

Westpac Technology

Tomorrow's CAN Technology for you Today!

WT ISO 14229 UDS CAN Scripting



User Manual

V1.00

26 June 2020

Diagnostics



Scripting



Quick Start Guide

Preliminary (DRAFT)

This document is a property of Westpac Technology Limited, Westpac Technology Limited reserved the Rights to change the contents of this document without prior notice, for further information, please contact us at www.westpactech.com

Table of Contents

1	Preface	4
2	Introduction	4
3	System Requirement and Installation	4
3.1.	Execute the Scripting File	4
4	Scripting Commands and Formats	4
4.1.	Scripting File Identification	4
4.2.	Comments	4
4.3.	Command String format	5
5	Controls for Dialog Display	5
5.1.	Erase the Dialog Display - Erase	5
5.2.	Show Command String on Dialog Display - Show	5
5.2.1.	Display All Command String type : CommandStringAll or CmdStrAll	5
5.2.2.	Display Transmit Command String type : CommandStringTx or CmdStrTx	6
5.2.3.	Display Receive Command String type : CommandStringRx or CmdStrRx	6
6	Setting for Transmit Identifiers	6
6.1.	Set Transmit Identifier: SetTxID	6
6.1.1.	Alternate	6
6.1.2.	Standard	6
7	Mathematical Data and Logical Operation	7
7.1.	Declaration of an Integer memory Cell – Variable	7
7.2.	Declaration of Constant reference – Constant	7
7.3.	Declaration of ECU Memory Address reference – ECUAddress	7
7.4.	Assigning Data value to the Variable	7
7.5.	Mathematical operators	8
7.6.	Logical Operators	8
7.7.	Conditional Statement- If-Then-Else	8
7.8.	Other Keywords for If-Then-Else	9
7.9.	Specify Data bit length for data usage	9
8	Scripting Operation Flow Controls	10
8.1.	Pause	10
8.2.	Repeat – Loop	10
8.2.1.	Repeat: x	10
8.2.2.	Loop: Condition	10
8.2.3.	Abort Scripting operation	10
8.2.4.	Repeat-Loop Examples	11
9	Transmit Diagnostic Command String: Tx	13
9.1.	Command Field - Tx (TextDescription)	13
9.2.	Transmit Data String Field “CAN Message String”	13
9.3.	Special keywords for “TextDescription”	14
9.3.1.	“DiagnosticMode”	14
9.3.2.	“RequestSeedType”	14
10	Processing Received Diagnostic CAN Message: Rx	15
10.1.	Verify the Reply Message Data : VerifyReplyData	15
10.2.	Report on Verify Reply Data: ReportVerifyReplyData	16
10.3.	Declaring Error handling on reply message data	16
10.3.1.	Negative Response Message: NegativeResponse or NAK	16
10.3.2.	Reply Data String Not Match: ReplyDataNotMatch	17
10.4.	Display the received Diagnostic Data String: Show	18
10.4.1.	N\$ option for Full Description of Value name and Unit	19
10.4.2.	V\$ option for Value only	20
10.4.3.	D\$ Option for the complete received diagnostic Data string	21

11	Data Interpretation: Formula-Value-Bit-Ascii-Hex-BCD	21
11.1.	Formula (F)	21
11.2.	Value (V)	22
11.3.	Value (B)	22
11.4.	Ascii Notation	22
11.5.	Hex or BCD Notation.....	22
12	Save Rx data to Log File.....	23
12.1.	Open/Create a log file	23
12.1.1.	Set Comma as Data Separator	23
12.1.2.	Set Tab Space as Data Separator.....	23
12.2.	Set Data Separator.....	24
12.3.	Print a text line on to the log file	24
12.4.	Output the received diagnostic data to log file	24
13	Appendix A: Multi-Loop Scripting file sample.....	26
13.1.	Original Scripting Text in File	26
13.2.	Output on Dialog Display Panel	27
13.3.	Output of Log File	28
14	Appendix B: Scripting file sample to Log data with Tab Space as Separator	29
14.1.	Original Scripting Text in File	29
14.2.	Output of Log File	30
14.3.	Analysis on 100 Data by Excel Graph.....	30

1 Preface

This document describes the power features of Scripting operation that is available from the license free UDS diagnostic software- “WT ISO 14229 UDS CAN”.

2 Introduction

This Scripting feature is an extension to the UDS diagnostic tool WT ISO 14229 UDS CAN, it provides a commonly used programming features in Scripting text; allowing engineers and technicians a handy tool for ECU testing in a repeated manner, as well as to log the received diagnostic data string in to a file for detail analysis.

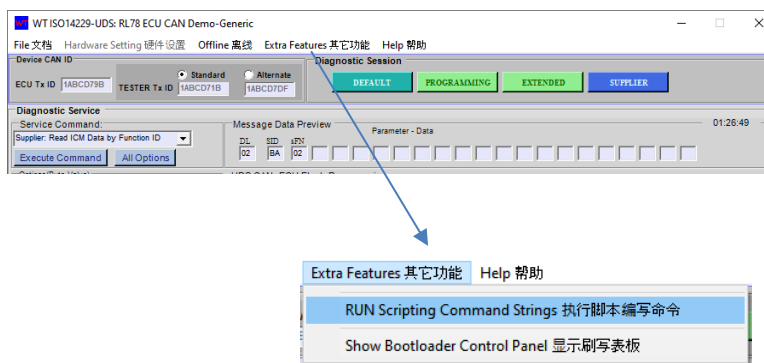
It is designed with the following features to enhanced the engineering performance without spending large amount of money on software tools:

- Scripting commands does not need sophisticated language compiler to operate
- Scripting commands are stored in files and can be used repeatedly
- UDS commands can be repeated in timely manner, so that the ECU performance can be overserved
- ECU data can be stored in a designated file in a timely and orderly manner for detail analysis by Excel worksheets
- The communication with the ECU as Device Under Test will be displayed on Screen Dialog display Panel as well as the log data file, in which they can be copied for Test Report or other documentations.
- On Screen Dialog display for diagnostic communication with ECU can be Copy for documentation.

3 System Requirement and Installation

As mentioned in the Introduction, this Scripting Tool is an extension to “WT ISO 14229 UDS CAN’ diagnostic toll, therefore you need to have “WT ISO 14229 UDS CAN” be installed and a Kvaser CAN interface be connected. For more details please refer to the User Manual of “WT ISO 14229 UDS CAN”.

3.1. Execute the Scripting File



On the tool bar as shown in the above picture, click “Extra Feature” to open the option for “RUN Scripting Command Strings 执行脚本编写命令”, Click this option and follow the instruction to start the Script file operation.

4 Scripting Commands and Formats

4.1. Scripting File Identification

The Scripting file must contain this text string **“WT ISO14229 UDS CAN Tool: Scripting”** as a qualify identifier be recognised as legitimate Scripting command file.

4.2. Comments

Double back slashes will be recognised as comments, that is any text after “//” will be ignored by the Scripting processor.

4.3. Command String format

A standard command string contains a Command and Operand; separated by a “:” as follows:

(Command) : (Operand)

Example 1:

Instruct the Scripting Processor to Pause for 75ms. The Command String will be

Pause : 75

Where “Pause” is a reserved Key word for Pause and 75 = 75 mili-Seconds

Example 2:

To set the Diagnostic session to “Programming, the command string will be:

Tx (DiagnosticMode) : Default

Where

- (a) “Tx” = Scripting command for Transmit the command
- (b) The Text within the two brackets (“ ”) is the text description, but in this case the description “DiagnosticMode” is the reserved key word to set the diagnostic session to “Default”, “Programming”, “Extended”, etc.- See section 9.3.1 for more details.
- (c) “Default” is reserved word for Default Session that is 0x01

So, the result of this command execution will be:

```
Start Diagnostic Session: Default 进入默认诊断会话
Req >> 02,10,01
Ack >> 02 50 01
```

5 Controls for Dialog Display

5.1. Erase the Dialog Display - Erase

Following two commands format are the same to clear the Dialog display:

Format -> **Erase: DialogPanel**
 Or
 Erase: MessagePanel

5.2. Show Command String on Dialog Display - Show

This Command to control the message display On or Off on the Dialog display panel

5.2.1. Display All Command String type : CommandStringAll or CmdStrAll

Format -> **Show: CommandStringAll = Status**, where Status is On or Off

Example A:

Scripts	Show: CommandStringAll = On
Output Dialog	...Activated the Display for all command String 激活显示所有命令字符串

Example B:

Scripts	Show: CmdStrAll = Off
Output Dialog	... Deactivated the Display for all command String 停止显示所有命令字符串

Note: This **OFF** command option will also set the CmdStrTx and CmdStrRx as discussed below sections to OFF

5.2.2. Display Transmit Command String type : CommandStringTx or CmdStrTx

Format -> **Show: CommandStringTx = Status**, where Status is On or Off

Example A:

Scripts	Show: CommandStringTx = On
Output Dialog	...Activated the Display for all Tx command String 激活显示所有发出的命令字符串

Example B:

Scripts	Show: CmdStrTx = Off
Output Dialog	... Deactivated the Display for all Tx command String 停止显示所有发出的命令字符串

5.2.3. Display Receive Command String type : CommandStringRx or CmdStrRx

Format -> **Show: CommandStringRx = Status**, where Status is On or Off

Example A:

Scripts	Show: CommandStringRx = On
Output Dialog	...Activated the Display for all Rx command String 激活显示所有接收的命令字符串

Example B:

Scripts	Show: CmdStrRx = Off
Output Dialog	... Deactivated the Display for all Rx command String 停止显示所有接收的命令字符串

6 Setting for Transmit Identifiers

6.1. Set Transmit Identifier: SetTxID

WT ISO14229 UDS CAN Tool supports two transmit ID ie. Alternate for Broadcast ID and Standard for dedicated ECU .

