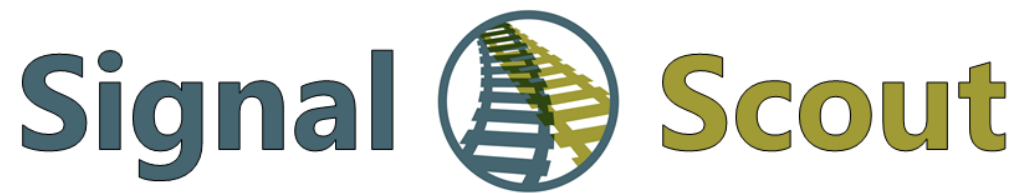


**Semaphore Software LLC**



**Nomenclature Definiton Files**

June 21, 2023

## Table of Contents

1	Introduction .....	1
2	Configuration and Setup .....	1
2.1	File Placement .....	1
2.2	File Selection .....	1
2.2.1	Step-by-Step .....	2
3	Creating an NDF File .....	3
3.1	File Format .....	3
3.2	Parts of the Expression .....	3
3.2.1	Line Format .....	3
3.2.2	Regular Expression String .....	4
3.2.3	Definition Text .....	4
3.3	Multiple Matches .....	4
4	Using Nomenclature Definitions .....	5
5	Example of a partial NDF File .....	6

## 1 Introduction





Signal Scout features the ability to provide detailed information about signal variable acronyms and their associated nomenclature. This feature makes use of a pattern recognition format called “Regular Expressions” to detect and define sequences of characters. Definitions are generalized to allow for less up-front work of defining every possible combination, while also providing adequate information so that it is useful and informative to the user.

## 2 Configuration and Setup

Signal Scout makes use of specially formatted and named files to utilize the Nomenclature Definition function. The following sections will describe how to create and modify these files. This section will explain how to use these files from within Signal Scout.

### 2.1 File Placement

Nomenclature Definition Files are identified for Signal Scout’s use by utilizing a “.ndf” file extension (NDF file). NDF files must also be placed within a specific folder.

 logs	6/2/2021 11:21 AM	File folder
 nom_def	6/2/2021 11:21 AM	File folder
 Signal Scout.exe	4/15/2021 7:25 PM	Application
 sss.ini	6/1/2021 9:45 AM	INI File

A folder named “nom\_def” will be created in the same location as the “Signal Scout.exe” file is located. Within the “nom\_def” folder the user may place any number of NDF files, as shown below.

