String Object

(STR)

Introduction

In contrast to typical digital and analogue signals in industrial control, in some systems, such as telecommunication and internet applications the control information has to be extracted from a string. For instance it might be required to check if in a dialled phone number the international prefix ‘00’ is provided, or when extracting an XML message the value of an attribute will be evaluated. Also simple ‘understanding’ of an incoming word like ‘start’ might be useful.
The STR object offers all possible flexibility. It can be set up to exactly match the incoming strings, or to extract matching sub strings and provide them to other objects (other STR or a SWIP for converted values) for further evaluation. The functionality is provided by the regular expressions as known in UNIX tools like sed.

Regular Expression

A regular expression (RE) is a notation used for string evaluation. To fulfil various format requirements the RE defines a set of characters as listed in Table 1 below[1].

RE	Meaning	Example
.	Matches one arbitrary character	a.c matches ‘abc’ but not ‘abbc’
^	Matches the beginning of a string	^ab matches ‘abcd’ but not ‘cdab’
$	Matches the end of a string	ab$ matches ‘cdab’ but not ‘abcd’
\n	n=1..9, matches the same string of characters as was matched by a sub expression enclosed between () preceding the \n. n specifies the n-th sub expression	(ab(cd)ef)A\2 matches ‘abcdefAcd’
( )	sub expression	(\d)A(\d) matches 1A2, 0A4 ...
[ ]	Defines a set of characters to be matched	[a-z] matches ‘s’, ‘w’… but not ‘S’, ‘W’…
[^]	Defines all characters except the Characters in the set	[^1-9] matches ‘s’, ‘W’ … but not ‘1’, ‘2’…
(| |)	Matches one of the alternatives	(ab|cd) matches ‘ab’ and ‘cd’
RE+	Matches one or more times the RE	[^1-9]+ matches ‘Good’ but not ‘Obj5’
RE?	Matches one or zero times the RE	abc? matches ‘ab’ and ‘abc’
RE*	Matches zero or more times the RE	ab* matches ‘a’, ‘ab’, ‘abb’ …
RE{n}	Matches exactly n times the RE	ab{2} matches ‘abb’ only
RE{n,}	Matches at least n times the RE	ab{2,} matches ‘abb’, ‘abbb’ but not ‘ab’
RE{n,m}	Matches any number of occurrences between n and m inclusive	ab{1,2} matches ‘ab’ and ‘abb’ only

Table 1: Special characters used with RE

Examples:

- to find leading ‘00’ substring in a phone number following RE could be used:
  ^(0){2}

[1] Note, that there is no standard definition of a regular expression. Thus, you may see descriptions that deviate a little bit from the definition used here.

- to evaluate the numeric value of the attribute ‘type’ of an XML tag following RE could be used: *type=”([0-9]+)”*
- to check if the incoming string is ‘start’, the RE would be *(start)*

Object Description

The STR object is itself a Virtual Finite State machine (VFSM). Its state transition diagram is shown in Figure 1. There are three commands defined:

on – activate the STR object
off – deactivate the STR object
set – reactivate the STR object

After the RTDB is started, the STR VFSM is in state OFF. When receiving command ‘on’ it goes to state INIT. In the INIT state the VFSM watches the string parameter which will deliver the control information to evaluate. When this string parameter changes, the STR VFSM is triggered and analyses the incoming string using the regular expression. Depending on the analysis result the STR VFSM changes then to one of the states: MATCH, NOMATCH or ERROR and stays there until the command ‘set’ is received. When receiving command ‘off’, the STR VFSM goes to state OFF independently of the state in which it was before.