6.1.1. Alternate

Format -> **SetTxID: Alternate**

Example:

Scripts	SetTxID: Alternate
Output Dialog	*** Alternate (Broadcast) Tx CAN ID selected 选用了功能CAN-ID: 1ABCD7DFh

6.1.2. Standard

Format -> **SetTxID: Standard**

Example:

Scripts	SetTxID: Standard
Output Dialog	*** Standard Tx CAN ID selected 选用了物理CAN-ID: 1ABCD71Bh

7 Mathematical Data and Logical Operation

When the 1st character of the command line is a "#"; it signifies that this command line is for

- Mathematical;
- Logical operation;
- Variable assignment;
- Constant reference assignment;
- ECU Memory Address reference assignment.

This scripting tool provides following memory type references assignment; arithmetic and logical operation:

7.1. Declaration of an Integer memory Cell – Variable

Use the key word "New Data" to assigned a Data variable to 1 of the 16 Memory Cell, the first 2 characters for the variable name must be "v_"; i.e.

Format -> **#: Variable = <v_Var_Name_1>**
 or
 #: Variable = <v_Var_Name_2, <v_Var_Name_3, .. <v_Var_Name_n>
 where n <=16

Example

Scripts	#: Variable = v_Data0 #: Variable = v_Data1, v_Data2, v_Data3
Output Dialog	...Added variable 新增数据变量: New v_Data0 ...Added variable 新增数据变量: New v_Data1, v_Data2, v_Data3

7.2. Declaration of Constant reference – Constant

Use the key word "New Data" to assigned a Data variable to 1 of the 16 Memory Cell, the first character for the variable name must be "c_"; i.e.

Format -> **#: Constant = <c_Const_Name_n>** where 1<= n <=32

Example

Scripts	#: Constant = c_DIDerase equ 0xFF00 #: Constant = c_ProDateYYYYMMDD equ 0x20200101
Output Dialog	...Added constant 新增加常数: New c_DIDerase equ 0xFF00 ...Added constant 新增加常数: New c_ProDateYYYYMMDD equ 0x20200101

7.3. Declaration of ECU Memory Address reference – ECUAddress

Use the key word "New Data" to assigned a Data variable to 1 of the 16 Memory Cell, the first character for the variable name must be "a_"; i.e.

Format -> **#: ECUAddress = <a_ECU_Address_1>** where n <=32

Example

Scripts	#: ECUAddress = a_Speed equ 0x1234FE
Output Dialog	...Added ECU Memory Address 新增加ECU内存地址: New a_Speed equ 0x1234FE

7.4. Assigning Data value to the Variable

Following data notations are supported for value:

- #: v_Data10 = 0x1234 // assigning a hex value 1234 to variable v_Data10
- #: v_Data5 = F199h // Assigning a hex value F199 to variable v_Data5
- #: v_Data4 = 300 // Assigning a decimal value of 300 to variable v_Data4

7.5. Mathematical operators

Following operator supported:

Description	Symbol	
Addition	+	A + B
Subtraction	-	A - B
Multiplication	*	A * B
Division	/	A / B
Bit Exclusive OR	^	A * B

Description	Symbol	
Bit AND	&	A & B
Bit OR		A B
Bit Shift Right	>>	A >> B
Bit Shift Left	<<	A << B
Remainder	%	A % B i.e. 14 % 4 = 2

Following short form also supported

Description	Symbol	Equivalent
A += 3	+=	A = A + 3
A -= 3	+=	A = A - 3

Description	Symbol	Equivalent
A *= 3	+=	A = A * 3
A /= 3	+=	A = A / 3

7.6. Logical Operators

Following logical operator supported:

Description	Symbol	
Equal	=	A = B
Greater Than	>	A > B
Less Than	<	A < B

Description	Symbol	
Greater or Equal Than	>=	A >= B
Less or Equal Than	<=	A <= B
Not Equal	!=	A != B

7.7. Conditional Statement- If-Then-Else

Format -> Following Condition Statement supported:

#: v_Variable1 = If <Condition> Then <Value for True> Else <Value for False>
Or
: v_Variable1 = <Condition>? <Value for True> : <Value for False>

Example A: ***#: v_Data5= if v_Data4=16 then v_Data0***

If v_Data 4 value = 16, assign value v_Data0 to v_Data5; otherwise no change to value in Data5

Scripts	<pre>#: v_Data5=F199h #: v_Data4=16 #: v_Data0=v_Data5*v_Data4 #: v_Data5 = if v_Data4 = 16 then v_Data0</pre>
Output Dialog	<pre>...v_Data5 = 61849 (F199h) ...v_Data4 = 16 (10h) ...v_Data0 = 989584 (F1990h) ...v_Data5 = 989584 (F1990h)</pre>

Example B: ***#: v_Data5 = (v_Data4 != 16)? v_Data0 : v_Data8***

If Data 4 value is NOT equal to 16, assign value Data0 to Data5; otherwise assign Data8 value to Data5

Scripts	<pre>#: Constant = c_DIDerase equ 0xFF00 #: v_Data5 = F199h #: v_Data4 = 0x100 #: v_Data0 = v_Data5 * Data4 #: v_Data5 = (v_Data4 != 0x100)? v_Data0:c_DIDerase</pre>
Output Dialog	<pre>...Added constant 新增加常数: New c_DIDerase equ 0xFF00 ...v_Data5 = 61849 (F199h) ...v_Data4 = 256 (100h) ...v_Data0 = 0 (0h) ...v_Data5 = 65280 (FF00h)</pre>

Example C: Using If_Then_Else statement for conditional execution

Scripts	<pre> Pause: 100#: v_Data0 = 99 #: v_Data4 = 10 #: v_Data5 = 20 Repeat #: (void) = (v_Data0=99)? v_Data4+=5 : v_Data5+=6 //#: (void) = if v_Data0!=100 then v_Data4+=5 else v_Data5+=6 #: v_data0 +=1 Loop: v_data0 <= 100 </pre>
Output Dialog	<pre> ...v_Data5 = 65280 (FF00h) ...v_Data0 = 99 (63h) ...v_Data4 = 10 (Ah) ...v_Data5 = 20 (14h) ...Start of Repeat-Loop 开始重复循环 L1: 1 continue ...v_Data4 = 15 (Fh) ...v_Data0 = 100 (64h) ...Cont. Repeat-Loop 继续 L1: 2 Continue ...v_Data5 = 26 (1Ah) ...v_Data0 = 101 (65h) ...End of Repeat-Loop 结束 L1 </pre>

7.8. Other Keywords for If-Then-Else

Following Keywords are available to enhance the Dialog Display controls, they are

- **NewLine** or **BlankLine**: To display blank line i.e. add a new line on the dialog panel
- **ClearDialog** or **ClearDialogPanel** or **EraseDialog** or **EraseDialogPanel**: To erase/clear the Dialog Display panel

Example:

```
(void) = if v_Data = 10 then ClearDialogPanel else NewLine
```

Since there is no variable to store the result, thus use “(void)” as instruction for no data saving is required, when the v_Data = 10, it clear the Display Panel, otherwise just add a blank line to the Dialog Panel.

7.9. Specify Data bit length for data usage

The value assigned to Variables, Constants and Memory Address are stored in 32 bits, following bit length declarators are available for extractions:

- (u8) : use the lowest 8 bits (1 byte)
- (u16) : Use the lowest 16 bits (2 bytes)
- (u24) : Use the lowest 24 bits (3 bytes)
- (u32): Use all 32 bits (4 bytes)

If bit length is not specified, (u8) will be used as default.

Example A:

```

Script: #: Constant = c_DIDRdPrgDate equ F199h
        #: Constant = c_RdDataBylD equ 22h
        Tx (Read Programming Date) : c_RdDataBylD, (u16)c_DIDRdPrgDate

```

```

Result: Tx (Read Programming Date) : 22,F1,99

```

Example B:

```

Script: #: Constant = c_RdDataBylD equ 0x22
        #: Constant = DIDBlankCheck equ 0xFE00
        #: ECUAddress = a_ProgStart equ 0x2000
        #: Constant = c_ProgSize equ 0xFFFF
        Tx (ECU Blank Check) : c_RdDataBylD, (u16)DIDBlankCheck, (u32) a_ProgStart, (u32)c_ProgSize

```

```

Result: Tx (ECU Blank Check) : 22,FE,00,00,00,20,00,00,0F,FF,FF

```

8 Scripting Operation Flow Controls

8.1. Pause

This command is to suspend (Pause) the execution for a given time; this is practical usage to allow time for the ECU to respond.