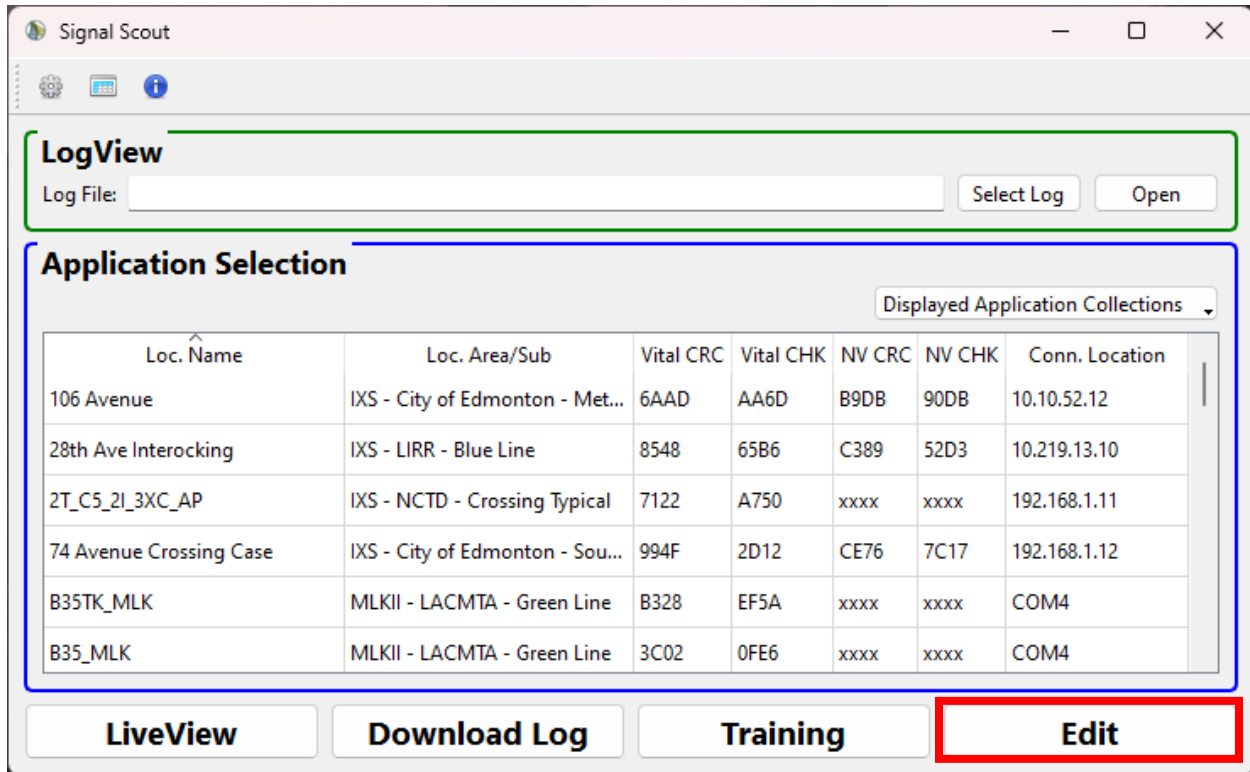
 NCTD - San Diego.ndf	12/12/2020 11:41 AM	NDF File
 RTD - North Metro.ndf	11/28/2020 6:25 PM	NDF File

### 2.2 File Selection

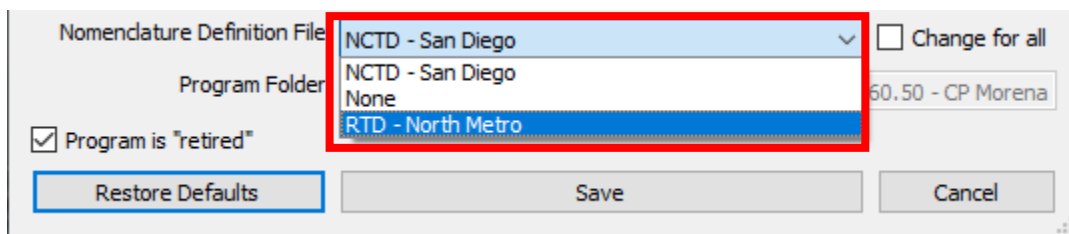
Once the proper folder structure has been created and files placed as described above, the user must select which NDF file to associate with each program. Signal Scout gives the user the ability to create multiple NDF files with different definition information. This allows for railroads that have different lines or subdivisions that have varying types of logic and naming to be customized and prevent any overlap or confusion. For railroads that have consistent nomenclature use across their property, it is expected to only need one (1) NDF file.

### 2.2.1 Step-by-Step

- Open Signal Scout and left click select the program you wish to associate an NDF file with from the main screen.
- After selecting the program, click the “Edit” button.



- Clicking the Edit button will bring up the Edit Application Information Window. Near the bottom of this window there is a row that starts with “Nomenclature Definition File”. Select the drop-down arrow. The selections should match the names of the files placed into the “nom\_def” folder, as well as the “None” selection.
- By left clicking, select the file to associate with this program.



- You must click “Save” to retain these changes. The NDF file is now ready to use. If the program was open in LiveView or LadderView before making this change, you must close and re-open for the changes to take place.
- Optional: If you would like to associate this NDF with multiple programs at once, you can select the “Change for all” checkbox to the right. Doing so will associate the NDF file with all the programs identified in the main window table.

