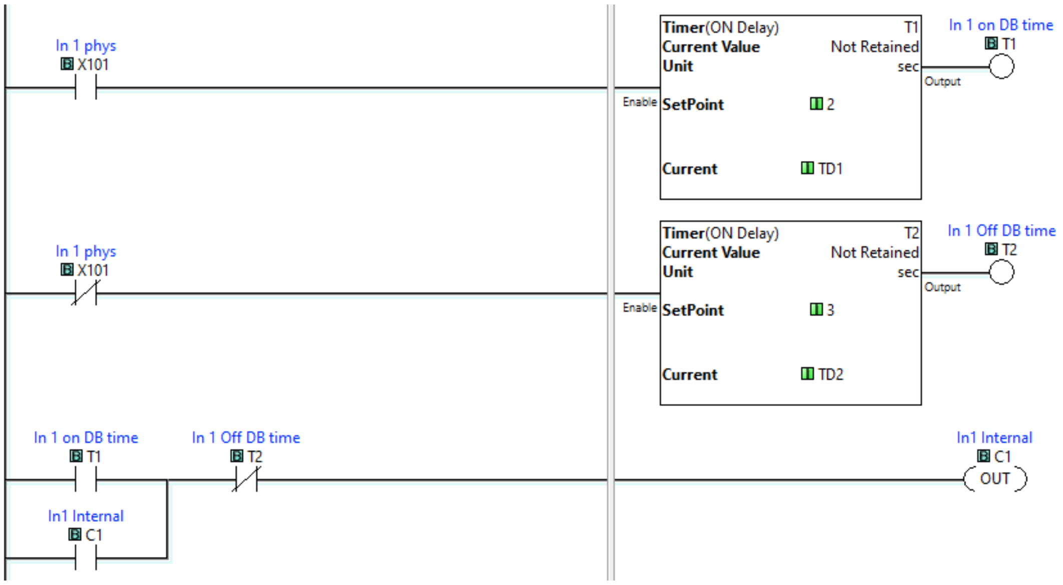
On-Delay, Off-Delay Debounce (Fast-acting)

When an input turns on, capture the on state and hold it for a minimum time to allow the input to stabilize. Fast acting can potentially capture transient voltage spikes, so use sparingly.



On-Delay, Off-Delay Debounce (Slow-acting)

When an input turns on, wait for a minimum time to allow the input to stabilize before enabling the circuit. This approach eliminates potential spikes, but delays the recognition of an on event for a specified time.

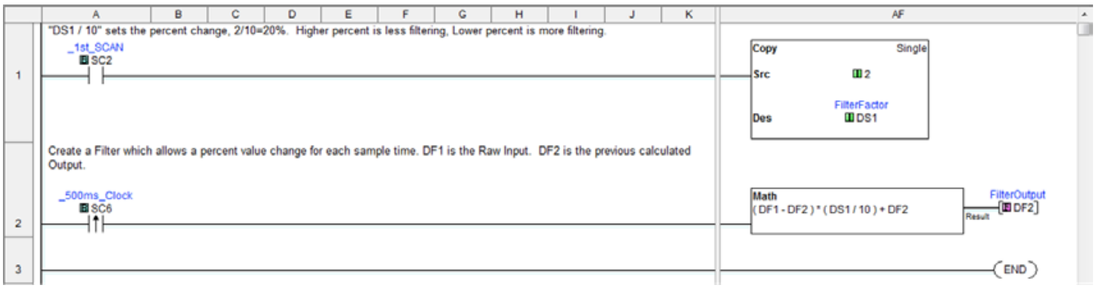


Analog Input Filter

Two methods of creating an Analog Filter in Ladder Logic are shown here. These can be useful for filtering analog inputs. The following examples provide simple methods to create a filter and average. Be sure to change the addresses to unused memory addresses in your project.

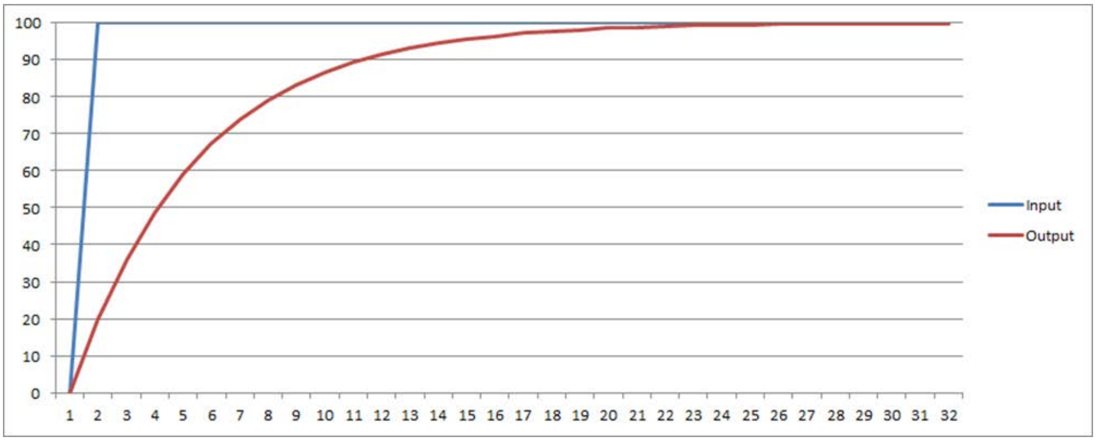
Filter Example

This method provides flexibility to adjust the sample time and percent change. At each sample time the difference is calculated between the Input and Output, then a percentage of this difference is applied to the new Output. A higher value in DS1 allows more change, or less filtering. A lower value in DS1 allows less change, or more filtering.


"DS1 / 10" sets the percent change, 2/10=20%. Higher percent is less filtering, Lower percent is more filtering.

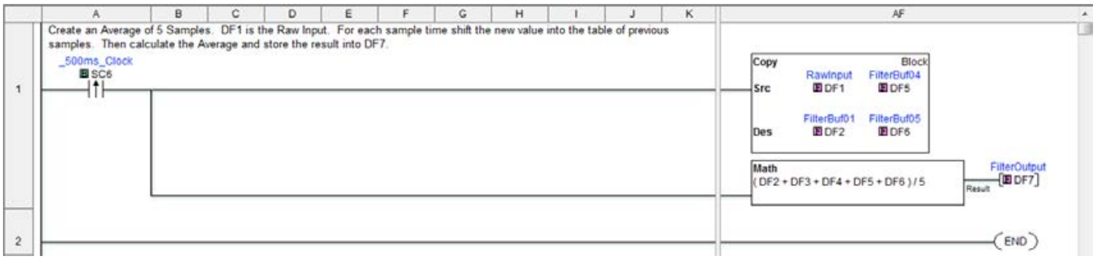
Create a Filter which allows a percent value change for each sample time. DF1 is the Raw Input. DF2 is the previous calculated Output.

In this step response, the Input changes from 0 to 100.



Average Example

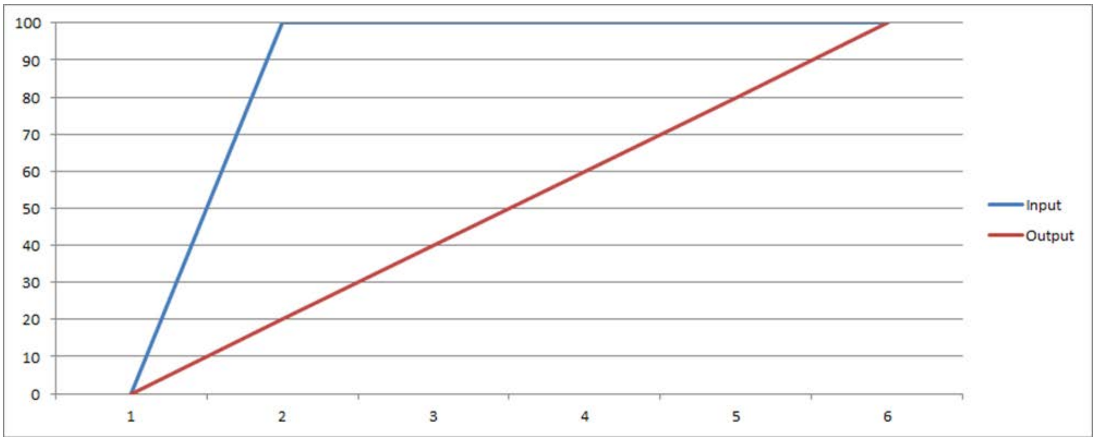
This method enables the flexibility to adjust the sample time and number of samples. At each sample time a new Raw Input is shifted into the group of 5 samples. A new Average is then calculated. In this example, it takes 5 sample times for the Input to reach the Output.



Create an Average of 5 Samples. DF1 is the Raw Input. For each sample time shift the new value into the table of previous samples. Then calculate the Average and store the result into DF7.

```
1  _500ms_Clock SC6 (rising edge)
     Copy Block  Src: RawInput DF1 – FilterBuf04 DF5
                 Des: FilterBuf01 DF2 – FilterBuf05 DF6
     Math ( DF2 + DF3 + DF4 + DF5 + DF6 ) / 5   Result: FilterOutput DF7
2  (END)
```

In this step response, the Input changes from 0 to 100.