Format -> **Pause: <Time in mili-second>**

Example to set the Pause time for 100ms

Scripts	Pause: 100
Output Dialog	...PAUSE: 100ms

8.2. Repeat – Loop

This “Repeat – Loop” instruction set allows a set of commands be executed orderly in sequential order for number of specify loops. Nested loop is permissible, but is limited to 10 level of sub loops.

Format ->

Repeat : x // where x is the number of Repeat for this main loop Level L1, if x is not specified, continue by default

Command_x_1
Command_x_2
Command_x_3

Repeat : y // Where x is the number of Repeat for this sub loop Level L2

Command_y_1
Command_y_2
Command_y_n

Loop: (Condition) // Option to include the condition to repeat; **Loop: Continue** by default

Command_x_4
Command_x_n

Loop: (Condition) // Option to include the condition to repeat; **Loop: Continue** by default

8.2.1. Repeat: x

This **Repeat** instruction sets the start of the command blocks, **x** is to instruct the script to repeat x number of command block. If x is omitted; the default will be **continuing**.

Value x can be a declared variable or constant.


8.2.2. Loop: Condition

This Loop instruction works together with Repeat instruction as discussed above; it serves as the end of the Command Block.

A Boolean **Condition** is included as option to exit the **Repeat-Loop**. If **Condition** is omitted; the default will be **continuing**.

8.2.3. Abort Scripting operation

At start of the running of the scripting file, this button “**ABORT Script Operation**” as shown on the right-hand side picture will be pop up on the screen, click this button to abort the scripting operation as required.



8.2.4. Repeat-Loop Examples

8.2.4.1. Repeat: 2 - for repeat 2 times with unconditional Loop

Scripts	<pre> Repeat: 2 Tx (DiagnosticMode): Default Pause: 100 Tx (DiagnosticMode): Extended Pause:100 Loop </pre>
Output Dialog	<pre> ...Start of Repeat-Loop 开始重复循环 L1: 1 of 2 Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01 P. ...PAUSE: 100ms Start Diagnostic Session: Extended 进入扩展诊断会话 Req >> 02,10,03 Ack >> 02 50 03 P. ...PAUSE: 100ms ...Cont. Repeat-Loop 继续 L1: 2 of 2 Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01 P. ...PAUSE: 100ms Start Diagnostic Session: Extended 进入扩展诊断会话 Req >> 02,10,03 Ack >> 02 50 03 P. ...PAUSE: 100ms ...End of Repeat-Loop 结束 L1 </pre>

8.2.4.2. Repeat- with conditional Loop for termination

Scripts	<pre> #: v_Data10 = 6 // Preset 6 tp v_Data10 Repeat Tx (DiagnosticMode): Default Pause: 100 Tx (DiagnosticMode): Extended Pause:100 #: v_Data10 = v_Data10 + 1 loop: v_Data10 < 8 </pre>
Output Dialog	<pre> ...v_Data10 = 6 (6h) ...Start of Repeat-Loop 开始重复循环 L1: 1 continue Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01 P. ...PAUSE: 100ms Start Diagnostic Session: Extended 进入扩展诊断会话 Req >> 02,10,03 Ack >> 02 50 03 P. ...PAUSE: 100ms ...v_Data10 = 7 (7h) ...Cont. Repeat-Loop 继续 L1: 2 Continue Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01 P. ...PAUSE: 100ms Start Diagnostic Session: Extended 进入扩展诊断会话 Req >> 02,10,03 Ack >> 02 50 03 P. ...PAUSE: 100ms ...v_Data10 = 8 (8h) ...End of Repeat-Loop 结束 L1 </pre>

8.2.4.3. Repeat 10 times, but stop on early with Loop condition

Scripts	<pre> #: v_Data10 = 6 // Preset 6 tp v_Data10 Repeat: 10 Tx (DiagnosticMode): Default Pause: 100 Tx (DiagnosticMode): Extended Pause:100 #: v_Data10 = v_Data10 + 1 loop: v_Data10 < 8 </pre>
Output Dialog	<pre> ...v_Data10 = 6 (6h) ...Start of Repeat-Loop 开始重复循环 L1: 1 of 10 Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01 P. ...PAUSE: 100ms Start Diagnostic Session: Extended 进入扩展诊断会话 Req >> 02,10,03 Ack >> 02 50 03 P. ...PAUSE: 100ms ...v_Data10 = 7 (7h) ...Cont. Repeat-Loop 继续 L1: 2 of 10 Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01 P. ...PAUSE: 100ms Start Diagnostic Session: Extended 进入扩展诊断会话 Req >> 02,10,03 Ack >> 02 50 03 P. ...PAUSE: 100ms ...v_Data10 = 8 (8h) ...End of Repeat-Loop 结束 L1 </pre>

8.2.4.4. Nested Repeat 2 times with unconditional loops

Scripts	<pre> Repeat: 2 Pause: 10 Repeat: 2 Pause: 20 Loop Repeat: 2 Pause: 30 Loop Loop </pre>
Output Dialog	<pre> ...Start of Repeat-Loop 开始重复循环 L1: 1 of 2 ...PAUSE: 10ms ...Start of Repeat-Loop 开始重复循环 L2: 1 of 2 ...PAUSE: 20ms ...Cont. Repeat-Loop 继续 L2: 2 of 2 ...PAUSE: 20ms ...End of Repeat-Loop 结束 L2 and Resume Repeat-Loop 持续 L1 ...Start of Repeat-Loop 开始重复循环 L2: 1 of 2 ...PAUSE: 30ms ...Cont. Repeat-Loop 继续 L2: 2 of 2 ...PAUSE: 30ms ...End of Repeat-Loop 结束 L2 and Resume Repeat-Loop 持续 L1 ...Cont. Repeat-Loop 继续 L1: 2 of 2 ...PAUSE: 10ms ...Start of Repeat-Loop 开始重复循环 L2: 1 of 2 ...PAUSE: 20ms ...Cont. Repeat-Loop 继续 L2: 2 of 2 ...PAUSE: 20ms ...End of Repeat-Loop 结束 L2 and Resume Repeat-Loop 持续 L1 ...Start of Repeat-Loop 开始重复循环 L2: 1 of 2 ...PAUSE: 30ms ...Cont. Repeat-Loop 继续 L2: 2 of 2 ...PAUSE: 30ms ...End of Repeat-Loop 结束 L2 and Resume Repeat-Loop 持续 L1 ...End of Repeat-Loop 结束 L1 </pre>

9.3. Special keywords for “TextDescription”

In the “**TextDescription**” field, it supports the following reserved keywords for common usage:

9.3.1. “DiagnosticMode”

Diagnostic mode supports session for Default, Programming, Extended and Supplier as follows:

Format -> ***Tx (DiagnosticMode) : <Session type>***

(a) Default: ***Tx (DiagnosticMode) : Default*** will send out 02,10,01

Scripts	Tx (DiagnosticMode) : Default
Output Dialog	Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01

(b) Programming: ***Tx (DiagnosticMode) : Programming*** will send out 02,10,02

Scripts	Tx (DiagnosticMode) : Programming
Output Dialog	Start Diagnostic Session: PROGRAMMING 进入编程诊断会话 Req >> 02,10,02 Ack >> 06 50 02 00 19 01 F4

(c) Programming: ***Tx (DiagnosticMode) : Extended*** will send out 02,10,02

Scripts	Tx (DiagnosticMode) : Extended
Output Dialog	Start Diagnostic Session: EXTENDED 进入扩展诊断会话 Req >> 02,10,03 Ack >> 02 50 03

(d) Programming: ***Tx (DiagnosticMode) : SysSupplier*** will send out 02,10,02

Scripts	Tx (DiagnosticMode) : SysSupplier
Output Dialog	Start Diagnostic Session: System Supplier 进入ECU供应商自定义诊断会话 Req >> 02,10,61 Ack >> 02 50 61

9.3.2. “RequestSeedType”

Command to request Security Seed via the “WT ISO14229 UDS CAN” operating method and auto generate the send key – See “WT ISO14229 UDS CAN” User Manual for details.

Format -> ***Tx (RequestSeedType) : <Seed Type>***

Following example of the command for request Seed of type 01 and auto send out Security Key in type 02

Tx (RequestSeedType) : 01

Scripts	Tx (RequestSeedType) : 01
Output Dialog	Request Security Seed 请求种子:类型 01h Req >> 02,27,01 Ack >> 06 67 01 03 1D 03 1D g..... Request Access Key 发送密钥:类型02h Req >> 06,27,02,04,1E,04,1E Ack >> 04 67 02 04 1E g...

10 Processing Received Diagnostic CAN Message: Rx

This Receive CAN Message – Rx command is optional to the Tx command; it is included for verification of the Tx reply message data string against an expected data string or condition; so that a decision to Continue or Stop the Scripting operation.