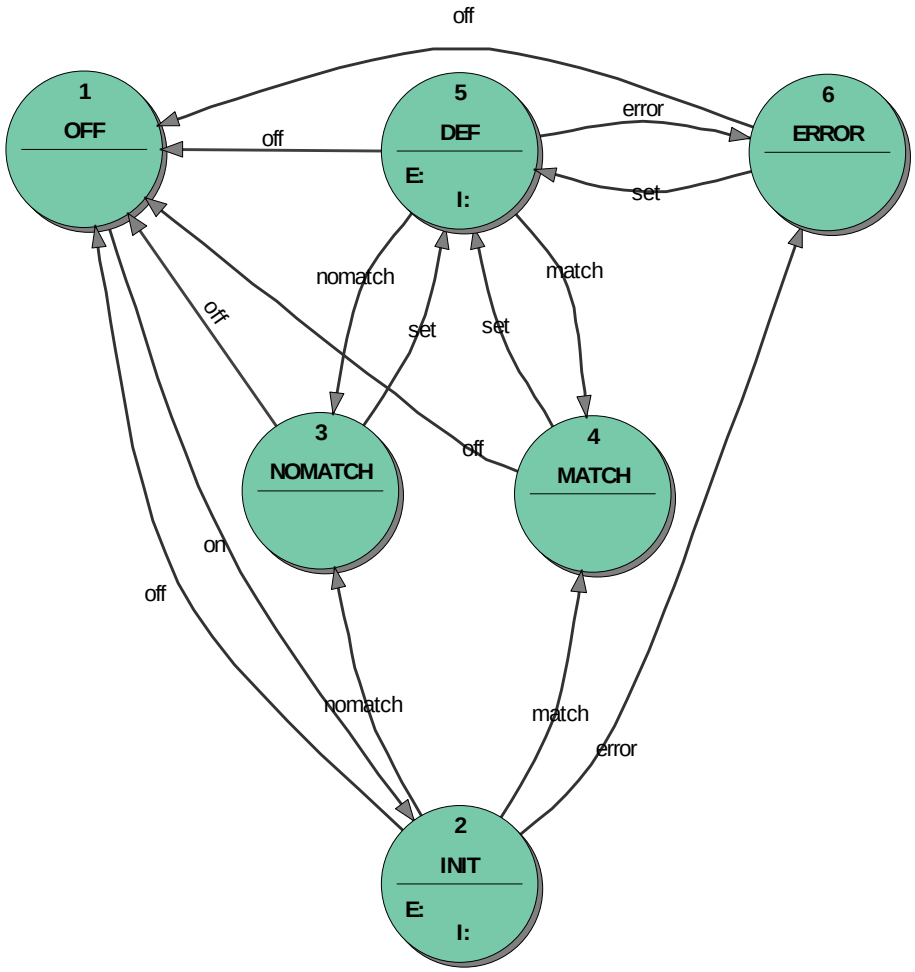

Figure 1. STR Object VFSM

The STR object uses several properties:
Input – Specifies the data object containing the string to be evaluated
Regular expression – the RE can be hard coded or read from an data object
List of substrings – list of data objects which will receive the found substrings for further evaluation. This field can contain data objects of various data types. If a substring will be a number it can be written to a data object with a proper data type and evaluated then using a SWIP object. If a substring will be just a string, it can be evaluated by another STR object after the extraction. The Figure 2 below shows the property window of the STR object:

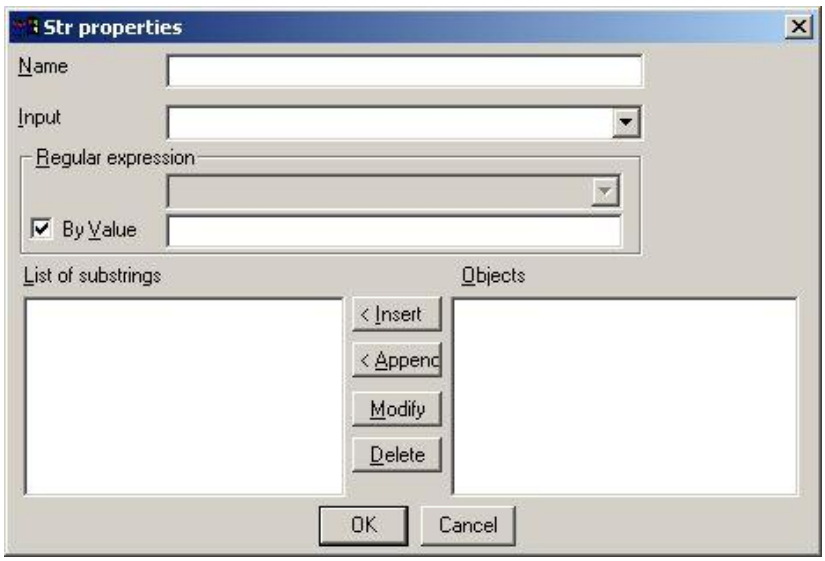

Figure 2. STR Object Properties

Examples

Two examples will be discussed here:

1. Controlling of a VFSM by a string command:
   This example shows how to evaluate incoming strings as pure commands
2. Evaluation of substrings for control purposes
   This example shows how to extract substrings from the received string and convert them to other data types for further evaluation.