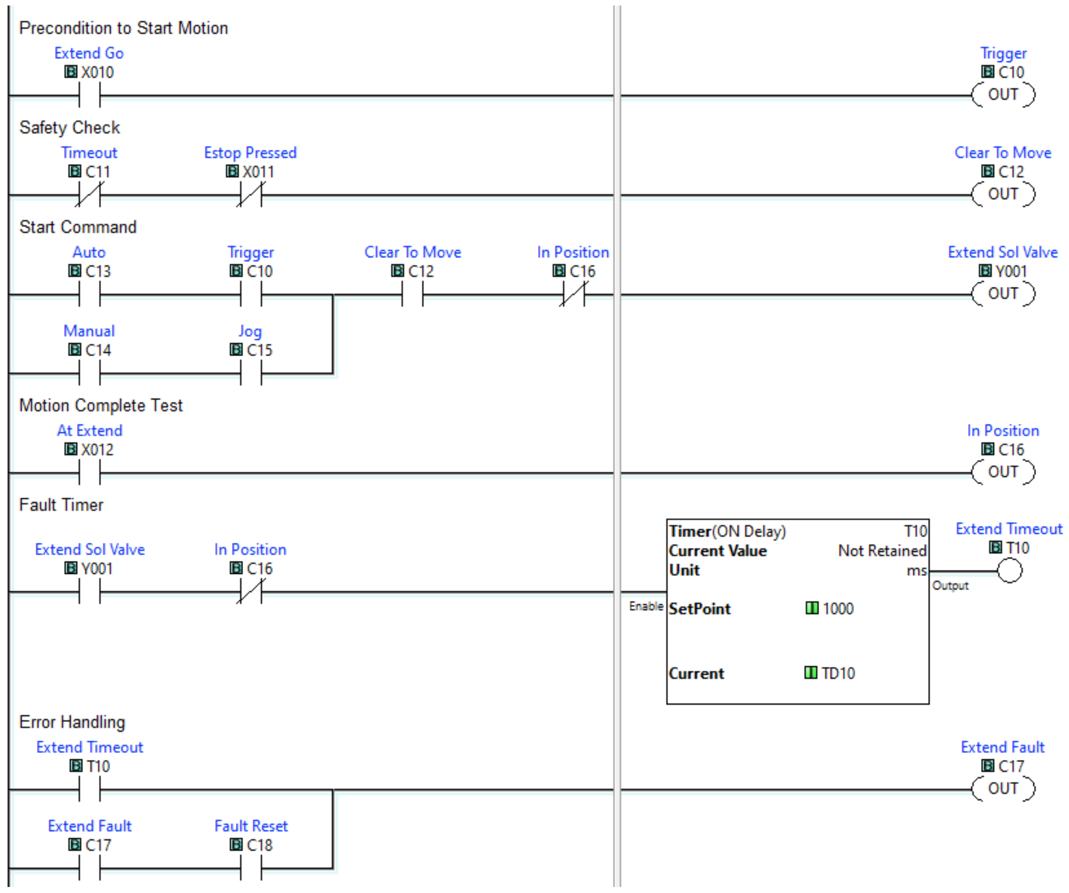
State Machine Programming

Sometimes it is helpful to control a sequence of actions where you have output events that you want to happen and gates that control when those events are complete and its time to move to the next step.

Motion Control

This pattern requires five coils and is sometimes called "Five Rung". It is used whenever an air cylinder or motor will initiate motion in order to ensure safe and complete execution with proper error handling.

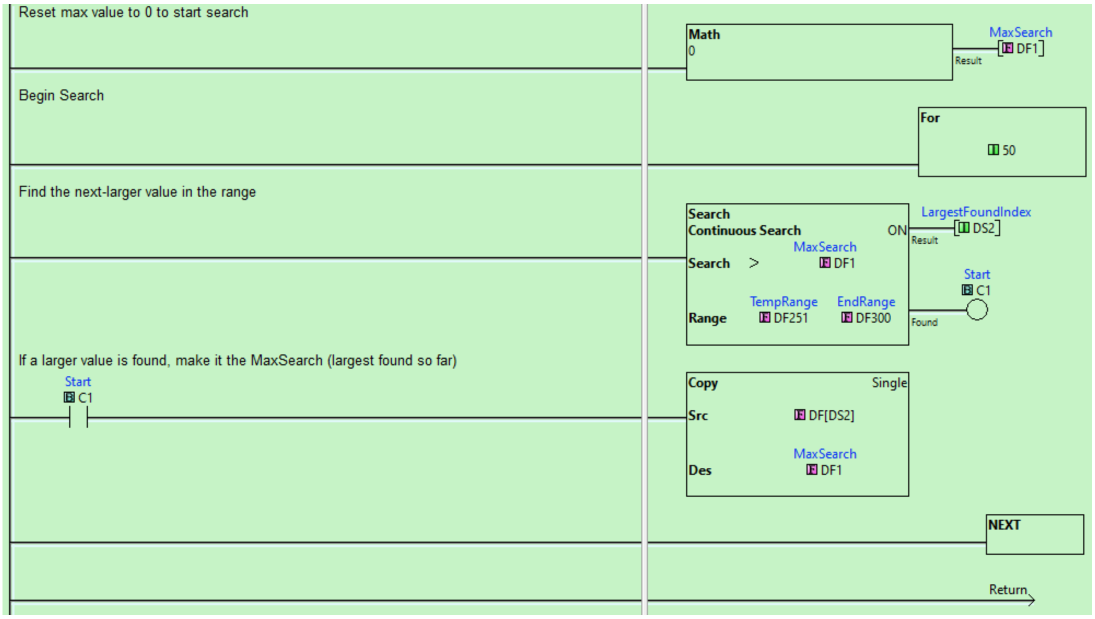
1. Trigger to start the move event
2. Safety check
3. Move Command
4. Move Complete Check
5. Fault test


Precondition to Start Motion
Extend Go
X010
Trigger
C10
OUT
Safety Check
Timeout
C11
Estop Pressed
X011
Clear To Move
C12
OUT
Start Command
Auto
C13
Trigger
C10
Clear To Move
C12
In Position
C16
Extend Sol Valve
Y001
OUT
Manual
C14
Jog
C15
Motion Complete Test
At Extend
X012
In Position
C16
OUT
Fault Timer
Extend Sol Valve
Y001
In Position
C16
Timer(ON Delay) T10
Current Value Not Retained
Unit ms
Enable
SetPoint 1000
Current TD10
Output
Extend Timeout
T10
Error Handling
Extend Timeout
T10
Extend Fault
C17
OUT
Extend Fault
C17
Fault Reset
C18

Advanced Math Logic

Min/Max

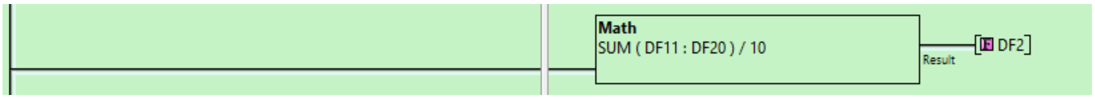
Find a minimum or maximum value from a range of numbers. This example searches a range of 50 values stored in registers DF251-DF300. Each iteration of the loop will look for the next largest number until it can't find any larger number.



```
Reset max value to 0 to start search
    Math
    0                                   Result -> MaxSearch DF1

Begin Search
    For  50

Find the next-larger value in the range
    Search
    Continuous Search                   ON   Result -> LargestFoundIndex DS2
    Search  >   MaxSearch DF1
    Range   TempRange DF251   EndRange DF300   Found -> Start C1 (out)

If a larger value is found, make it the MaxSearch (largest found so far)
  Start C1 --| |--
    Copy                                Single
    Src   DF[DS2]
    Des   MaxSearch DF1

    NEXT

    Return
```

Average

The Math "SUM" command provides an easy start to calculating the average value of a list of values. This capability translates into methods for filtering (see below).



```
Math
SUM ( DF11 : DF20 ) / 10                Result -> DF2
```

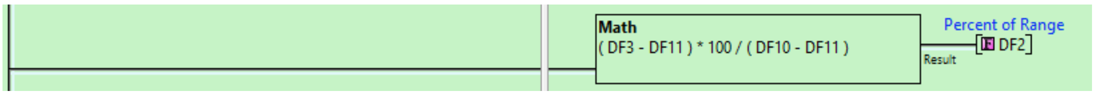
Convert to Percent

This is useful if you want to graph data in C-more with more than one Y-Axis or with Dynamic Y-Axis ranges.

DF3 is the measured value.

DF10 is the Range max value.

DF11 is the Range minimum value.


Math
(DF3 - DF11) * 100 / (DF10 - DF11)
Result
Percent of Range
DF2

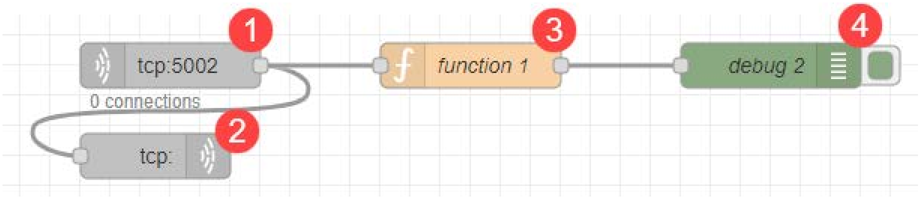
To use Temperature as an example, if measured is 72°F, Max is 212 and min is 32, the percent of range is 22.2% (Coincidentally, that is also the conversion from Fahrenheit to Celsius). This makes it easy to create a C-more chart with dual Y-axes by setting up a chart with a range from 0–100% and then creating a range of values on the right starting at 32 and going up to 212.