Following are the available options:

10.1. Verify the Reply Message Data : VerifyReplyData

The conditional setting to Continue or Stop upon unmatching of the comparing data strings is described in section 10.3.2

Format -> **Rx (VerifyReplyData) : <DD, DD, DD, DD, \$\$, D\$,..... DD>**

This is the String to check against the reply message string, “DD” is the 2 digit Hex data, “\$” serves as the don’t care digit

Example A: Stop Scripting operation upon reply data string is not match

Scripts	<pre>#: Constant = c_ProDateYYYYMMDD equ 0x20191205 #: Constant = c_DIDRdPrgDate equ F199h ErrorOnReplyData: ReplyDataNotMatch=Stop Tx (Read Programming Date F199) : 22, (u16)c_DIDRdPrgDate Rx (VerifyReplyData) : 62, (u16)c_DIDRdPrgDate, (u32)c_ProDateYYYYMMDD Rx (VerifyReplyData) : 62, F1, 99, FF, FF, FF, FF Tx (DiagnosticMode) : Default</pre>
Output Dialog	<pre>...Added constant 新增加常数: New c_ProDateYYYYMMDD equ 0x20191205 ...Added constant 新增加常数: New c_DIDRdPrgDate equ F199h ...Scripting operation will stop on Unmatch Message Response 脚本操作将在不匹配消息响应时停止 Tx (Read Programming Date F199) : 22, (u16)c_DIDRdPrgDate Req >> 03,22,F1,99 Ack >> 07 62 F1 99 20 19 12 05 b.. ... Rx (VerifyReplyData) : 07,62,F1,99,20,19,12,05 ...Verification 验证: Reply data string match 回复资料字符串匹配 ! Rx (VerifyReplyData) : 07,62,F1,99,FF,FF,FF,FF ...Note 注意: Reply data string not match 回复资料字符串不匹配 ! ...Stop Scripting Operation! 运作停止!</pre>

Example B: Continue Scripting operation upon reply data string is not match

Scripts	<pre>#: Constant = c_ProDateYYYYMMDD equ 0x20200101 #: Constant = c_DIDRdPrgDate equ F199h ErrorOnReplyData: ReplyDataNotMatch= Cont Tx (Read Programming Date F199) : 22, (u16)c_DIDRdPrgDate Rx (VerifyReplyData) : 62, (u16)c_DIDRdPrgDate, (u32)c_ProDateYYYYMMDD Tx (DiagnosticMode) : Default</pre>
Output Dialog	<pre>...Added constant 新增加常数: New c_ProDateYYYYMMDD equ 0x20200101 ...Added constant 新增加常数: New c_DIDRdPrgDate equ F199h ...Scripting operation will continue on Unmatch Message Response 在不匹配的消息响应中，脚本操作将继续 Tx (Read Programming Date F199) : 22, (u16)c_DIDRdPrgDate Req >> 03,22,F1,99 Ack >> 07 62 F1 99 20 19 12 05 b.. ... Rx (VerifyReplyData) : 07,62,F1,99,20,20,01,01 ...Note 注意: Reply data string not match 回复资料字符串不匹配 ! ...Continue Scripting Operation! 继续执行脚本操作! Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01</pre>

10.2. Report on Verify Reply Data: ReportVerifyReplyData

This option is to report on status count on the “Verify Reply Data” from the beginning of Scripted file execution i.e.

Format -> **Rx (ReportVerifyReplyData) : Count**

Count is a syntax key word must be included

An example of the result display for the “Report Verify Reply Data” of 4 verifications have carried out and were all match

Scripts	Rx (ReportVerifyReplyData):Count
Output Dialog	***Total Data Verify 总数据验证: 4; Match 匹配: 4; Not Match 不匹配: 0

10.3. Declaring Error handling on reply message data

There are two type of declaration are available to stop or continuing the scripting operation, they as follow:

- Negative response; and
- Comparing the reply data string against a specified data string

10.3.1. Negative Response Message: NegativeResponse or NAK

Upon setting a STOP condition for a negative message response, the Scripting Operation will monitor and track the Reply data string to a transmit message, if it encounters a negative (NAK) response, it will Stop/Abort the remaining scripting instructions immediately

Format

ErrorOnReplyData : NegativeResponse = Stop / Continue

Or

ErrorOnReplyData : NAK = Stop / Continue

Note: Short form **Cont** can be used as Continue

Example A: Stop Scripting operation upon received a NAK (0x7F) reply message

Scripts	<pre>ErrorOnReplyData: NAK = Stop Repeat: 2 Tx (Read ICM Data): BA,02 Rx (VerifyReplyData): FA 02 19 12 16 01 01 02 5A 00 00 FF Loop</pre>
Output Dialog	<pre>...Scripting operation will stop on Negative Message Response 脚本操作将在负面消息响应时停止 ...Start of Repeat-Loop 开始重复循环 L1: 1 of 2 Tx (Read ECU Supplier Data) : BA,02 Req >> 02,BA,02 Nak >> 03 7F BA 7F (Service not supported in active session) Negative Response received! Abort/Stop operation ! 收到负面回应, 运作停止 !</pre>

Example B: Continue Scripting operation upon received a NAK (0x7F) reply message

Scripts	<pre>ErrorOnReplyData: NAK = Cont Repeat: 2 Tx (Read ECU Supplier Data): BA,02 Rx (VerifyReplyData): FA 02 19 12 16 01 01 02 5A 00 00 FF Loop</pre>
Output Dialog	<pre>...Scripting operation will continue on Negative Message Response 负面消息响应时将继续执行脚本操作 ...Start of Repeat-Loop 开始重复循环 L1: 1 of 2 Tx (Read ECU Supplier Data) : BA,02 Req >> 02,BA,02 Nak >> 03 7F BA 7F (Service not supported in active session) ...Cont. Repeat-Loop 继续 L1: 2 of 2 Tx (Read ECU Supplier Data) : BA,02 Req >> 02,BA,02 Nak >> 03 7F BA 7F (Service not supported in active session) ...End of Repeat-Loop 结束 L1</pre>

10.3.2. Reply Data String Not Match: ReplyDataNotMatch

This setting sets the condition for comparing the Reply data string against a specified data string; whether the unmatched result should continue or be stopped further scripting command executions.

This section just described the setting for the condition, but the comparison operation is described in section 10.1.

Upon setting a STOP condition for a negative message response, the Scripting Operation will monitor and track the Reply data string to a transmit message, if it encounters a negative (NAK) response, it will Stop/Abort the remaining scripting instructions immediately

Format -> **ErrorOnReplyData : ReplyDataNotMatch = Stop / Continue**

Note: Short form **Cont** can be used as Continue

Example A: Stop Scripting operation upon reply data string is not matched

Scripts	<pre>#: Constant = c_ProDateYYYYMMDD equ 0x20191205 #: Constant = c_DIDRdPrgDate equ F199h ErrorOnReplyData: ReplyDataNotMatch=Stop Tx (Read Programming Date F199) : 22, (u16)c_DIDRdPrgDate Rx (VerifyReplyData) : 62, (u16)c_DIDRdPrgDate, (u32)c_ProDateYYYYMMDD Rx (VerifyReplyData) : 62, (u16)c_DIDRdPrgDate, FF, FF, FF, FF Tx (DiagnosticMode) : Default</pre>
Output Dialog	<pre>...Added constant 新增加常数: New c_ProDateYYYYMMDD equ 0x20191205 ...Added constant 新增加常数: New c_DIDRdPrgDate equ F199h ...Scripting operation will stop on Unmatch Message Response 脚本操作将在不匹配消息响应时停止 Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01 P. Tx (Read Programming Date F199) : 22, (u16)c_DIDRdPrgDate Req >> 03,22,F1,99 Ack >> 07 62 F1 99 20 19 12 05 b.. ... Rx (VerifyReplyData) : 07,62,F1,99,20,19,12,05 ...Verification 验证: Reply data string match 回复资料字符串匹配 ! Rx (VerifyReplyData) : 07,62,F1,99,FF,FF,FF,FF ...Note 注意: Reply data string not match 回复资料字符串不匹配 ! ...Stop Scripting Operation! 运作停止!</pre>

Example B: Continue Scripting operation upon replay data string is not match

Scripts	<pre> #: Constant = c_ProDateYYYYMMDD equ 0x20191205 #: Constant = c_DIDRdPrgDate equ F199h ErrorOnReplyData: ReplyDataNotMatch=Cont //ErrorOnReplyData: ReplyDataNotMatch= Cont //ErrorOnReplyData: NegativeResponse = Stop //ErrorOnReplyData: NegativeResponse = Cont Tx (DiagnosticMode) : Default Tx (Read Programming Date F199) : 22, (u16)c_DIDRdPrgDate Rx (VerifyReplyData) : 62, (u16)c_DIDRdPrgDate, FF, FF, FF, FF Rx (VerifyReplyData) : 62, (u16)c_DIDRdPrgDate, (u32)c_ProDateYYYYMMDD </pre>
Output Dialog	<pre> ...Added constant 新增加常数: New c_ProDateYYYYMMDD equ 0x20191205 ...Added constant 新增加常数: New c_DIDRdPrgDate equ F199h ...Scripting operation will continue on Unmatch Message Response 在不匹配的消息响应中，脚本操作将继续 Start Diagnostic Session: Default 进入默认诊断会话 Req >> 02,10,01 Ack >> 02 50 01 P. Tx (Read Programming Date F199) : 22, (u16)c_DIDRdPrgDate Req >> 03,22,F1,99 Ack >> 07 62 F1 99 20 19 12 05 b.. ... Rx (VerifyReplyData) : 07,62,F1,99,FF,FF,FF,FF ...Note 注意: Reply data string not match 回复资料字符串不匹配 ! ...Continue Scripting Operation! 继续执行脚本操作! Rx (VerifyReplyData) : 07,62,F1,99,20,19,12,05 ...Verification 验证: Reply data string match 回复资料字符串匹配 ! </pre>

10.4. Display the received Diagnostic Data String: Show

Use this keyword “Show: RxData = ” to display the received Diagnostic Data on to the Dialog Display Panel.