### 3 Creating an NDF File

It is beyond the scope of this manual to explain the complete function of Regular Expressions, how they are written, and what they can do. Regular Expressions are well-documented and widely used, so it is up to the user to gain some familiarity with Regular Expressions before proceeding. Two useful resources for getting started are:

YouTube Tutorial: <https://www.youtube.com/watch?v=sa-TUpSx1JA>

Regex Cheat Sheet: <https://www.debuggex.com/cheatsheet/regex/python>

Regex Evaluator: <https://regex101.com/>

#### 3.1 File Format

Any NDF file has a plain text format and can be viewed or edited with a text editor such as Notepad, Notepad++, or Atom (this editor is useful for checking the validity of your Regex Expressions). Each line in the NDF file represents one potential definition and has the following format (with the <> indicators not being present in the file):

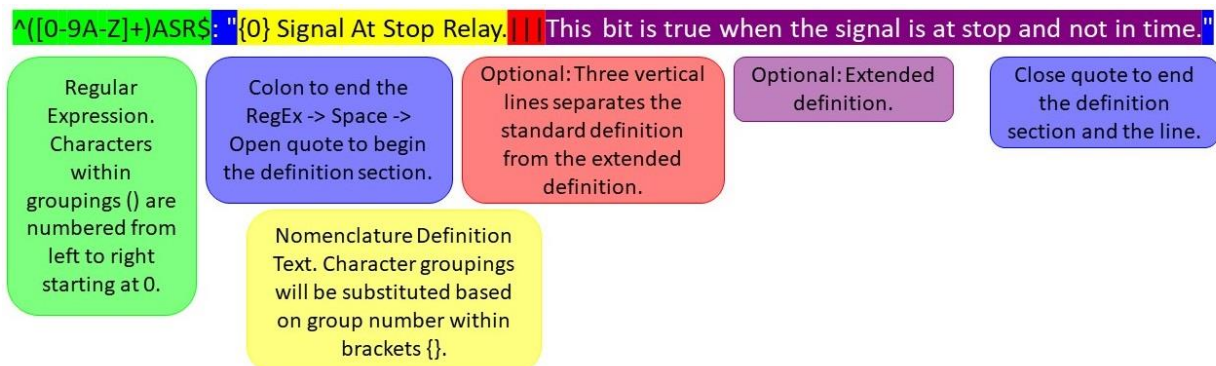
<regular expression>: "Definition with variables based on regular expression group number added between {} symbols | | Additional definition information"

#### 3.2 Parts of the Expression

The following will be used as an example of a line that could be pulled directly from an NDF file for the coming sections.

^([0-9A-Z]+)ASR\$: "{0} Signal At Stop Relay. | | This bit is true when the signal is at stop and not in time. This bit is false otherwise."

##### 3.2.1 Line Format



Following the regular expression is a colon followed by a space. The definition is then encapsulated within a pair of double quotation marks (""). Optionally, this definition can be split in two with the use of three vertical line characters. Text before the vertical line characters will be shown in the "Moused-over Variable Info." box. Text after the vertical line characters will be shown as a tooltip when mousing over the information box in the lower left. The line ends with a regular end line or newline character (enter or return).

### 3.2.2 Regular Expression String

The regular expression above would match with terms such as “2NASR”, “4ASR” and many others.

The example above could be read as follows. The variable name starts with (^) a group of characters (inside the parenthesis). The characters can be any number between 0 and 9 or any capital letter A through Z. There must be at least one but may be more of these characters (+). Following these characters will be the characters ASR in that order, which will be the end of the variable name (\$). Any pattern that matches with this will be assigned the definition text.

### 3.2.3 Definition Text

Any variable name pattern that matches the Regular Expression String will be assigned the definition text. In the example the characters “{0}” would be replaced with the matching characters from the beginning of the Regular Expression string within the parenthesis. The user can insert multiple groupings, which can then be referenced later to be inserted into the definition text. The groupings are numbered starting at 0 going up and are matched from left to right.