Node-RED Project Examples

PROJECT: Initiate a Node-RED flow from a CLICK event

Project Summary:

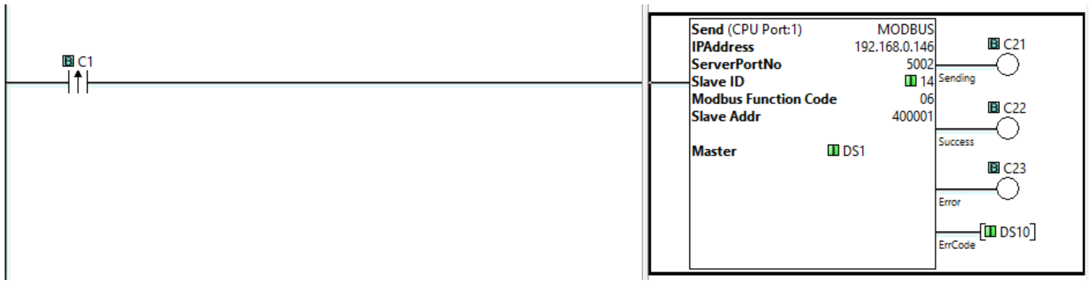
This script and corresponding ladder rung initiates a flow in Node-RED and sends a value from the CLICK to that flow. That value will be 0 or 1 for a boolean value, or a signed int between -32767 and 32767 for an integer.



***NOTE:** The CLICK CPU and the C2-NRED module must be connected to the same Ethernet network in order to communicate via Modbus TCP.*

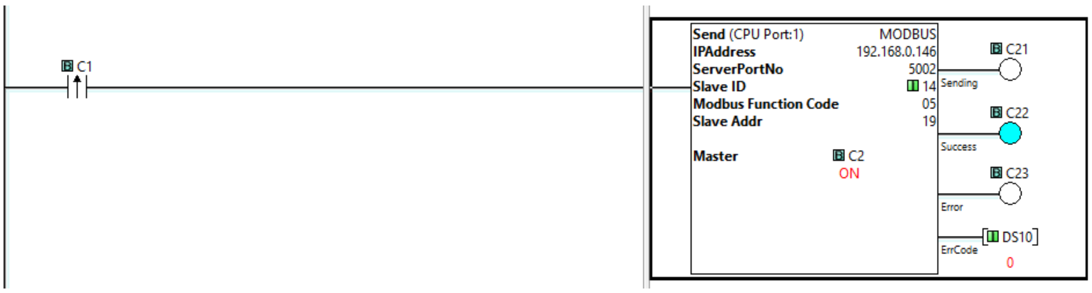
***NOTE:** Node 3 is optional and only necessary if you want to pass data from the CLICK CPU to Node-RED as part of this function. If the CPU is only trying to initiate a flow, only nodes 1 and 2 are needed.*

Send an integer (DS register). Configure the Modbus send as function 05 (Send single Register) using the same port number defined in the Node-RED TCP In node. The Slave ID and slave address are ignored. The data value sent will be the "Starting Master Address"

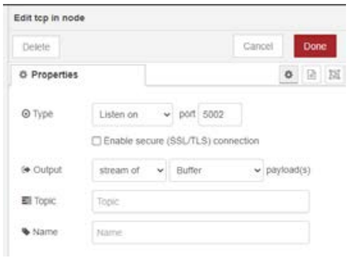


Send (CPU Port:1)	MODBUS
IPAddress	192.168.0.146
ServerPortNo	5002
Slave ID	14
Modbus Function Code	06
Slave Addr	400001
Master	DS1

Send a bit (X, Y, C Register) Configure the Modbus send as function 05 (Send single Register) using the same port number defined in the Node-RED TCP In node. The Slave ID and slave address are ignored. The data value sent will be the "Starting Master Address"



1: TCP IN—Set a port other than the default modbus port. This must be greater than 1024.



Edit tcp in node

Delete | Cancel | Done

Properties

Type: Listen on | port 5002

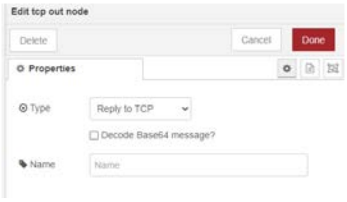
☐ Enable secure (SSL/TLS) connection

Output: stream of | Buffer | payload(s)

Topic: Topic

Name: Name

2: TCP OUT—This node will only Echo the TCP message back to the CLICK. This is default the Modbus expectation.



Edit tcp out node

Delete | Cancel | Done

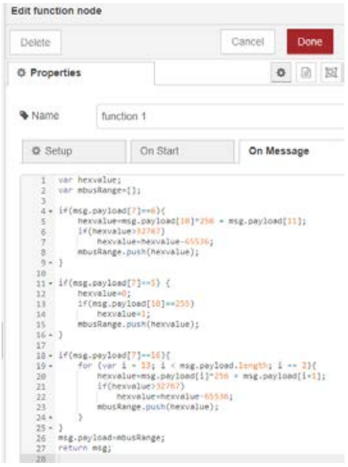
Properties

Type: Reply to TCP

☐ Decode Base64 message?

Name: Name

3: Function—Convert the Hex data sent in the Modbus packet to an array of bits or integers.



Edit function node

Delete | Cancel | Done

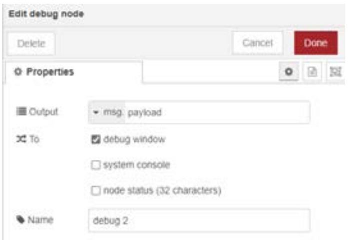
Properties

Name: function 1

Setup | On Start | On Message

```
var hexvalue;
var mbusRange=[];

if(msg.payload[7]==6){
    hexvalue=msg.payload[10]*256 + msg.payload[11];
    if(hexvalue>32767)
        hexvalue=hexvalue-65536;
    mbusRange.push(hexvalue);
}

if(msg.payload[7]==5) {
    hexvalue=0;
    if(msg.payload[10]==255)
        hexvalue=1;
    mbusRange.push(hexvalue);
}

if(msg.payload[7]==16){
    for (var i = 13; i < msg.payload.length; i += 2){
        hexvalue=msg.payload[i]*256 + msg.payload[i+1];
        if(hexvalue>32767)
            hexvalue=hexvalue-65536;
        mbusRange.push(hexvalue);
    }
}
msg.payload=mbusRange;
return msg;

```

4: Display results.



Edit debug node

Delete | Cancel | Done

Properties

Output: msg. payload

To: ☑ debug window

☐ system console

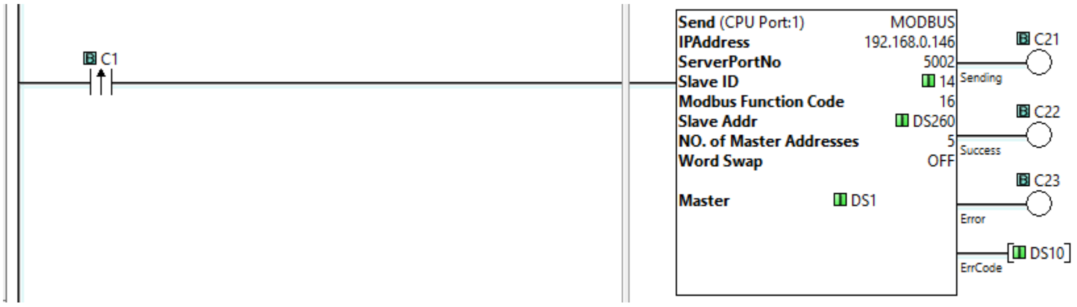
☐ node status (32 characters)

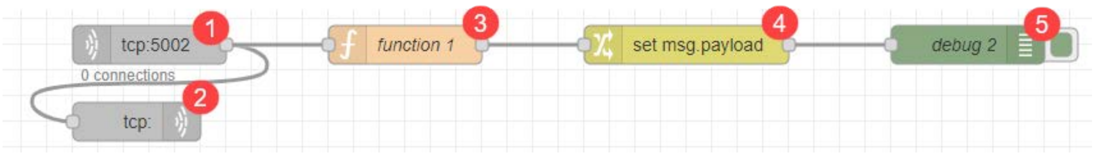
Name: debug 2

Importable Project:

See our repository at https://github.com/AutomationDirect/CLICK-PLC/tree/main/Node-RED for an importable copy of this project.

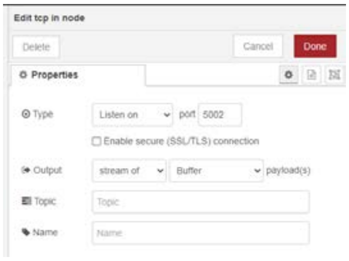
PROJECT: Find the Max Value





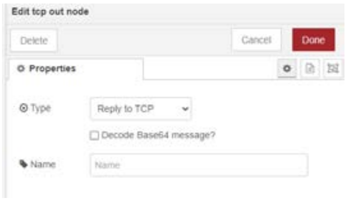
You can pass in a range of numbers using the Modbus Function Code 16 and the following function to convert those values to an array of signed integers. Then use a single node to find the max value in that array. Compare the one rung in this solution to the solution in ladder shown above.

1: TCP IN—Set a port other than the default modbus port. This must be greater than 1024.