Format -> **Show: RxData = <Function Description> = <Data Interpretation>**

Where:

- <Function Description> = The name of this function to be displayed
- <Processed of Receive Data> = The result of the received data in Hex/decimal string or in translated value

Following is the list of options available for **< Data Interpretation >**:

- N\$ for the translated value to be displayed with value name and unit
- V\$ for the translated value to be displayed of value only, no name and unit
- D\$ for displaying the received diagnostic data string in Hexadecimal or Decimal notation

10.4.1. N\$ option for Full Description of Value name and Unit

Format for N\$-> **Show: RxData = <Function Description> = N\$ > <Item_1 name> (Instruction 1) : <Item_2 name> (Instruction 2).... <Item_n name> (Instruction n)**

Where <Instruction> covers the following Data Type Processing (See section 11 For more details)-

- Formula Calculation
- Value description translation
- Bit position interpretation
- ASCII interpretation
- BCD or HEX notation display

Example A: N\$ for Formula calculation

Scripts	<pre>Tx (KL30) : c_RdDataById, (u16)c_DID_RdVbattIEEE Show: RxData = KL30-Full Desc = N\$ > IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)</pre>
Output Dialog	<pre>Tx (KL30) : c_RdDataById, (u16)c_DID_RdVbattIEEE Req >> 03,22,64,03 Ack >> 08 62 64 03 41 51 99 9A 83 bd.AQ... ...KL30 - Full Desc: IEEE Batt.Volt = 13.1000003814697 V, Norm Batt.Volt = 13.1 V</pre>

Example B: N\$ For Value interpretation

Scripts	<pre>Show: RxData = TCU = N\$ > Gear Position(V;1;1;{0=Invalid 1=1st Gear 2=2nd Gear 3=3rd Gear 4=4th Gear 5=5th Gear 6=6th Gear 7=7th Gear 8=8th Gear 9=9th Gear 10=Low 11=Drive 12=Neutral 13=Reverse 14=Park 15=Invalid/Limp}), Word</pre>
Output Dialog	<pre>Tx (TCU) : 22, (u16)a_DID_GearSts Req >> 03,22,62,07 Ack >> 04 62 62 07 06 bb... ...TCU: Gear Position = 6th Gear</pre>

Example C: N\$ For Bit Position interpretation

Scripts	<pre>Show: RxData = ICM = N\$ > Trafficator Status(B;1;1;{0=Trafficator Right 1=Trafficator Left 2=Vehicle Move}), Word</pre>
Output Dialog	<pre>Tx (ICM) : 22,62,08 Req >> 03,22,62,08 Ack >> 04 62 62 08 00 bb... ...ICM: Trafficator Status = bit 0: Trafficator Right = 0; bit 1: Trafficator Left = 0; bit 2: Vehicle Move = 0</pre>

Example D: N\$ for Ascii interpretation

Scripts	<pre>Tx (SW Version): 22,F1,88 Show: RxData = ECU Sw Version = N\$ > Sw Version(Ascii)</pre>
Output Dialog	<pre>Tx (SW Version) : 22,F1,88 Req >> 03,22,F1,88 Ack >> 11 62 F1 88 56 32 30 32 30 2E 30 37 2E 31 38 2E b..V2020.07.18. 30 31 ...ECU Sw Version: Sw Version = V2020.07.18.01 01</pre>

Example E: N\$ for Hex or BCD display

Scripts	<pre>Tx (SW Version): 22,F1,88 Show: RxData = ECU Sw Version = N\$ > Sw Version(BCD)</pre>
Output Dialog	<pre>Tx (SW Version) : 22,F1,88 Req >> 03,22,F1,88 Ack >> 11 62 F1 88 56 32 30 32 30 2E 30 37 2E 31 38 2E b..V2020.07.18. 30 31 ...ECU Sw Version: Sw Version = 56-32-30-32-30-2E-30-37-2E-31-38-2E-30-31</pre>

10.4.2. V\$ option for Value only

Note: V\$ format is almost the same as N\$, but for V\$, the output display will not show the Item Name and Unit

Format for V\$-> **Show: RxData = <Function Description> = V\$ > <Item_1 name> (Instruction 1) : <Item_2 name> (Instruction 2).... <Item_n name> (Instruction n)**

Where <Instruction> covers the following Data Type Processing (See section 11 For more details)-

- Formula Calculation
- Value description translation
- Bit position interpretation
- ASCII interpretation
- BCD or HEX notation display

Example A: V\$ for Formula calculation

Scripts	Tx (KL30) : c_RdDataById, (u16)c_DID_RdVbattIEEE Show: RxData = KL30-Value Only = V\$ > IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
Output Dialog	Tx (KL30) : c_RdDataById, (u16)c_DID_RdVbattIEEE Req >> 03,22,64,03 Ack >> 08 62 64 03 41 51 99 9A 83 bd.AQ... ...KL30 - Value Only: 13.1000003814697, 13.1

Example B: V\$ For Value interpretation

Scripts	Show: RxData = TCU = V\$ > Gear Position(V;1;1;{0=Invalid 1=1st Gear 2=2nd Gear 3=3rd Gear 4=4th Gear 5=5th Gear 6=6th Gear 7=7th Gear 8=8th Gear 9=9th Gear 10=Low 11=Drive 12=Neutral 13=Reverse 14=Park 15=Invalid/Limp}), Word
Output Dialog	Tx (TCU) : 22, (u16)a_DID_GearSts Req >> 03,22,62,07 Ack >> 04 62 62 07 06 bb... ...TCU: 6th Gear

Example C: V\$ For Bit Position interpretation

Scripts	Show: RxData = ICM = V\$ > Trafficator Status(B;1;1;{0=Trafficator Right 1=Trafficator Left 2=Vehicle Move}), Word
Output Dialog	Tx (ICM) : 22,62,08 Req >> 03,22,62,08 Ack >> 04 62 62 08 02 bb... ...Gear: bit 0: Trafficator Right = 0; bit 1: Trafficator Left = 1; bit 2: Vehicle Move = 0

Example D: V\$ for Ascii interpretation

Scripts	Tx (SW Version): 22,F1,88 Show: RxData = ECU Sw Version = V\$ > Sw Version(Ascii)
Output Dialog	Tx (SW Version) : 22,F1,88 Req >> 03,22,F1,88 Ack >> 11 62 F1 88 56 32 30 32 30 2E 30 37 2E 31 38 2E b..V2020.07.18. 30 31 ...ECU Sw Version: V2020.07.18.01 01

Example E: V\$ for Hex or BCD display

Scripts	Tx (SW Version): 22,F1,88 Show: RxData = ECU Sw Version = V\$ > Sw Version(BCD)
Output Dialog	Tx (SW Version) : 22,F1,88 Req >> 03,22,F1,88 Ack >> 11 62 F1 88 56 32 30 32 30 2E 30 37 2E 31 38 2E b..V2020.07.18. 30 31 ...ECU Sw Version: 56-32-30-32-30-2E-30-37-2E-31-38-2E-30-31

10.4.3. D\$ Option for the complete received diagnostic Data string

This \$D option is slightly different to the Hex or BCD option in N\$ and V\$, the entire received diagnostic Data string will be displayed and separated by Comma or Tab space, primary for File output that can be ported to Excel for analysis.

Format (D\$) for Hexadecimal -> **Show: RxData = <Function Description> = D\$ = (Hex)**

Format (D\$) for Decimal -> **Show: RxData = <Function Description> = D\$ = (Dec)**

Example A: for the display the received Data String in Hexadecimal notation

Scripts	Tx (SW Version): 22,F1,88 Show: RxData = ECU Sw Version = D\$ > (Hex)
Output Dialog	Tx (SW Version) : 22,F1,88 Req >> 03,22,F1,88 Ack >> 11 62 F1 88 56 32 30 32 30 2E 30 37 2E 31 38 2E b..V2020.07.18. 30 31 ...ECU Sw Version: (Hex) 11,62,F1,88,56,32,30,32,30,2E,30,37,2E,31,38,2E,30,31

Example B: for the display the received Data string in decimal notation

Scripts	Tx (SW Version): 22,F1,88 Show: RxData = ECU Sw Version = D\$ > (Dec)
Output Dialog	Tx (SW Version) : 22,F1,88 Req >> 03,22,F1,88 Ack >> 11 62 F1 88 56 32 30 32 30 2E 30 37 2E 31 38 2E b..V2020.07.18. 30 31 ...ECU Sw Version: (Dec) 17,98,241,136,86,50,48,50,48,46,48,55,46,49,56,46,48,49

11 Data Interpretation: Formula-Value-Bit-Ascii-Hex-BCD

There are 3 types of translation are available as follows:

11.1. Formula (F)

Format -> **(F,<Start Position>, <Data Length>,<Formula>,<Unit>)**

Where:

- <Start Position> : The position of the actual Received Diagnostic Data bytes, that is after the UDS commands and parameters
- <Data Length>: Number of bytes for the Data
- <Formula> : Mathematical formula or IEEE-754 Floating Point Calculation
- <Unit> : To be Displayed after the result

The received diagnostic data can be translated by the following parameters:

Example A: (F;1;4;IEEE754;V) for Voltage

-> "F" = Formula type, 1st Data Position of 4 byte long with Unit "V"; the calculation will be in IEEE-754 Floating Point

Received Data : 08 62 64 03 **41 51 99 9A 83** Note: The yellow highlighted part is the actual Data

Result -> 0x413E6666 = IEEE754(0x413E6666) = 13.1000003814697 V

Example B: (F;5;1;D*0.1;V) for Voltage

-> "F" = Formula type, 5th Data Position of 1 byte long with Unit "V"; the calculation will be Data x 0.1

Received Data : 08 62 64 03 **41 51 99 9A 83** Note: The yellow highlighted part is the actual Data

Result -> 0x83 = 131 x 0.1 = 13.1 V

11.2. Value (V)

Format -> (V,<Start Position>, <Data Length>,<Value Description>)

Where:

- <Start Position> : The position of the actual Received Diagnostic Data bytes, that is after the UDS commands and parameters
- <Data Length>: Number of bytes for the Data
- <Value Description> : Reference Description of all value combinations

Example: (V;1;1;{0=Invalid|1=1st Gear|2=2nd Gear|3=3rd Gear|4=4th Gear|5=5th Gear| 6=6th Gear| 7=7th Gear| 8=8th Gear| 9=9th Gear|10=Low|11=Drive|12=Neutral|13=Reverse|14=Park| 15=Invalid/Limp})

-> "V" = Value type, 1st Data Position of 1 byte long, Index Reference for Gear Position

Received Data : 05 62 62 07 04 02 Note: The yellow highlighted part is the actual Data

Result -> 0x04 = 4th Gear

11.3. Value (B)

Format -> (B,<Start Position>, <Data Length>,<Value Description>)

Example: (B;2;1;{0=Trafficator Right | 1=Trafficator Left | 2=Vehicle Move})

-> "B" = Bit type, 1st Data Position of 1 byte long, Bit index interpretation

Received Data : 05 62 62 07 04 02 Note: The yellow highlighted part is the actual Data

Result -> 0x02= bit 0: Trafficator Right = 0; bit 1: Trafficator Left = 1; bit 2: Vehicle Move = 0

11.4. Ascii Notation

Format -> (Ascii)

Note: Data position and length is not required as it will interpret the entire data section of the received diagnostic data string

Example:

Received Data : 11 62 F1 88 56 32 30 32 30 2E 30 37 2E 31 38 2E 30 31

Result -> V2020.07.18.01

11.5. Hex or BCD Notation

Format -> (Hex)

Format -> (BCD)

Note:

- Hex and BCD produced the same output
- Data position and length is not required as it will interpret the entire data section of the received diagnostic data string

Example:

Received Data : 11 62 F1 88 56 32 30 32 30 2E 30 37 2E 31 38 2E 30 31

Result -> 56-32-30-32-30-2E-30-37-2E-31-38-2E-30-31

12 Save Rx data to Log File

Following are available commands for File operations:

- File: OpenLogFile = *<Set Data Separator Type >*
- File: DataSeparator = *<Comma/Tab >*
- File: Print = *<Text to be printed to file>*
- File: RxData = *<Instruction>*

12.1. Open/Create a log file

Prior to saving the received diagnostic Data into a file, the file needs to be created.

Use this command **File: OpenLogFile = *<Set Data Separator Type >*** to create a log file and set the data separator for the output data string. A Save File Dialog Window will pop up, follows the on screen instruction to set up the log file. While the ***<Set Data Separator Type >*** will be processed to set the data separator for the output data i.e. Comma or TAB space.

12.1.1. Set Comma as Data Separator

Format -> File: OpenLogFile = Use Comma For Data Separator

Or

Format -> File: OpenLogFile = UseCommaForDataSeparator

Or

Format -> File: OpenLogFile = DataSeparator = Comma

12.1.2. Set Tab Space as Data Separator

Format -> File: OpenLogFile = Use Tab For Data Separator

Or

Format -> File: OpenLogFile = UseTabForDataSeparator

Or

Format -> File: OpenLogFile = DataSeparator = Tab

The advantage of using TAB spacing as the Data separator is the data can be Copied and Paste directly on to the Excel Worksheet's working cells.

Example of using comma as data separation

07,62,F1,00,01,02,03,04

Example of using Tab Space for data separation

07 62 F1 00 01 02 03 04

Tab Space

12.2. Set Data Separator

This option is to allow the output data separator be changed any time during the running of the Scripting commands.

Format -> **File: DataSeparator = <Comma/Tab >**

Example for using Comma as Data separator:

File: DataSeparator = Comma

Example for using Tab Space as Data separator:

File: DataSeparator = Tab

12.3. Print a text line on to the log file

This option is provided for output a text note on to the log file

File: Print = <Text to be printed to file>

Example:

File: Print = Start of Vehicle Testing

Text string "Start of Vehicle Testing" will be output to the log file

12.4. Output the received diagnostic data to log file

Use this keyword "File: RxData =" to display the received Diagnostic Data on to the Log File.

Note: This file command structure is identical to "Show: RxData =" as described in section 10.4, the interpreted data will be output to the Log File, instead of showing it on the Dialog Display.

Format -> **Show: RxData = <Function Description> = <Data Interpretation>**

Where:

- <Function Description> = The name of this function to be displayed
- <Processed of Receive Data> = The result of the received data in Hex/decimal string or in translated value

Following is the list of options available for < Data Interpretation >:

- N\$ for the translated value to be displayed with value name and unit
- V\$ for the translated value to be displayed of value only, no name and unit
- D\$ for displaying the received diagnostic data string in Hexadecimal or Decimal notation

Please also refer to section 10.4 and section 12 for full details discussion on <Data Interpretation> and samples.

Example A: N\$ Log Data for Formula calculation

Scripts	Tx (KL30): c_RdDataById, (u16)c_DID_RdVbattIEEE File: RxData = KL30-Full Desc = N\$ > IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
Output Dialog	Tx (KL30) : c_RdDataById, (u16)c_DID_RdVbattIEEE Req >> 03,22,64,03 Ack >> 08 62 64 03 41 51 99 9A 83 bd.AQ...
Output File	23:07:08.085, KL30 - Full Desc, IEEE Batt.Volt = 13.1000003814697 V, Norm Batt.Volt = 13.1 V

Example B: V\$ Log Data for Formula calculation

Scripts	Tx (KL30): c_RdDataById, (u16)c_DID_RdVbattIEEE File: RxData = KL30=Value Only = V\$ > IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
Output Dialog	Tx (KL30) : c_RdDataById, (u16)c_DID_RdVbattIEEE Req >> 03,22,64,03 Ack >> 08 62 64 03 41 51 99 9A 83 bd.AQ...
Output File	23:07:08.145, KL30=Value Only, 13.1000003814697, 13.1

Example C: N\$ Log Data for Value and Bit Interpretation

Scripts	Tx (TCU): 22,(u16)a_DID_GearSts File: RxData = TCU = N\$ > Gear Position(V;1;1;{0=Invalid 1=1st Gear 2=2nd Gear 3=3rd Gear 4=4th Gear 5=5th Gear 6=6th Gear 7=7th Gear 8=8th Gear 9=9th Gear 10=Low 11=Drive 12=Netural 13=Reverse 14=Park 15=Invalid/Limp}):Trafficator Status(B;2;1;{0=Trafficator Right 1=Trafficator Left 2=Vehicle Move}),Word
Output Dialog	Tx (TCU) : 22, (u16)a_DID_GearSts Req >> 03,22,62,07 Ack >> 05 62 62 07 03 02 bb...
Output File	23:07:08.285, TCU, Gear Position = 3 rd Gear, Trafficator Status = bit 0: Trafficator Right = 0; bit 1: Trafficator Left = 1; bit 2: Vehicle Move = 0

Example D: V\$ Log Data for Value and Bit Interpretation

Scripts	Tx (TCU): 22,(u16)a_DID_GearSts File: RxData = TCU = V\$ > Gear Position(V;1;1;{0=Invalid 1=1st Gear 2=2nd Gear 3=3rd Gear 4=4th Gear 5=5th Gear 6=6th Gear 7=7th Gear 8=8th Gear 9=9th Gear 10=Low 11=Drive 12=Netural 13=Reverse 14=Park 15=Invalid/Limp}):Trafficator Status(B;2;1;{0=Trafficator Right 1=Trafficator Left 2=Vehicle Move}),Word
Output Dialog	Tx (TCU) : 22, (u16)a_DID_GearSts Req >> 03,22,62,07 Ack >> 05 62 62 07 03 02 bb...
Output File	23:07:08.285, 3 rd Gear, bit 0: Trafficator Right = 0; bit 1: Trafficator Left = 1; bit 2: Vehicle Move = 0

Example E: D\$ Log Data for Hexadecimal notation

Scripts	Tx (TCU): 22,(u16)a_DID_GearSts File: RxData = TCU = D\$ > (Hex)
Output Dialog	Tx (TCU) : 22, (u16)a_DID_GearSts Req >> 03,22,62,07 Ack >> 05 62 62 07 03 02 bb...
Output File	01:06:55.734, TCU, (Hex),05,62,62,07,03,02

Example F: D\$ Log Data for Decimal notation

Scripts	Tx (TCU): 22,(u16)a_DID_GearSts File: RxData = TCU = D\$ > (Dec)
Output Dialog	Tx (TCU) : 22, (u16)a_DID_GearSts Req >> 03,22,62,07 Ack >> 05 62 62 07 03 02 bb...
Output File	01:06:55.739, TCU, (Dec),5,98,98,7,3,2

13 Appendix A: Multi-Loop Scripting file sample

This exercise demonstrates the reading of the Battery Voltage (KL30) from ECU via ReadDataByID command, it illustrates the use of Multi-Loop and timing control to retrieve the data.