The user may wish to add additional information to the definition. Since there is limited space within the “Additional Variable Info” text box the user can add information that will be displayed as a tooltip while mousing over the additional variable information box. To enable this, the user will add three vertical lines (| | |) after the definition text. Fill in the additional information to be displayed and close the line with a close quote.

## 3.3 Multiple Matches

It is possible to match multiple lines in an NDF file with a given variable name. Consider the NDF file consisting of just two lines below:

```
R: "Relay"
```

```
^([0-9A-Z]+)ASR$: "{0} Signal At Stop Relay."
```

In this case, any string that would match the second line, would also match the first since the string is guaranteed to contain an “R”. Signal Scout checks for matching patterns from top to bottom in the NDF file, meaning that as soon as the first line matched, the result would be used, and the second line would never be used. For this reason, the user is encouraged write Regular Expression definitions as tightly as possible, and place more general definitions that can be used for un-caught items toward the end of the file like so:

```
^([0-9A-Z]+)ASR$: "{0} Signal At Stop Relay."
```

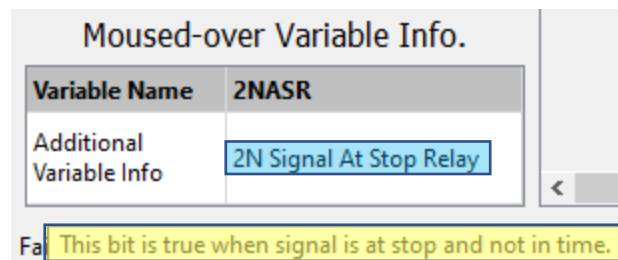
```
R: "Relay"
```

Making the above re-arrangement in the example would allow for ASR bits to be properly identified, and other variables containing “R” to be caught after failing the first test.

## 4 Using Nomenclature Definitions

Once an NDF file is properly created, saved, and associated with a program, it is ready to use in any of the LadderView formats. Mousing over a variable will display additional information about it in the lower left-hand corner of the screen including the nomenclature definition text with values substituted by group as described previously. This information can be used to help train newer signal employees, as well as simplify complex locations and unusual naming schemes for experienced signal professionals.

An example of the function is shown below. The example uses the string below to provide all the information displayed.



`^([0-9A-Z]+)ASR$: "{0} Signal At Stop Relay" || "This bit is true when signal is at stop and not in time."`