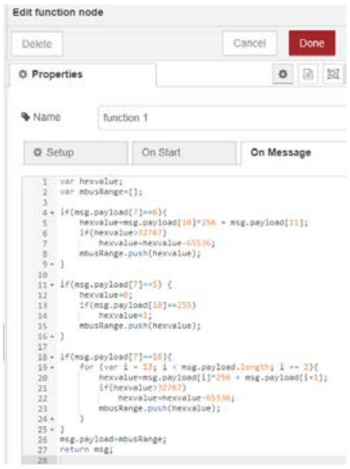
Type: Listen on port 5002
☐ Enable secure (SSL/TLS) connection
Output: stream of Buffer payload(s)
Topic: Topic
Name: Name

2: TCP OUT—This node will only Echo the TCP message back to the CLICK. This is default the Modbus expectation.



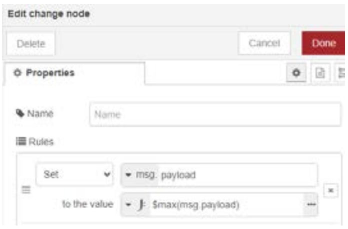
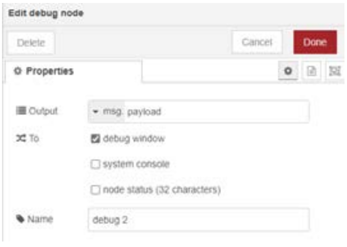
Type: Reply to TCP
☐ Decode Base64 message?
Name: Name

3: Function—Convert the Hex data sent in the Modbus packet to an array of bits or integers.



Name: function 1

```
var hexvalue;
var mbusRange=[];

if(msg.payload[7]==6){
    hexvalue=msg.payload[10]*256 + msg.payload[11];
    if(hexvalue>32767)
        hexvalue=hexvalue-65536;
    mbusRange.push(hexvalue);
}

if(msg.payload[7]==5) {
    hexvalue=0;
    if(msg.payload[10]==255)
        hexvalue=1;
    mbusRange.push(hexvalue);
}

if(msg.payload[7]==16){
    for (var i = 13; i < msg.payload.length; i += 2){
        hexvalue=msg.payload[i]*256 + msg.payload[i+1];
        if(hexvalue>32767)
            hexvalue=hexvalue-65536;
        mbusRange.push(hexvalue);
    }
}
msg.payload=mbusRange;
return msg;
```

4: Use JSONata to find the max value in the array and return an integer.  Edit change node; Name: Name; Rules: Set msg. payload to the value J: $max(msg.payload)	5: Display results.  Edit debug node; Output: msg. payload; To: ☑ debug window ☐ system console ☐ node status (32 characters); Name: debug 2	

Importable Project:

See our repository at https://github.com/AutomationDirect/CLICK-PLC/tree/main/Node-RED for an importable copy of this project.

PROJECT: Find your IP Address

node-red-contrib-ip

Some Services Like AzureSQL require you to setup your C2-NRED's IP Address to be able to connect to their servers as part of a robust security authentication process. This node will return your public IP address.

Download the IP node here:

https://registry.npmjs.org/node-red-contrib-ip/-/node-red-contrib-ip-1.0.1.tgz

PROJECT: Connect to a Microsoft SQL Azure Database

This should only be attempted by (or with) an experienced Database Administrator. Automation Direct will not support database configuration. This guide is a minimal explanation of the basic steps involved. Every database configuration is different and tailored to the security requirements of each individual organization.

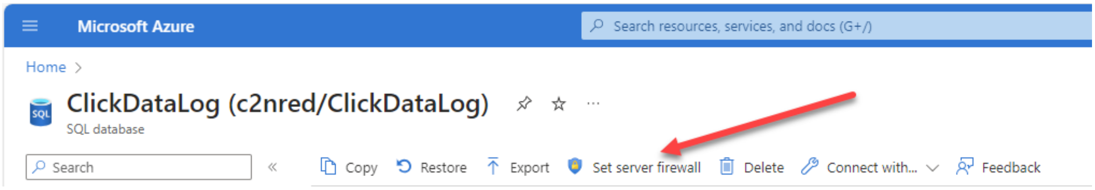
The MSSQL Node found here has been tested to work with Azure:

https://registry.npmjs.org/node-red-contrib-mssql-plus-box/-/node-red-contrib-mssql-plus-box-0.1.4.tgz

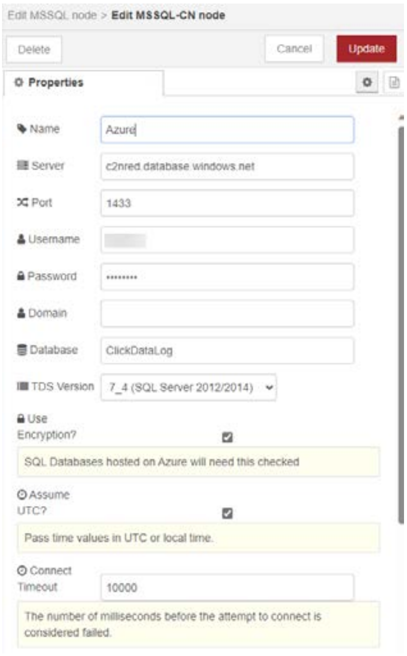
Create a Microsoft Azure Account.

Create an Azure Database – Enable SQL Authentication (ADO.NET).

From the Azure SQL Dashboard, near the top middle of the screen, select "Set Firewall Rule".



Create a Firewall rule in Azure to allow your Node-RED unit to connect to Azure by entering the IP Address of the C2-NRED module. See Instructions to find your IP in the previous example.



Name: Azure
Server: c2nred.database.windows.net
Port: 1433
Username: [redacted]
Password: ••••••••
Domain:
Database: ClickDataLog
TDS Version: 7_4 (SQL Server 2012/2014)
Use Encryption? ✓
SQL Databases hosted on Azure will need this checked
Assume UTC? ✓
Pass time values in UTC or local time.
Connect Timeout: 10000
The number of milliseconds before the attempt to connect is considered failed.

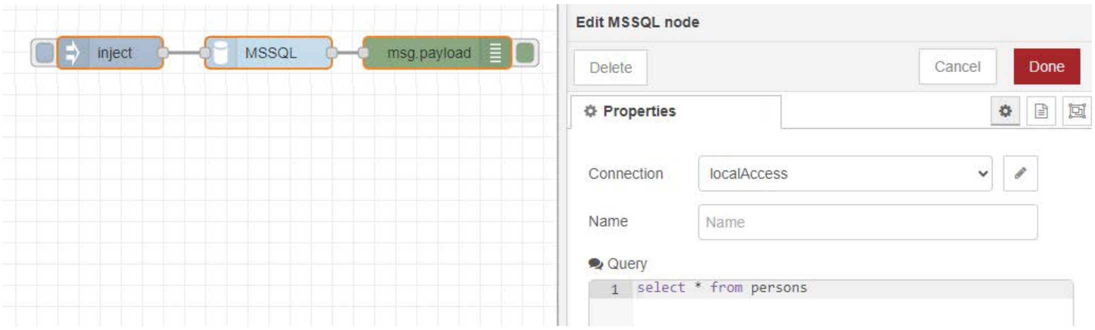
Create test data. The following SQL will create a table named "Persons" in a database named NodeRed. It will insert 4 records into that table that you can use for testing.

```
USE [NodeRed]
GO;

DROP TABLE If EXISTS dbo.Persons

CREATE TABLE Persons (
    Personid int IDENTITY(1,1) PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);

Insert into dbo.Persons
(LastName, FirstName, Age)
Values ('Smith','Andy',25),
('Carson','Kelly',18),
('Nells','Jeff',47),
('Peters','Bill',54);
```

SQL Select

1. The following Select statement will retrieve all the rows in the Persons table

```
Select * from persons
```



Connection: localAccess

Name: Name

Query

```
select * from persons
```

2. That will return a JSON object:

```
[{
   “Personid”:1,
   “LastName”:”Smith”,
   “FirstName”:”Andy”,
   “Age”:25
 },
 {
   “Personid”:2,
   “LastName”:”Carson”,
   “FirstName”:”Kelly”,
   “Age”:18
},
{
   “Personid”:3,
   “LastName”:”Nells”,
   “FirstName”:”Jeff”,
   “Age”:47
},
{
   “Personid”:4,
   “LastName”:”Thompson”,
   “FirstName”:”Bill”,
   “Age”:54
}]”
```

3. Notice, it is an array of objects, so there is one object per array element returned in the query. Each object has a Key:value pair. This enables Node-RED to address these using JSONata or JavaScript using dot notation. For example, msg.payload[1].LastName will return “Carson”.