The demonstration covers:

- Use of Variable and Constant values
- Multi-Loop operational controls
- Output information on to the Panel Display as well as to a file
- Processed data in form of just Hex string, Full Description and Value Only

13.1. Original Scripting Text in File

```
*** WT ISO14229 UDS CAN Tool: Scripting *** // This is the Scripting File Identification

Erase: DialogPanel

//ErrorOnReplyData: ReplyDataNotMatch=Stop
ErrorOnReplyData: ReplyDataNotMatch= Cont
//ErrorOnReplyData: NegativeResponse = Stop
//ErrorOnReplyData: NegativeResponse = Cont

Show: CmdStrAll = On
#: Variable = v_Data0
#: Variable = v_Data1, v_Data2, v_Data3 // Declaring 3 variables
Show: CmdStrAll = Off
#: Variable = v_Data4

#: Variable = v_LoopCounter
#: Constant = c_RdDataByID equ 0x22
#: Constant = c_DID_RdVbattIEEE equ 0x6403
#: Constant = c_50ms equ 50
#: Constant = c_10ms equ 100

Show: CmdStrAll = Off

File: OpenLogFile = Use Comma for Data Separator // or Tab
//File: DataSeparator = Comma // or Tab

File: Print = // Blank Line
File: Print =
*****
File: Print = *** Note: Comma is used for Data separator for this exercise
File: Print =
*****
File: Print = // Blank Line

#: v_LoopCounter = 2

Repeat
  Tx (DiagnosticMode): Default

  File: Print = *** Just the received Diagnostic Data String
  Repeat: 3
    Tx (KL30): c_RdDataByID, (u16)c_DID_RdVbattIEEE
    Show: RxData = KL30 = D$ > (Hex)
    File: RxData = KL30 = D$ > (Hex)
    Pause: c_50ms

  Loop

  File: Print = *** Battery Value only
  Repeat: 3
    Tx (KL30): c_RdDataByID, (u16)c_DID_RdVbattIEEE
    Show: RxData = KL30 = V$> IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
    File: RxData = KL30 = V$> IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
    Pause: c_50ms

  Loop

  File: Print = *** Battery Value in Full Description
  Repeat: 3
    Tx (KL30): c_RdDataByID, (u16)c_DID_RdVbattIEEE
    Show: RxData = KL30 = N$> IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
    File: RxData = KL30 = N$> IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
    Pause: c_50ms

  Loop

  Pause:c_10ms
  #: v_LoopCounter = v_LoopCounter -1

loop: v_LoopCounter != 0
```

13.2. Output on Dialog Display Panel

```

=====
=== Start of Scripting Commands in File 开始执行脚本编写 ===
=====

...Scripting operation will continue on Unmatch Message Response 在不
匹配的消息响应中·脚本操作将继续

...Activated the Display for all command String 激活显示所有命令字符串
Cmd 命令 >> # : Variable = v_Data0
...Added variable 新增数据变量: New v_Data0
Cmd 命令 >> # : Variable = v_Data1, v_Data2, v_Data3
...Added variable 新增数据变量: New v_Data1, v_Data2, v_Data3
Cmd 命令 >> Show : CmdStrAll = Off
...Deactivated the Display for all command String 停止显示所有命令字符串
...Added variable 新增数据变量: New v_Data4
...Added variable 新增数据变量: New v_LoopCounter
...Added constant 新增加常数: New c_RdDataByID equ 0x22
...Added constant 新增加常数: New c_DID_RdVbattIEEE equ 0x6403
...Added constant 新增加常数: New c_50ms equ 50
...Added constant 新增加常数: New c_10ms equ 100
...Deactivated the Display for all command String 停止显示所有命令字符串
...File: E:\VB.NET\WT-UDS CAN 2020\Validation\Scripting Log
Data\Test1.log
...Comma symbol will be used to separate data output 逗号符号将用于分隔
数据输出
...Ready to Save Diagnostic Data 准备保存脚本数据
...v_LoopCounter = 2 (2h)
...Start of Repeat-Loop 开始重复循环 L1: 1 continue
Start Diagnostic Session: Default 进入默认诊断会话
Req >> 02,10,01
Ack >> 02 50 01 P.
...Start of Repeat-Loop 开始重复循环 L2: 1 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 50 00 00 82 bd.AP...
...KL30: (Hex) 08,62,64,03,41,50,00,00,82
...PAUSE: 50ms
...Cont. Repeat-Loop 继续 L2: 2 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 4E 66 66 81 bd.ANff.
...KL30: (Hex) 08,62,64,03,41,4E,66,66,81
...PAUSE: 50ms
...Cont. Repeat-Loop 继续 L2: 3 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 4C CC CD 80 bd.AL...
...KL30: (Hex) 08,62,64,03,41,4C,CC,CD,80
...PAUSE: 50ms
...End of Repeat-Loop 结束 L2 and Resume Repeat-Loop 持续 L1
...Start of Repeat-Loop 开始重复循环 L2: 1 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 4C CC CD 80 bd.AL...
...KL30: 12.8000001907349, 12.8
...PAUSE: 50ms
...Cont. Repeat-Loop 继续 L2: 2 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 4E 66 66 81 bd.ANff.
...KL30: 12.8999996185303, 12.9
...PAUSE: 50ms
...Cont. Repeat-Loop 继续 L2: 3 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 51 99 9A 83 bd.AQ...
...KL30: 13.1000003814697, 13.1
...PAUSE: 50ms
...End of Repeat-Loop 结束 L2 and Resume Repeat-Loop 持续 L1
...Start of Repeat-Loop 开始重复循环 L2: 1 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 50 00 00 82 bd.AP...
...KL30: IEEE Batt.Volt = 13 V, Norm Batt.Volt = 13 V
...PAUSE: 50ms
...Cont. Repeat-Loop 继续 L2: 2 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 51 99 9A 83 bd.AQ...
...KL30: IEEE Batt.Volt = 13.1000003814697 V, Norm Batt.Volt = 13.1
V
...PAUSE: 50ms
...Cont. Repeat-Loop 继续 L2: 2 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 50 00 00 82 bd.AP...
...KL30: IEEE Batt.Volt = 13 V, Norm Batt.Volt = 13 V
...PAUSE: 50ms
...Cont. Repeat-Loop 继续 L2: 3 of 3
Tx ( KL30 ) : c_RdDataByID, (u16)c_DID_RdVbattIEEE
Req >> 03,22,64,03
Ack >> 08 62 64 03 41 4E 66 66 81 bd.ANff.
...KL30: IEEE Batt.Volt = 12.8999996185303 V, Norm Batt.Volt = 12.9
V
...PAUSE: 50ms
...End of Repeat-Loop 结束 L2 and Resume Repeat-Loop 持续 L1
...PAUSE: 100ms
...v_LoopCounter = 0 (0h)
...End of Repeat-Loop 结束 L1
=====
=== End of Scripting Commands in File 结束执行脚本编写 ===
=====

```

13.3. Output of Log File

```
//*****
// WT UDS CAN Scripting - Log File for Diagnostic Receive Data
//=====
// Init File selected: UDS Demo ECU CAN-Generic 29bit ID w_IEEE754 200717.ini
// Scripting File Directory: M:\A3\Diagnostic\M1A Brazil\EEPROM update
// Scripting File name: AppendixA-RdDataByID - Battery Voltage .scp
//
// Tester Transmit ID - Std: 0x1ABCD71B
//                      - Alt: 0x1ABCD7DF
// Tester Receive ID: 0x1ABCD79B
//
// Date: 2020-Jul-29
// Time: 16:04:35.697
//
//*****

*****
*** Note: Comma is used for Data separator for this exercise
*****

*** Just the received Diagnostic Data String
16:04:38.656, KL30, (Hex),08,62,64,03,41,50,00,00,82
16:04:38.845, KL30, (Hex),08,62,64,03,41,4E,66,66,81
16:04:39.045, KL30, (Hex),08,62,64,03,41,4C,CC,CD,80
*** Battery Value only
16:04:39.245, KL30, 12.8000001907349, 12.8
16:04:39.457, KL30, 12.8999996185303, 12.9
16:04:39.687, KL30, 13.1000003814697, 13.1
*** Battery Value in Full Description
16:04:39.905, KL30, IEEE Batt.Volt = 13 V, Norm Batt.Volt = 13 V
16:04:40.097, KL30, IEEE Batt.Volt = 13 V, Norm Batt.Volt = 13 V
16:04:40.305, KL30, IEEE Batt.Volt = 13.1000003814697 V, Norm Batt.Volt = 13.1 V
*** Just the received Diagnostic Data String
16:04:40.648, KL30, (Hex),08,62,64,03,41,51,99,9A,83
16:04:40.851, KL30, (Hex),08,62,64,03,41,51,99,9A,83
16:04:41.069, KL30, (Hex),08,62,64,03,41,51,99,9A,83
*** Battery Value only
16:04:41.278, KL30, 13.1000003814697, 13.1
16:04:41.481, KL30, 13.1000003814697, 13.1
16:04:41.684, KL30, 13.1999998092651, 13.2
*** Battery Value in Full Description
16:04:41.902, KL30, IEEE Batt.Volt = 13.1000003814697 V, Norm Batt.Volt = 13.1 V
16:04:42.105, KL30, IEEE Batt.Volt = 13 V, Norm Batt.Volt = 13 V
16:04:42.298, KL30, IEEE Batt.Volt = 12.8999996185303 V, Norm Batt.Volt = 12.9 V
```