Example1: Controlling of a VFSM by a string command

The following state machine is very theoretical and uses mainly two states: IDLE and BUSY. It goes to state BUSY when receiving a string containing the string “start” and returns to state IDLE when receiving a string containing the substring “stop”. It ignores any other incoming strings. Figure 3 shows the VFSM.

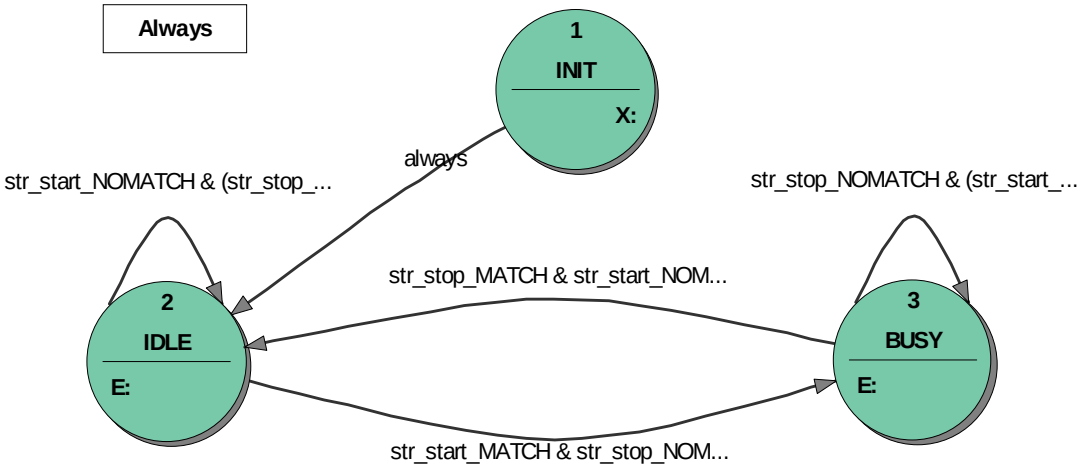

Figure 3. StrCmd VFSM

The used objects are (in the object Id dictionary):

Name	Object Type	Description
MyCmd	CMD-IN	Command for control purposes.
par_command	PAR	Object containing the command string
str_start	STR	Start command string definition. RE=(start)
str_stop	STR	Start command string definition. RE=(stop)

The input names used in the VFSM are:

Name	I/O Object ID	Description
always		Input name always existing
MyCmd_Set	MyCmd	Command to reset the STR objects
str_start_MATCH	str_start	Input name generated when the par_Command object data value is ‘start’
str_start_NOMATCH	str_start	Input name generated when the par_Command object data value is other then ‘start’
str_stop_MATCH	str_stop	Input name generated when the par_Command object data value is ‘stop’
str_stop_NOMATCH	str_stop	Input name generated when the par_Command object data value is other then ‘stop’

The output names used in the VFSM are:

Name	I/O Object ID	Description
1 MyCmd_Clear	MyCmd	Clears MyCmd
2 str_start_On	str_start	Switches the str_start object on
3 str_start_Set	str_start	Sends the ‘Set’ command to the str_start object
4 str_stop_On	str_stop	Switches the str_stop object on
5 str_stop_Set	str_stop	Sends the ‘Set’ command to the str_stop object

To test this VFSM only the par_command has to be changed. With MyCmd_Set the used STR objects can be reset in case it is in the ERROR state due to a wrong regular expression.This example is implemented in project StrCmd.prj.

Example2: Evaluation of substrings for control purposes

The following state machine controls incoming requests to add or remove money to/from an account. It does not allow operations with amounts higher then 20,-EUR
The incoming request is a parameter in the following format:
**amount# - to add an amount
##amount# - to remove an amount
Hence, '**' string means the 'add' operation, '##' string means the 'delete' operation. The amount can be a number between 0 and 20. Single '#' marks the end of the request string.
The state machine as shown in Figure 4 fulfils the entire required functionality, but without real database access where the amounts would be calculated.