SQL Update

1. The following update will change the First name of personid=4 to Larry:

```
Update NodeRed.dbo.Persons
set FirstName=’Larry’
Where Personid=4;
```

2. The return value (msg.payload) will be null.

SQL Insert

1. The following insert will add a new person to the Persons table:

```
Insert into dbo.Persons
(LastName, FirstName,Age)
Values ('Brooks','Paul',25)
```

2. The return value (msg.payload) will be null.

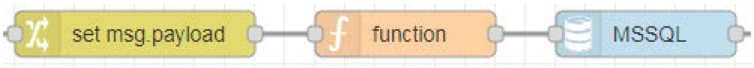
SQL Delete

1. The following SQL will delete the person with Last Name='Nells'.

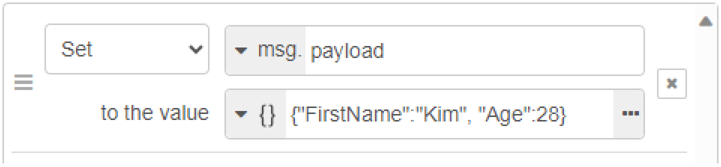
```
Delete from persons
where LastName='Nells'
```

2. The return value (msg.payload) will be null.

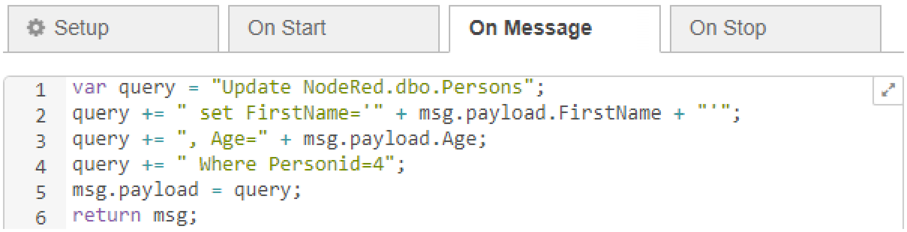
You may notice these are all static queries. There is no variable information in the query, so they are great for testing, but don't provide much practical value. In order to add variables, we need to add a couple of nodes to our flow:



In this flow, the set msg.payload creates a JSON Object with 2 key value pairs representing our variables


Set msg. payload
to the value {} {"FirstName":"Kim", "Age":28}

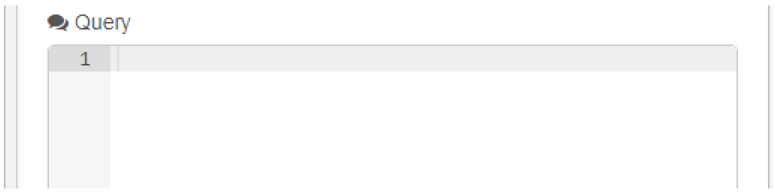
The function node creates a query string using those variables.



```
var query = "Update NodeRed.dbo.Persons";
query += " set FirstName='" + msg.payload.FirstName + "'";
query += ", Age=" + msg.payload.Age;
query += " Where Personid=4";
msg.payload = query;
return msg;
```

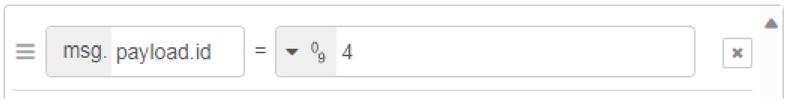
This is JavaScript and is adding together String values including the variables passed in from the prior Set node.

Finally, the MSSQL node is blank. It has a rule that if you don't give it anything to run, it will run whatever you pass in through the msg.payload value:



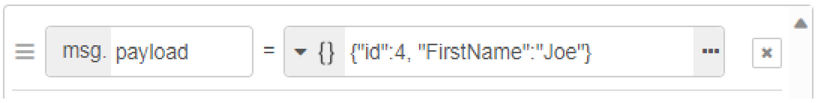
In this example, the Function node enables you to create whatever query you want with all the variables you can pass it. For instance, if you have a barcode reader connected to your CLICK PLUS PLC, you could have a CLICK Read node, read the barcode value and then create a query to look up the barcode and return a Bin number and quantity so your automated pallet system can retrieve the proper bin and number of parts.

An alternate approach which doesn't use JavaScript is to pass a value into the query. This can be an object if you have multiple values:



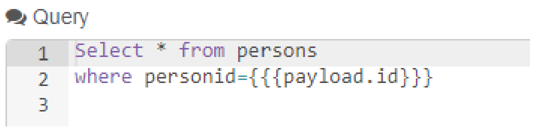
msg. payload.id = 4

or



msg. payload = {"id":4, "FirstName":"Joe"}

Then in the query you can use the "Mustache" format to extract the object's values and inject them into the query:



```
Select * from persons
where personid={{{payload.id}}}
```

This example in conjunction with the "Initiate a flow from a CLICK" and the "Find your IP" provides everything needed for a system to "Send" a recipe ID from a CLICK, start a flow which runs a query to retrieve a recipe from SQL. The next step would be to use a CLICK Write node to write the recipe back to the CLICK. These steps enable a system with machines distributed around the country to use a common data set that can be administered from a central point.

PROJECT: Connect to the local SQLite Database

The C2-NRED comes with a SQLite database preinstalled as part of the firmware. SQLite, as the name suggests, is a lightweight single file database that follows standard SQL query rules. It is important to note there are several pros and cons of using an on-device database instead of connecting to a remote SQL database.

Pros and Cons of using a local database:

Pros	Cons
Development speed – there is nothing additional you need to install.	SQLite is a single connection DB – It is not thread safe so only the Node-RED application should read/write to it. No concurrent users or external connections are allowed.
No additional database server so you can save hardware costs, software costs, and electricity costs associated with servers.	There are no tools available to backup or restore the data stored in the C2-NRED memory. It will be the user's responsibility to write any code necessary to generate backup files.
Read/write data regardless of network availability.	The data in the DB is only available to the host Node-RED (without custom code to expose it).
Read/write speeds are not network dependent.	The CLICK Project Loader does not backup user files.
If writing to the SD Card, up to 32gb of storage is available.	Solid state memory has limited read/write cycles and DB can wear them out prematurely.
Possible to remove the SD Card and read/write the DB from a PC. (You must power cycle the PLC to reconnect to a DB on a SD card.)	

Frequently Asked Questions:

Can the DB be written to the SD card and read /written by an instance on the PC?

YES, but if a DB is on an SD Card, after that card is removed, the CLICK must be restarted to reconnect to the DB on the card.

Is there a network path to the user directory on the C2-NRed?

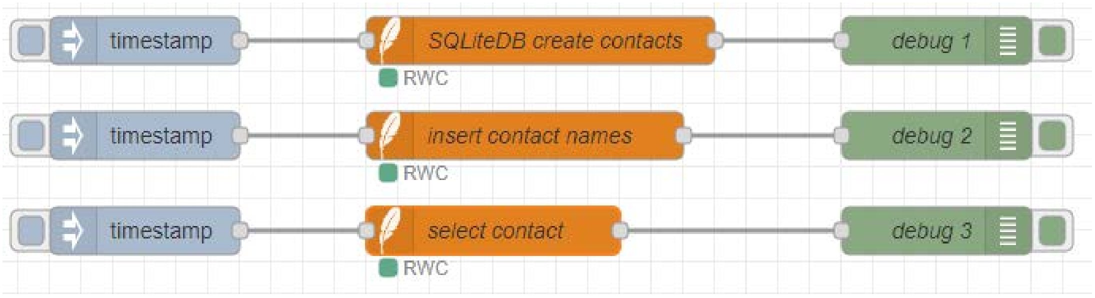
No, this would require software running in Linux like Samba (software that provides access to SMB/CIFS protocols used by Windows). See How to Share Files Between Windows and Linux (howtogeek.com) for more information.

Will files be backed up and restored by the Project Loader tool for CLICK?

No.

Minimum configuration to use SQLite

Using the SQLite module includes a minimum of three nodes:

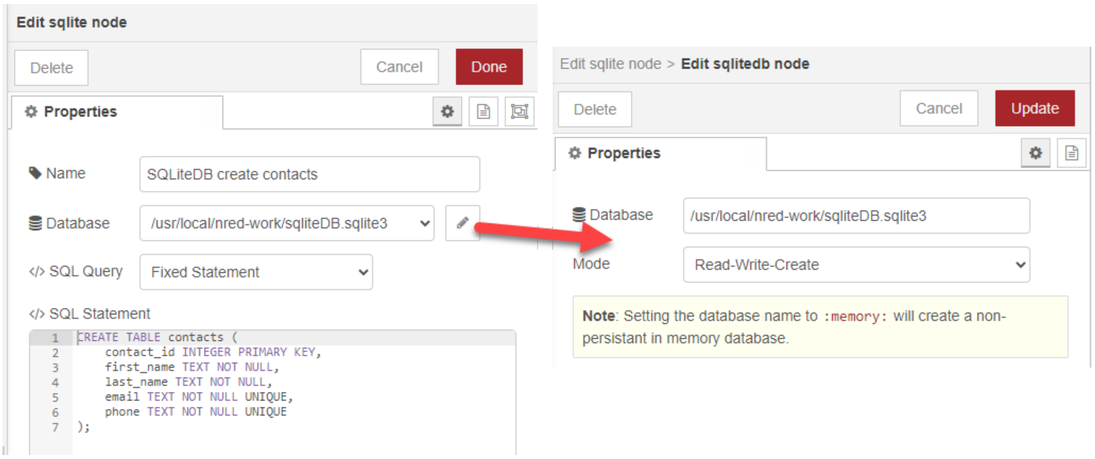


First, create a database.

There are no extra steps required to install or configure the DB. You simply drop a sqlite node from the storage group onto your canvas. Click the pencil icon to set the DB file path and name. There are only two choices of file path (parent directory):

1. Local storage: /usr/local/nred-work/
2. SD Card storage: /run/media/mmcblk0p1/

The database name and extension can be anything you want, but it is recommended to use .sqlite3 as the extension to make it easy to identify.