14 Appendix B: Scripting file sample to Log data with Tab Space as Separator

This demonstration illustrates the Data Log of 100 Data into a file, the data is separated by Tab Space; thus, they are easily be copied to an Excel Worksheets, so that Graph can be plotted for visual analysis.

14.1. Original Scripting Text in File

```

*** WT ISO14229 UDS CAN Tool: Scripting *** // This is the Scripting File Identification

Erase: DialogPanel

//ErrorOnReplyData: ReplyDataNotMatch=Stop
ErrorOnReplyData: ReplyDataNotMatch= Cont
//ErrorOnReplyData: NegativeResponse = Stop
//ErrorOnReplyData: NegativeResponse = Cont

Show: CmdStrAll = On
#: Variable = v_Data0
#: Variable = v_Data1, v_Data2, v_Data3 // Declaring 3 variables
Show: CmdStrAll = Off
#: Variable = v_Data4

#: Variable = v_LoopCounter
#: Constant = c_RdDataByID equ 0x22
#: Constant = c_DID_RdVbattIEEE equ 0x6403
#: Constant = c_50ms equ 50
#: Constant = c_10ms equ 10

Show: CmdStrAll = Off

File: OpenLogFile = Use Tab for Data Separator // or Comma
//File: DataSeparator = Comma // or Tab

File: Print = // Blank Line
File: Print =
*****
File: Print = *** Note: Tab Space is used for Data separator for this exercise; Data can directly copy to Excel
File: Print =
*****
File: Print = // Blank Line

#: v_LoopCounter = 100

Tx (DiagnosticMode): Default
File: Print = *** Battery Value only

Repeat

Tx (KL30): c_RdDataByID, (u16)c_DID_RdVbattIEEE
//Show: RxData = KL30 = V$> IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
File: RxData = KL30 = V$> IEEE Batt.Volt(F;1;4;IEEE754;V):Norm Batt.Volt(F;5;1;D*0.1;V)
Pause: c_50ms

#: v_LoopCounter = v_LoopCounter -1

loop: v_LoopCounter != 0

```

14.2. Output of Log File

```
// WT UDS CAN Scripting - Log File for Diagnostic Receive Data
//=====
// Init File selected: UDS Demo ECU CAN-Generic 29bit ID w_IEEE754 200717.ini
// Scripting File Directory: M:\A3\Diagnostic\M1A Brazil\EEPROM update
// Scripting File name: AppendixB-RdDataByID - Battery Voltage then export to Excel .scp
//
// Tester Transmit ID - Std: 0x1ABCD71B
//                      - Alt: 0x1ABCD7DF
// Tester Receive ID: 0x1ABCD79B
//
// Date: 2020-Jul-29
// Time: 16:23:11.650
//
//*****
*****
*** Note: Tab Space is used for Data separator for this exercise; Data can directly copy to Excel
*****

*** Battery Value only
16:23:16.216      KL30      13.1000003814697      13.1      16:23:26.485      KL30      13      13
16:23:16.434      KL30      13.1000003814697      13.1      16:23:26.688      KL30      13      13
16:23:16.637      KL30      13.1000003814697      13.1      16:23:26.891      KL30      13      13
16:23:16.845      KL30      13      13      16:23:27.078      KL30      13      13
16:23:17.063      KL30      13      13      16:23:27.296      KL30      13      13
16:23:17.266      KL30      13      13      16:23:27.515      KL30      12.8999996185303      12.9
16:23:17.484      KL30      13.1000003814697      13.1      16:23:27.718      KL30      13      13
16:23:17.688      KL30      13.1000003814697      13.1      16:23:27.921      KL30      13.1000003814697      13.1
16:23:17.885      KL30      13.1000003814697      13.1      16:23:28.262      KL30      13.1000003814697      13.1
16:23:18.087      KL30      13      13      16:23:28.465      KL30      13.1000003814697      13.1
16:23:18.290      KL30      13.1000003814697      13.1      16:23:28.668      KL30      13.1000003814697      13.1
16:23:18.493      KL30      13      13      16:23:28.887      KL30      13.1000003814697      13.1
16:23:18.712      KL30      13      13      16:23:29.101      KL30      13.1000003814697      13.1
16:23:18.915      KL30      13      13      16:23:29.320      KL30      13      13
16:23:19.131      KL30      13.1000003814697      13.1      16:23:29.539      KL30      13      13
16:23:19.334      KL30      13      13      16:23:29.742      KL30      13.1000003814697      13.1
16:23:19.537      KL30      13      13      16:23:29.945      KL30      13.1000003814697      13.1
16:23:19.740      KL30      13      13      16:23:30.146      KL30      13      13
16:23:19.941      KL30      13.1000003814697      13.1      16:23:30.365      KL30      13      13
16:23:20.144      KL30      13.1000003814697      13.1      16:23:30.568      KL30      13.1000003814697      13.1
16:23:20.362      KL30      13      13      16:23:30.771      KL30      13.1999998092651      13.2
16:23:20.565      KL30      13      13      16:23:30.989      KL30      13.1000003814697      13.1
16:23:20.784      KL30      12.8999996185303      12.9      16:23:31.191      KL30      13.1000003814697      13.1
16:23:20.983      KL30      13      13      16:23:31.394      KL30      13.1000003814697      13.1
16:23:21.186      KL30      12.8000001907349      12.8      16:23:31.598      KL30      13      13
16:23:21.389      KL30      12.8000001907349      12.8      16:23:31.816      KL30      13.1000003814697      13.1
16:23:21.592      KL30      12.6999998092651      12.7      16:23:32.015      KL30      13.1000003814697      13.1
16:23:21.811      KL30      12.8000001907349      12.8      16:23:32.215      KL30      13      13
16:23:22.012      KL30      12.8999996185303      12.9      16:23:32.418      KL30      13      13
16:23:22.215      KL30      12.8000001907349      12.8      16:23:32.621      KL30      13.1000003814697      13.1
16:23:22.418      KL30      12.8999996185303      12.9      16:23:32.840      KL30      13.1000003814697      13.1
16:23:22.637      KL30      12.8000001907349      12.8      16:23:33.045      KL30      13      13
16:23:22.840      KL30      13      13      16:23:33.260      KL30      13.3000001907349      13.3
16:23:23.041      KL30      13      13      16:23:33.463      KL30      12.8999996185303      12.9
16:23:23.244      KL30      12.8000001907349      12.8      16:23:33.666      KL30      13      13
16:23:23.463      KL30      12.8999996185303      12.9      16:23:33.869      KL30      12.8999996185303      12.9
16:23:23.666      KL30      12.8999996185303      12.9      16:23:34.087      KL30      12.8999996185303      12.9
16:23:23.885      KL30      13.1000003814697      13.1      16:23:34.288      KL30      12.8999996185303      12.9
16:23:24.081      KL30      13      13      16:23:34.491      KL30      12.8000001907349      12.8
16:23:24.284      KL30      12.8999996185303      12.9      16:23:34.694      KL30      13      13
16:23:24.487      KL30      13.1000003814697      13.1      16:23:34.897      KL30      13.1000003814697      13.1
16:23:24.690      KL30      13.1000003814697      13.1      16:23:35.118      KL30      12.8999996185303      12.9
16:23:24.893      KL30      13      13      16:23:35.316      KL30      13.1000003814697      13.1
16:23:25.079      KL30      13.1000003814697      13.1      16:23:35.519      KL30      13      13
16:23:25.282      KL30      13      13      16:23:35.722      KL30      13.1000003814697      13.1
16:23:25.485      KL30      12.8999996185303      12.9      16:23:35.941      KL30      13.1000003814697      13.1
16:23:25.688      KL30      13.1000003814697      13.1      16:23:36.143      KL30      13      13
16:23:25.892      KL30      13.1000003814697      13.1      16:23:36.346      KL30      12.8999996185303      12.9
16:23:26.078      KL30      13.1000003814697      13.1      16:23:36.549      KL30      12.8999996185303      12.9
16:23:26.282      KL30      13      13      16:23:36.768      KL30      13      13
```

14.3. Analysis on 100 Data by Excel Graph

