tioScript Editor

Language and Environment Documentation

User's Guide

Literature Number: SBOU070
August 2008

Contents

List of Figures

List of Tables

tioScript Editor

1 Overview

This documentation describes the tio32 script language and its application. This script language is used with the USB DAQ Platform and the USB DIG Platform to directly communicate with the devices. The Script Editor and the runtime environment come with the application:

```
tioScript.exe
```

This software program is a Microsoft® .NET® application; in other words, it is necessary to have .NET 2.0 installed on the PC.

Scripts are stored in comma-separated value (CSV)-text files with a ***.ts0*** file extension.

It is recommended to associate *.ts0* files with the tioScript.exe application so that double-clicking on a script file automatically launches the tioScript.exe program.

2 Working with tioScript.exe

2.1 Installation

There is an installation package available for the script software that contains the following files and directories:

```
29.04.2008 07:47 <DIR> dotnetfx
29.04.2008 08:07 438.272 setup.exe
29.04.2008 08:07 1.487.872 tioScriptSetup.msi
```

Because the script software requires .NET 2.0, the installation also contains an optional .NET installation in the **dotnetfx** directory. This file is called automatically during the install process if it is detected that the PC does not already have .NET installed.

To start the installation, run the *setup.exe* file. It guides you through the installation process as described in Figure 1 through Figure 4.

As shown in Figure 1, the process begins with a Welcome window.

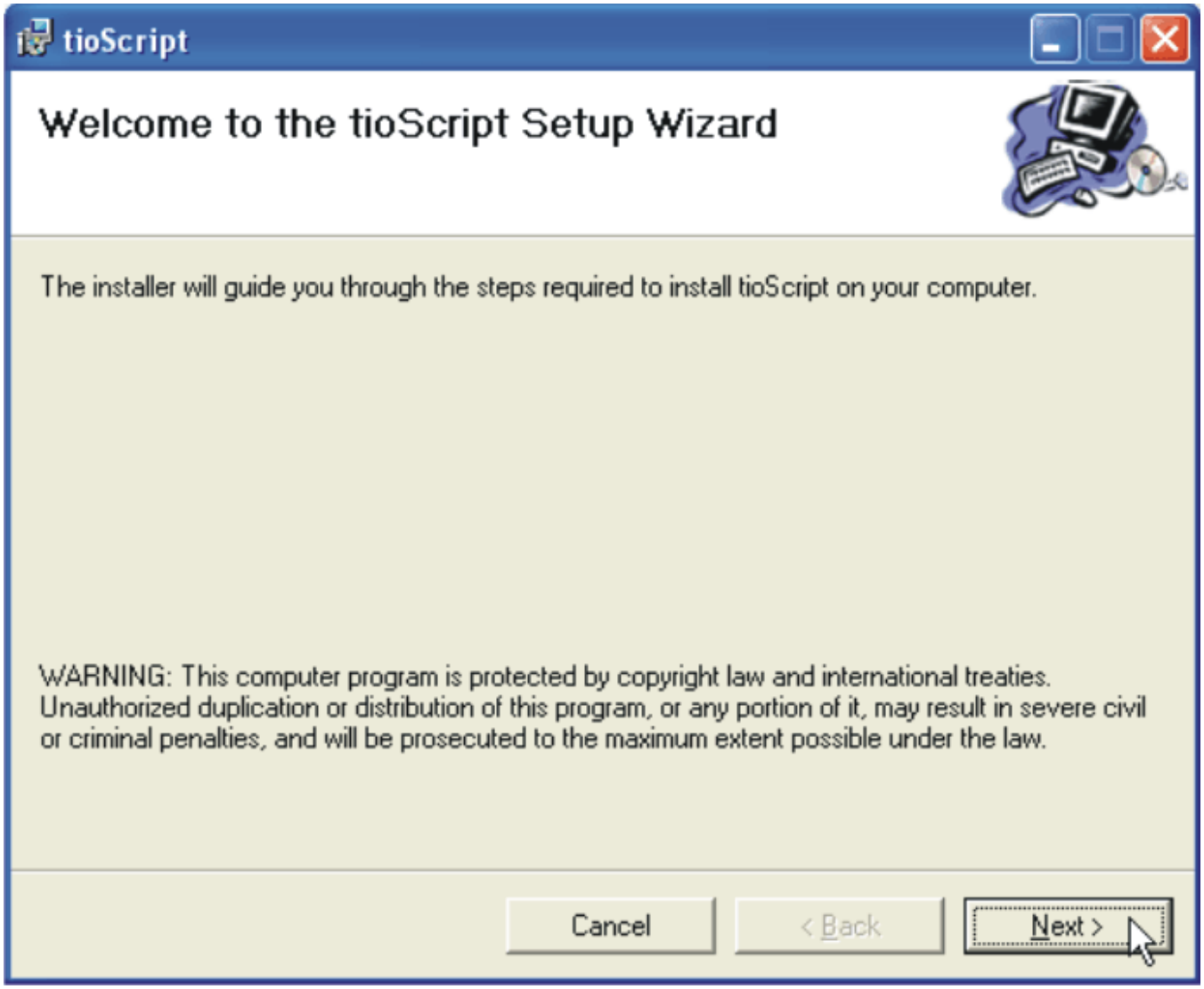


Figure 1. tioScript Installation Welcome Screen

Press the *Next* button; the user is then required to set the application program folder, as shown in Figure 2.