Name: SQLiteDB create contacts

Database: /usr/local/nred-work/sqliteDB.sqlite3

SQL Query: Fixed Statement

SQL Statement

```
CREATE TABLE contacts (
    contact_id INTEGER PRIMARY KEY,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    phone TEXT NOT NULL UNIQUE
);
```

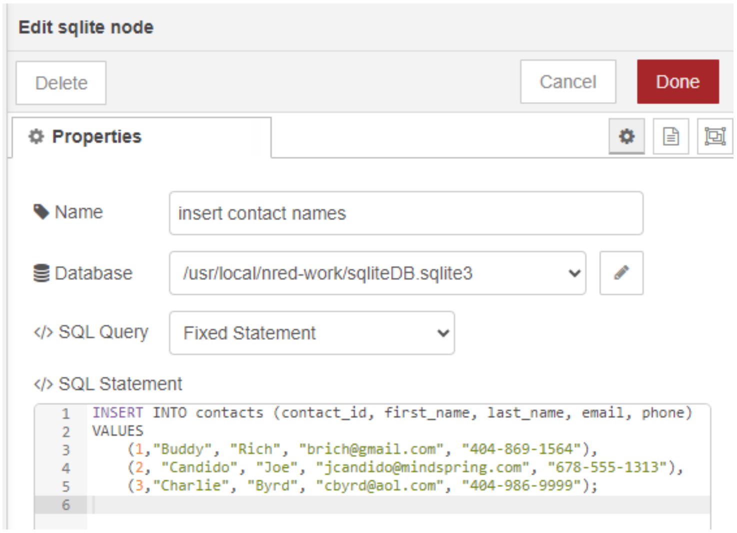
Database: /usr/local/nred-work/sqliteDB.sqlite3

Mode: Read-Write-Create

Note: Setting the database name to `:memory:` will create a non-persistant in memory database.

You will also need to set the mode to "Read-Write-Create" to enable the creation of a new DB. Click update to close the configuration window. In the SQL statement panel, you will want to create your table(s).

Use a second node to populate your table:



Name: insert contact names

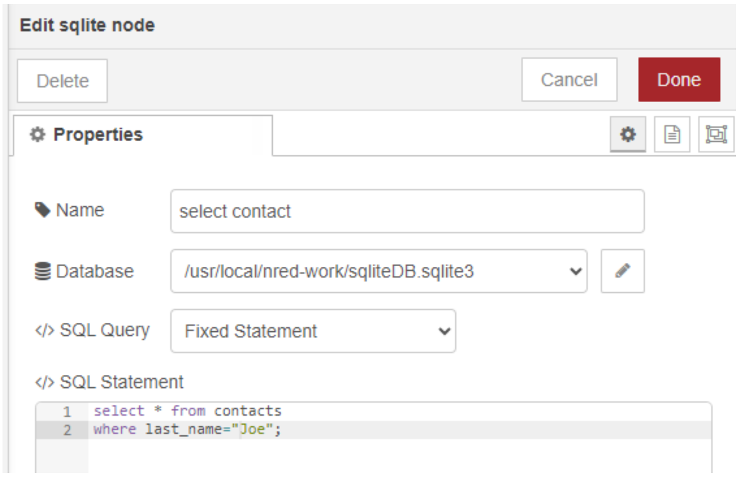
Database: /usr/local/nred-work/sqliteDB.sqlite3

SQL Query: Fixed Statement

SQL Statement

```
INSERT INTO contacts (contact_id, first_name, last_name, email, phone)
VALUES
    (1,"Buddy", "Rich", "brich@gmail.com", "404-869-1564"),
    (2, "Candido", "Joe", "jcandido@mindspring.com", "678-555-1313"),
    (3,"Charlie", "Byrd", "cbyrd@aol.com", "404-986-9999");
```

Use a third node to retrieve data from the table you just created and populated:



Name: select contact

Database: /usr/local/nred-work/sqliteDB.sqlite3

SQL Query: Fixed Statement

SQL Statement

```
select * from contacts
where last_name="Joe";
```

It is possible to create the Database file using a SQLite editor on your PC (e.g https://sqlitebrowser.org/). Write the DBfile to the correct path on the SD card, and insert it into the SD card slot on the Node-RED. However, you must cycle power to your PLC to get it to connect to the SD card slot and read the data.

Additional information can be found at:

https://www.sqlitetutorial.net/ or SQLite tutorial - W3schools

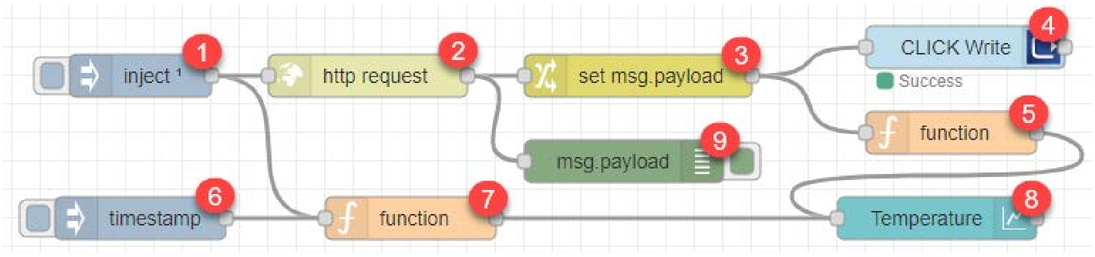
PROJECT: Call a Web Service to retrieve data and chart the results

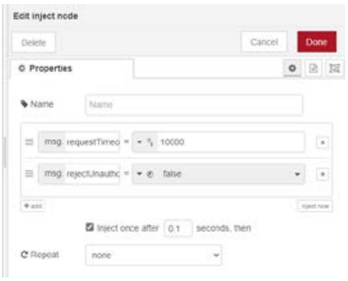
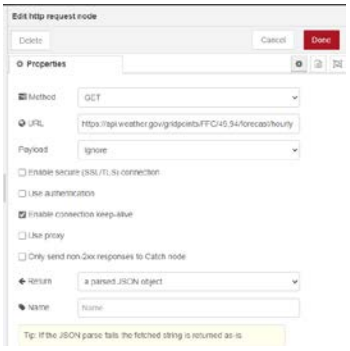
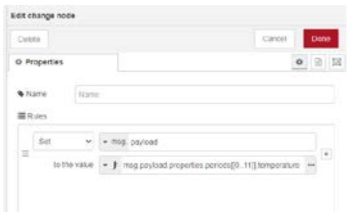
Project Summary:

This script initiates an HTTP request to a weather API. When it receives the response as JSON, it uses JSONata to strip out most of the data and is left with Time and Temperature for the next 12 hours. It writes those values to the CLICK and sends them to a function where the data is formatted as a line graph. In parallel, there is a second thread that calls a function to zero out the line graph.

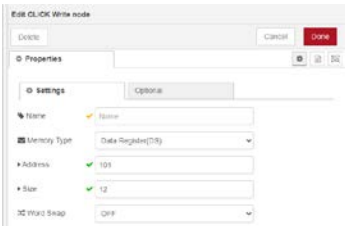
3rd Party Modules required: None

Skill Level: Beginner



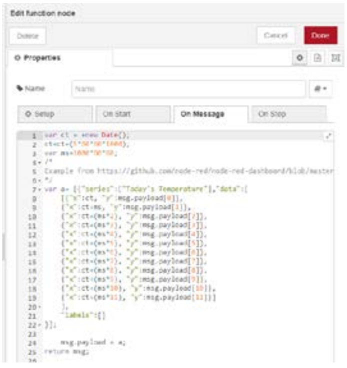
1: Inject.  Edit inject node; Name: (empty); msg. requestTimeo = 10000; msg. rejectUnauthc = false; ☑ Inject once after 0.1 seconds, then; Repeat: none	2: HTTP Request.  Edit http request node; Method: GET; URL: https://api.weather.gov/gridpoints/FFC/45,94/forecast/hourly; Payload: ignore; ☐ Enable secure (SSL/TLS) connection; ☐ Use authentication; ☑ Enable connection keep-alive; ☐ Use proxy; ☐ Only send non-2xx responses to Catch node; Return: a parsed JSON object; Name: (empty); Tip: If the JSON parse fails the fetched string is returned as-is	3: Set Payload – Use JSONata to strip everything but time and temp from the web service response.  Edit change node; Name: (empty); Rules: Set msg. payload to the value msg.payload.properties.periods[[0..11]].temperature

4: Write Temperature to the CLICK.


Edit CLICK Write node

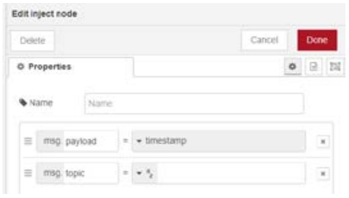
Name: Name
Memory Type: Data Register(DS)
Address: 101
Size: 12
Word Swap: OFF

5: Script.


Edit function node

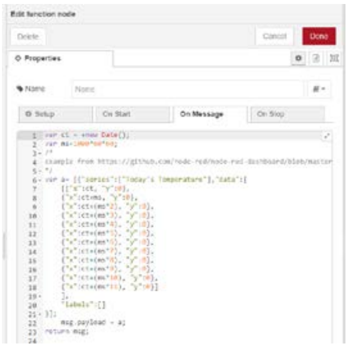
```
var ct = +new Date();
ct=ct-(5*60*60*1000);
var ms=1000*60*60;
/*
Example from https://github.com/node-red/node-red-dashboard/blob/master
*/
var a= [{"series":["Today's Temperature"],"data":[
    [{"x":ct, "y":msg.payload[0]},
    {"x":ct+ms, "y":msg.payload[1]},
    {"x":ct+(ms*2), "y":msg.payload[2]},
    {"x":ct+(ms*3), "y":msg.payload[3]},
    {"x":ct+(ms*4), "y":msg.payload[4]},
    {"x":ct+(ms*5), "y":msg.payload[5]},
    {"x":ct+(ms*6), "y":msg.payload[6]},
    {"x":ct+(ms*7), "y":msg.payload[7]},
    {"x":ct+(ms*8), "y":msg.payload[8]},
    {"x":ct+(ms*9), "y":msg.payload[9]},
    {"x":ct+(ms*10), "y":msg.payload[10]},
    {"x":ct+(ms*11), "y":msg.payload[11]}]
    ],
    "labels":[]
}];

    msg.payload = a;
return msg;
```

6: Inject.