## 5 Example of a partial NDF File

```

NCTD - San Diego.ndf
1 ^([0-9A-Z]+)WLR$: "{0} Switch Lock Relay. This is known elsewhere as
  an ASR (At Stop Relay).\"
2 ^([0-9A-Z]+)ASR$: "{0} Signal At Stop Relay.|||This bit is true when
  signal is at stop and not in time.\"
3 ^([0-9A-Z]+)WLRP$: "{0} Switch Lock Repeater.\"
4 ^([A-Z]+)_OK$: "{0} Module Health.\"
5 ^([0-9A-Z]+)AK$: \"Office {0} Track Indication.\"
6 ^([0-9A-Z]+)AKE$: \"LCP {0} Track Indication.\"
7 ^([0-9A-Z]+)C(\d)_IN$: \"{0}-Track Code {1} Received.\"
8 ^([0-9A-Z]+)C(\d)_OUT$: \"{0}-Track Code {1} Transmitted.\"
9 ^([0-9A-Z]+)DGE$: \"{0} Green Signal Output.\"
10 ^([0-9A-Z]+)DGLO$: \"{0} Green Signal Light Out.\"
11 ^([0-9A-Z]+)DKE$: \"LCP {0} Signal Clear Indication.\"
12 ^([0-9A-Z]+)FHGE$: \"{0} Flashing Yellow Signal Output.\"
13 ^([0-9A-Z]+)FRGE$: \"{0} Flashing Red Signal Output.\"
14 ^([0-9A-Z]+)FSR$: \"{0} Approach Traffic Stick.\"
15 ^([0-9A-Z]+)G_PB$: \"LCP {0} Signal Push Button Request.\"
16 ^([0-9A-Z]+)GHS$: \"Office {0} Signal Request.\"
17 ^([0-9A-Z]+)GK$: \"Office {0} Signal Indication.\"
18 ^([0-9A-Z]+)GZ$: \"Non-Vital {0} Signal Request.\"
19 ^([0-9A-Z]+)GZR$: \"Vital {0} Signal Request.\"
20 ^([0-9A-Z]+)HGE$: \"{0} Yellow Signal Output.\"
21 ^([0-9A-Z]+)HGLO$: \"{0} Yellow Signal Light Out.\"
22 ^([0-9A-Z]+)HGR$: \"{0} Route Clear Relay.\"
23 ^([0-9A-Z]+)RED$: \"{0} Red Signal Relay.\"
24 ^([0-9A-Z]+)RGE$: \"{0} Red Signal Output.\"
25 ^([0-9A-Z]+)RGLO$: \"{0} Red Signal Light Out.\"
26 ^([0-9A-Z]+)GLO$: \"{0} Signal Light Out.\"
27 ^([0-9A-Z]+)LO$: \"{0} Signal Light Out.\"
28 ^([0-9A-Z]+)RKE$: \"LCP {0} Signal at Stop Indication.\"
29 ^([0-9A-Z]+)STOP$: \"Non-Vital {0} Signal Stop.\"
30 ^([0-9A-Z]+)TR$: \"{0}-Track Code 1 Received.\"
31 ^(\d)([A-Z])L_TE$: \"Switch Lock Timer Complete Relay for Track {0} to
  the {1}.\"
32 ^(\d)([A-Z])LR$: \"Switch Lock Relay for Track {0} to the {1}.\\nTrue
  sends an unlock to the switch. False will keep it locked.\"
33 ^(\d)([A-Z])LR_EN$: \"Switch Lock Timer Enabled Relay for Track {0} to
  the {1}.\\nIs not used.\"
34 ^(\d)([A-Z])WNP$: \"Switch Normal Input Repeater for Track {0} to the
  {1}.\"
35 ^(\d)LKE$: \"LCP {0} Switch Lock Indication.\"
36 ^(\d)NOR$: \"{0} Normal Switch Overload Relay. Timer Enabled.\"
37 ^(\d)NORTE$: \"{0} Normal Switch Overload Relay. Timer Complete.\"
38 ^(\d)NW_PB$: \"LCP {0} Normal Switch Push Button Request.\"
39 ^(\d)NWCR$: \"{0} Normal Switch Correspondence Relay.\"
40 ^(\d)NWK$: \"Office {0} Normal Switch Indication.\"
41 ^([0-9A-Z]+)NWKE$: \"LCP {0} Normal Switch Indication.\"
42 ^(\d)NWR$: \"{0} Normal Switch Relay (Motor Power).\"
43 ^(\d)NWS$: \"Office {0} Normal Switch Request.\"
44 ^(\d)NWZ$: \"{0} Normal Switch Request (Non-Vital).\"
45 ^(\d)NWZR$: \"{0} Normal Switch Request (Vital).\"
46 ^(\d)ROR$: \"{0} Reverse Switch Overload Relay. Timer Enabled.\"
47 ^(\d)RORTE$: \"{0} Reverse Switch Overload Relay. Timer Complete.\"
48 ^(\d)RW_PB$: \"LCP {0} Reverse Switch Push Button Request.\"
49 ^(\d)RWCR$: \"{0} Reverse Switch Correspondence Relay.\"
50 ^(\d)RWK$: \"Office {0} Reverse Switch Indication.\"
51 ^(\d)RWKE$: \"LCP {0} Reverse Switch Indication.\"

length: 11,590 line Ln: 2 Col: 44 Pos: 139 Windows (CR LF) UTF-8 INS

```