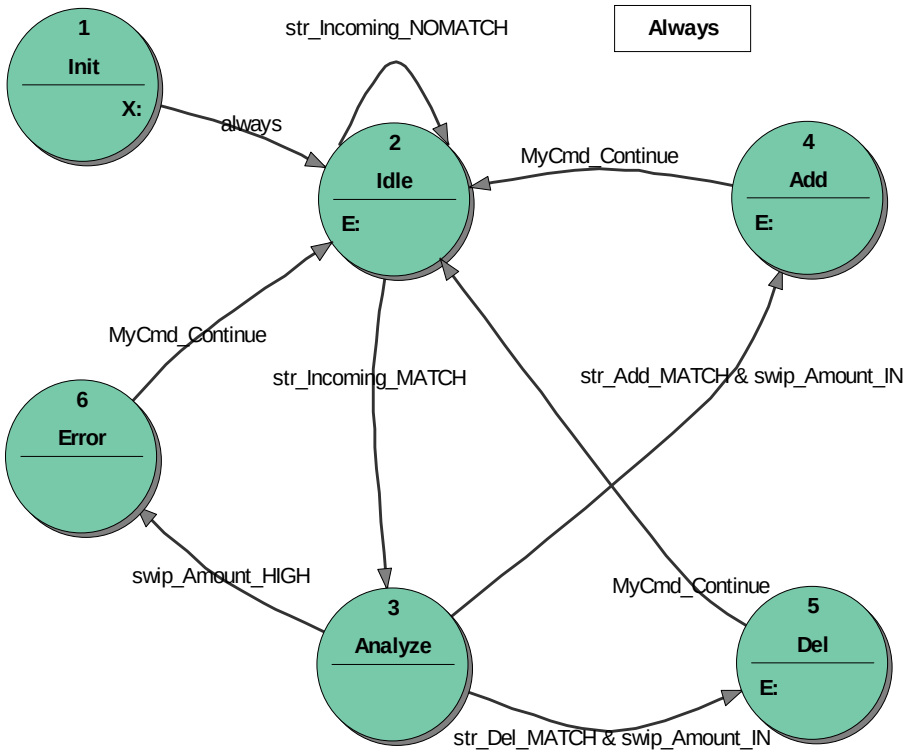

Figure 4: Account VFSM

The used objects are (object ID dictionary):

Name	Object Type	Description
MyCmd	CMD-IN	Command for control purposes; can be used for instance by a master VFSM
swip_Amount	SWIP	Object controlling the par_Amount object; will make sure that par_Amount does not exceed the defined limit
par_Amount	PAR	Object storing the amount as read from the par_Incoming object
par_Incoming	PAR	Object which receives the request from outside
par_Operation	PAR	Object storing the operation as read from the par_Incoming object
str_Add	STR	Add operation string definition. RE=(**)
str_Del	STR	Delete operation string definition. RE=(\#\#)
str_Incoming	STR	Request string definition. RE=(**|\#\#)([0-9]+)\#

The input names used in the VFSM are:

Name	Object ID	Description
always		Input name always existing
MyCmd_Continue	MyCmd	Command to confirm that the VFSM could continue with next request
swip_Amount_HIGH	swip_Amount	Input name generated when the par_Amount object data value exceeds the limit
swip_Amount_IN	swip_Amount	Input name generated when the par_Amount object data value is below the limit
par_Operation_CHANGED	par_Operation	Input name generated when the par_Operation object received new data
str_Add_MATCH	str_Add	Input name generated when the par_Operation object data value is '**'
str_Del_MATCH	str_Del	Input name generated when the par_Amount object data value is '##'
str_Incoming_MATCH	str_Incoming	Input name generated when the par_Incoming object data value matches the regular expression
str_Incoming_NOMATCH	str_Incoming	Input name generated when the par_Incoming object data value does not match the regular expression

The output names used by this VFSM are

Name	I/O Object ID	Description
MyCmd_Clear	MyCmd	Clears MyCmd
swip_Amount_On	swip_Amount	Switches the swip_Amount object on
str_Add_On	str_Add	Switches the str_Add object on
str_Add_Set	str_Add	Sends the 'Set' command to the str_Add object
str_Del_On	str_Del	Switches the str_Del object on
str_Del_Set	str_Del	Sends the 'Set' command to the str_Del object
str_Incoming_On	str_Incoming	Switches the str_Incoming object on
str_Incoming_Set	str_Incoming	Sends the 'Set' command to the str_Incoming object

To test this VFSM only the parameter par_Incoming has to be changed in the monitor. After each request evaluation the command MyCmd_Clear has to be sent to be able to proceed with the next operation. This example is implemented in account.prj.
When you install the StateWORKS Studio you will find the projects for the discusses examples in the folders:
..\Project\Examples-Web\StrCmd and
..\Project\Examples-Web\Account.