Edit inject node

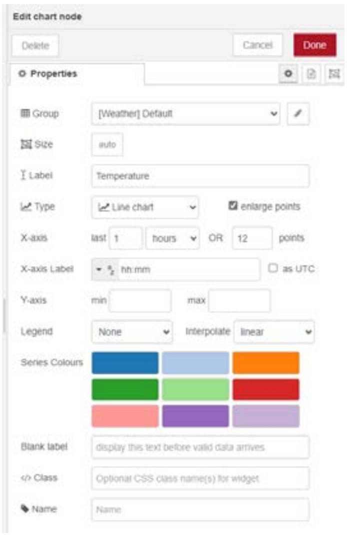
Name: Name
msg. payload = timestamp
msg. topic =

7: Script.


Edit function node

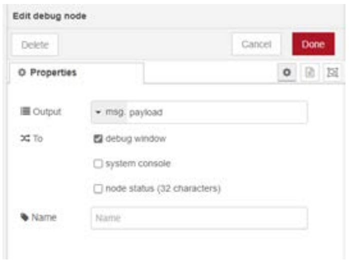
```
var ct = +new Date();
var ms=1000*60*60;
/*
Example from https://github.com/node-red/node-red-dashboard/blob/master
*/
var a= [{"series":["Today's Temperature"],"data":[
    [{"x":ct, "y":0},
    {"x":ct+ms, "y":0},
    {"x":ct+(ms*2), "y":0},
    {"x":ct+(ms*3), "y":0},
    {"x":ct+(ms*4), "y":0},
    {"x":ct+(ms*5), "y":0},
    {"x":ct+(ms*6), "y":0},
    {"x":ct+(ms*7), "y":0},
    {"x":ct+(ms*8), "y":0},
    {"x":ct+(ms*9), "y":0},
    {"x":ct+(ms*10), "y":0},
    {"x":ct+(ms*11), "y":0}]
    ],
    "labels":[]
}];
    msg.payload = a;
return msg;
```

8: Chart.


Edit chart node

Group: [Weather] Default
Size: auto
Label: Temperature
Type: Line chart ☑ enlarge points
X-axis: last 1 hours OR 12 points
X-axis Label: hh:mm ☐ as UTC
Y-axis: min max
Legend: None Interpolate: linear
Series Colours: 🟦 🟦 🟧 🟩 🟩 🟥 🟥 🟪 🟪
Blank label: display this text before valid data arrives
Class: Optional CSS class name(s) for widget
Name: Name

9: Debug.

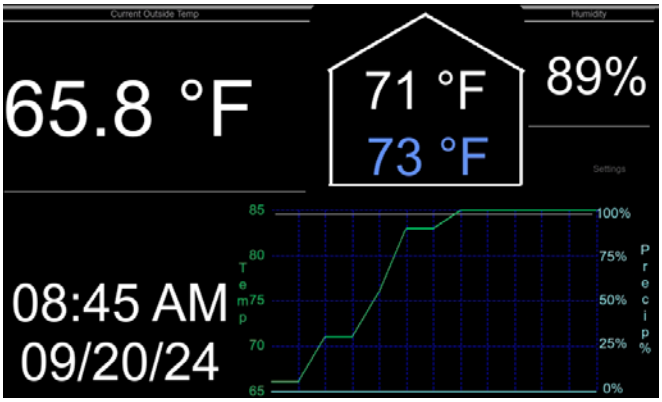

Edit debug node

Output: msg. payload
To: ☑ debug window
☐ system console
☐ node status (32 characters)
Name: Name

Importable Project

See our repository at https://github.com/AutomationDirect/CLICK-PLC/tree/main/Node-RED for an importable copy of this project.

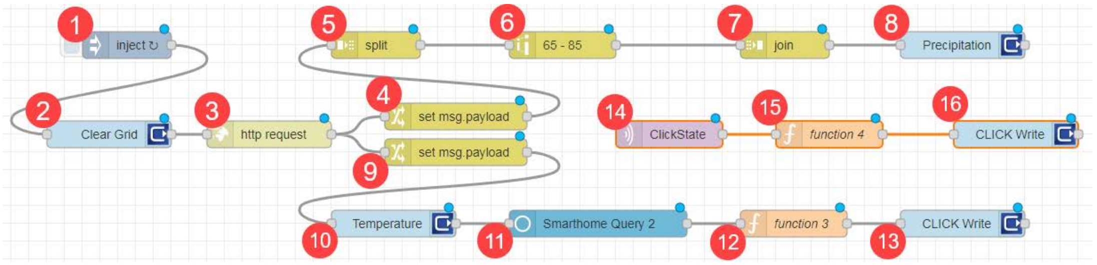
PROJECT: Local Weather Station



Technologies Implemented:

- Shelly Bluetooth-Temp and Humidity sensor
- Shelly Bluetooth-Wi-Fi Gateway
- MQTT over Wi-Fi
- Node-RED Hosting MQTT Broker
- Node-RED MQTT Client
- Node-RED HTTP Request to api.weather.gov to retrieve forecast
- Node-RED Alexa Nodes to integrate with Alexa, and Nest Thermostats
- JSONata to parse the JSON response from API.WEATHER.GOV
- CLICK PLC to process the data and communicate with CM5 HMI
- CM5 HMI for data display
- CM5 NTP Server to get time from the cloud

- CM5 Webserver to make the data available - securely – from anywhere.



1. Inject – start the Flow and rerun it every 10 minutes
2. Send a bit to CLICK To start charting incoming data
3. Http request – call https://api.weather.gov to get the 7 day forecast in JSON format
4. Use JSONata to parse the api response into an array of precip % values
5. Split the array into individual values and stream them out
6. Scale each precip value from 0-100 to 65-85 for charting
7. Rejoin the scaled data into an array
8. Sent the array of precip data to CLICK as a range of integers
9. Use JSONata to parse the api response into an array of temperature values
10. Send the temperatures directly to CLICK
11. Use ALEXA to query my NEST Thermostat. Google's Security requires a cloud-based server to access devices. This approach uses Amazon to access the device, then queries Amazon for the data.
12. Use a JavaScript function to convert Amazon's Object based response into an array.
13. Send the inside temp and the Nest setpoint to CLICK
14. The Shelly sensor is connected to the Shelly 1 gateway – the Gateway broadcasts temp, humidity, and sensor battery state every minute or when the data changes using MQTT. The "ClickState" node receives the MQTT message
15. This function parses the temp, converts it to Fahrenheit, and sends all three values as an array
16. Writes the temp, humidity and battery to the CLICK

Upon The CLICK receiving all the data, it starts a pulse signal for 13 pulses. Each rise of the pulse copies the next data value into a register that the Cmore will plot. Each fall of the pulse triggers C-more to plot the value.

Weather data is available via FREE API at: https://api.weather.gov/

Documentation is available at: https://www.weather.gov/documentation/services-web-api#/

The forecast info in this example can be found by running the Get /point/point API in the Swagger tool.

1. Open Google maps and find the location you are testing.
2. Right click to see the latitude and longitude of that location

3. Single click on those values to copy them to your clipboard
4. Paste those values into the String field for the point API
5. Remove the space after the comma
6. Execute

Look below in the response body field for the following values:

1. gridId
2. gridX
3. gridY
4. URLs for Forecast data, weatherstation information and more

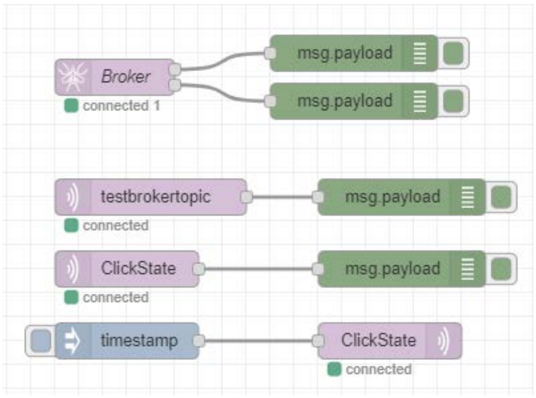
PROJECT: Create an MQTT Broker

Use the module node-red-contrib-aedes (MQTT Broker) (0.11.1) available at the following link:

https://registry.npmjs.org/node-red-contrib-aedes/-/node-red-contrib-aedes-0.11.1.tgz

This flow allows the C2-NRED module to act as an MQTT Broker, Subscriber and Publisher simultaneously. Even if you don't need MQTT, this flow enables the CLICK PLUS to trigger events in Node-RED. For example, if you have a sensor on an overflow gate, that sensor can trigger an MQTT message which will in turn trigger a Node-RED flow which can send a text message or publish the event to a web server.

For troubleshooting MQTT connections, an external client like MQTTX: Your All-in-one MQTT Client Toolbox is recommended.

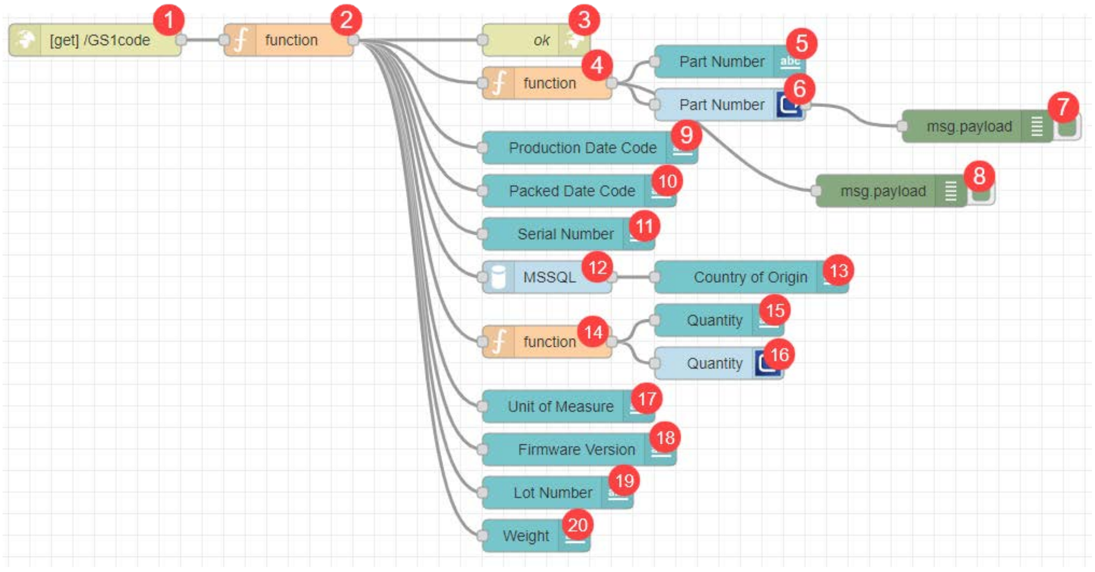


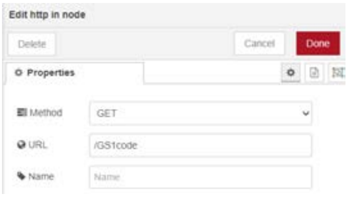
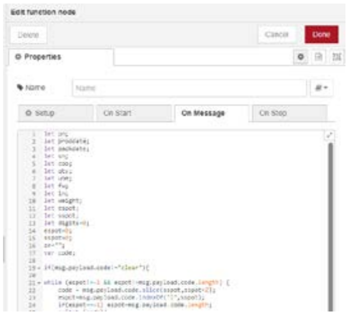
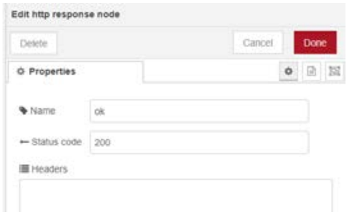
The AEDES Broker node defines the broker configuration. The gray MQTT Broker defines the connection for the publisher and subscriber nodes (mqtt in and mqtt out). The green debug nodes would be replaced with process nodes to handle the output from the MQTT subscribe nodes. The 2 debug nodes connected to the AEDES Broker are entirely optional to view events occurring within the broker. In this example, the C2-NRED's IP address is 192.168.0.115 and the Broker is monitoring port 1883.

Importable Project

See our repository at https://github.com/AutomationDirect/CLICK-PLC/tree/main/Node-RED for an importable copy of this project.

PROJECT: Send a Barcode value to Node-RED via HTTP Get, parse it into fields on the dashboard, and perform a SQL Lookup



1: http in ([get] /GS1code)	2: function	3: http response (ok)
Request page with http://192.168.0.56:1880/GS1code?code=XXX In Node-RED, the node will pass msg.payload.code=XXX.	See Source code below. This JavaScript block parses a GS1 Barcode into 10 distinct data vales.	A 200 Status code means the request was successfully received.
		

Edit http in node

Delete | Cancel | Done

Properties

Method: GET

URL: /GS1code

Name: Name

Edit function node

Delete | Cancel | Done

Properties

Name: Name

Setup | On Start | On Message | On Stop

```
let pn;
let proddate;
let packdate;
let sn;
let coo;
let qty;
let uom;
let fwv;
let ln;
let weight;
let espot;
let sspot;
let digits=0;
espot=0;
sspot=0;
ss="";
var code;

if(msg.payload.code!=="clear"){

while (espot!=-1 && espot!=msg.payload.code.length) {
    code = msg.payload.code.slice(sspot,sspot+2);
    espot=msg.payload.code.indexOf("(",sspot);
    if(espot==-1) espot=msg.payload.code.length;
```

Edit http response node

Delete | Cancel | Done

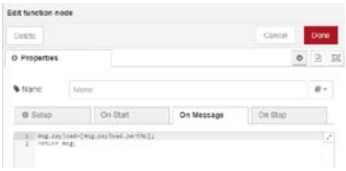
Properties

Name: ok

Status code: 200

Headers

4: function

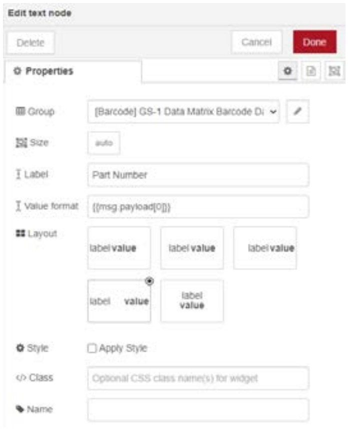

Edit function node

Name: Name

Setup | On Start | On Message | On Stop

```
msg.payload=[msg.payload.partNo];
return msg;
```

5: text (Part Number)


Edit text node

Group: [Barcode] GS-1 Data Matrix Barcode Di

Size: auto

Label: Part Number

Value format: {{msg.payload[0]}}

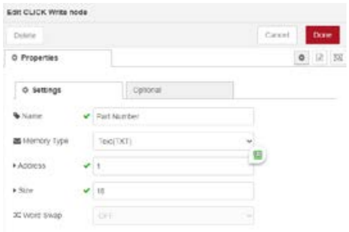
Layout: label value | label value | label value | label value | label value

Style: ☐ Apply Style

Class: Optional CSS class name(s) for widget

Name:

6: CLICK Write (Part Number)


Edit CLICK Write node

Settings | Optional

Name: Part Number

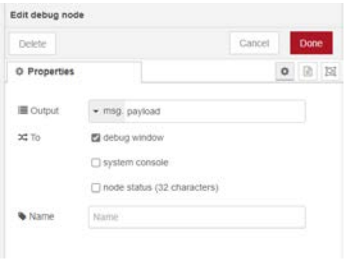
Memory Type: Text(TXT)

Address: 1

Size: 18

Word Swap: OFF

7: Debug.


Edit debug node

Output: msg. payload

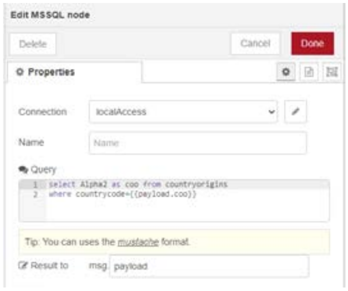
To: ☑ debug window

☐ system console

☐ node status (32 characters)

Name: Name

12: MSSQL


Edit MSSQL node

Connection: localAccess

Name: Name

Query:

```
select Alpha2 as coo from countryorigins
where countrycode={{payload.coo}}
```

Tip: You can uses the mustache format.

Result to: msg. payload

Repetitive nodes omitted for brevity.

1. This creates a web page at Http://[node red IP Address]/GS1code?data
   Calling the URL (for example http://192.168.0.5:1880/gs1code?barcodedata) will initiate the flow and put the barcode data in the msg.payload.
2. This JavaScript function parses the barcode into an object containing its 10 parameters.
3. Respond to the HTTP request with a 200.
4. This JavaScript function pulls out the part number object so that it can be written to the CLICK.
5. This dashboard node formats the Part number, adds a label and displays the part number on the dashboard. Nodes 9-11, 13, 15, and 17-20 do the same.
6. Write the part number to CLICK.
7. A debug statement to help troubleshoot.
12. This MSSQL statement takes a country code from the barcode and runs a query against a remote SQL database to look up the country of origin.

Importable Project:

See our repository at https://github.com/AutomationDirect/CLICK-PLC/tree/main/Node-RED for an importable copy of this project.

BONUS FUNCTION: Process Modbus - Four message types using JavaScript to create array of values:

```
var hexvalue;
var mbusRange=[];
var i;
//Single Register to signed INT
if(msg.payload[7]==6){              //Message type 06
    hexvalue=msg.payload[10]*256 + msg.payload[11];
    if(hexvalue>32767)
        hexvalue=hexvalue-65536;   //convert unsigned int to signed int
    mbusRange.push(hexvalue);
}
//Single Coil to array of binary number
if(msg.payload[7]==5) {             //Message type 05
    hexvalue=0;                     //default to 0
    if(msg.payload[10]==255)        //if bit is high, set to 1
        hexvalue=1;
    mbusRange.push(hexvalue);
}
//Multiple Coils to array of binary numbers
if(msg.payload[7]==15){             //Message type 15
    for (i = 13; i < msg.payload.length; i += 1){
        hexvalue=msg.payload[i].toString(2).padStart(8, ‘0’);
        mbusRange=hexvalue.split(“”).concat(mbusRange);
        mbusRange=mbusRange.map(Number);
    }
}
//Multiple registers to array of signed INT
if(msg.payload[7]==16){             //Message type 16
    for (i = 13; i < msg.payload.length; i += 2){
        hexvalue=msg.payload[i]*256 + msg.payload[i+1];
        if(hexvalue>32767)
            hexvalue=hexvalue-65536;
        mbusRange.push(hexvalue);
    }
}
msg.payload=mbusRange;
return msg;
```