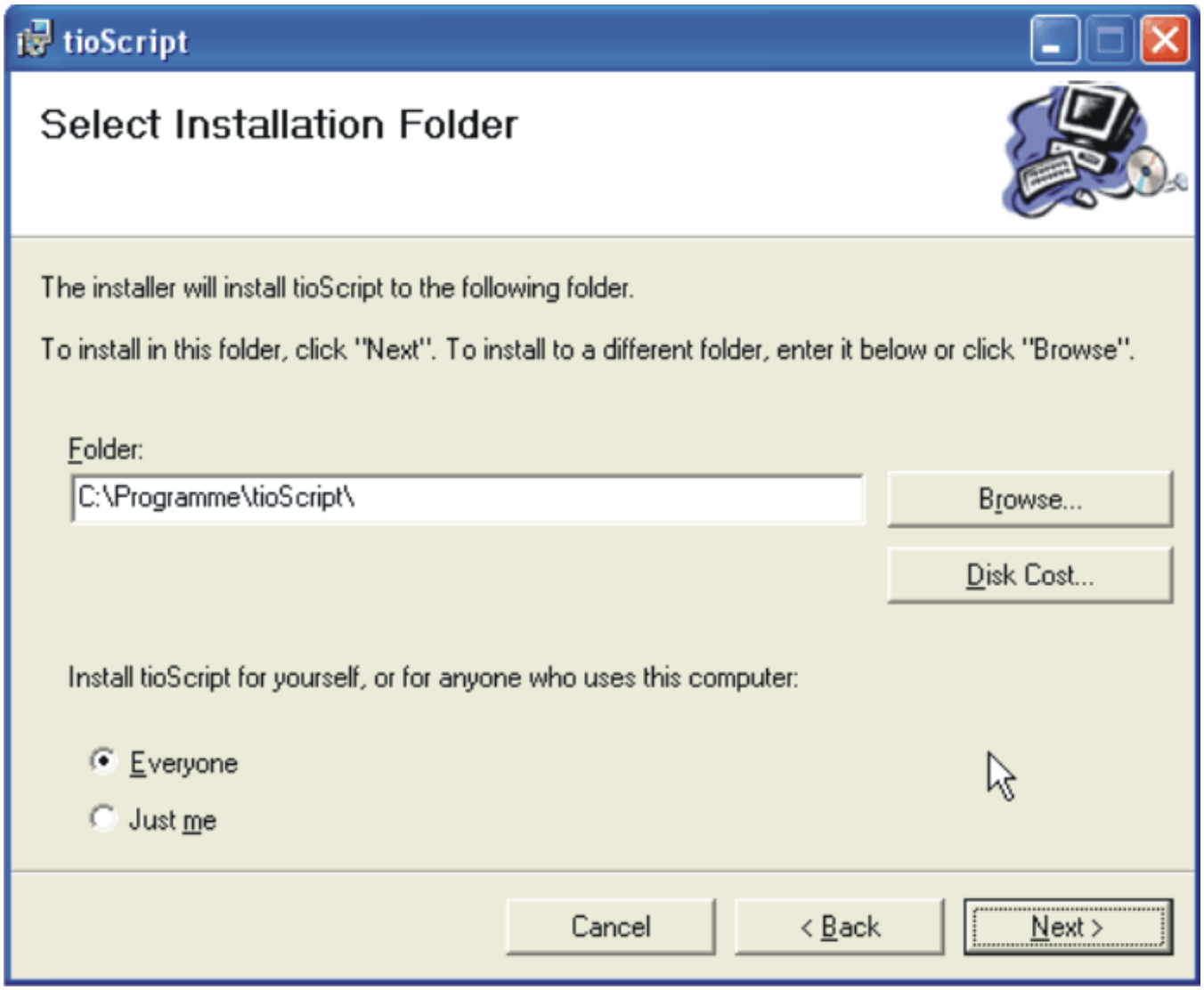


Figure 2. tioScript Installation: Destination Folder

The next screen asks the user to confirm the installation directory and begins the process, as Figure 3 illustrates.

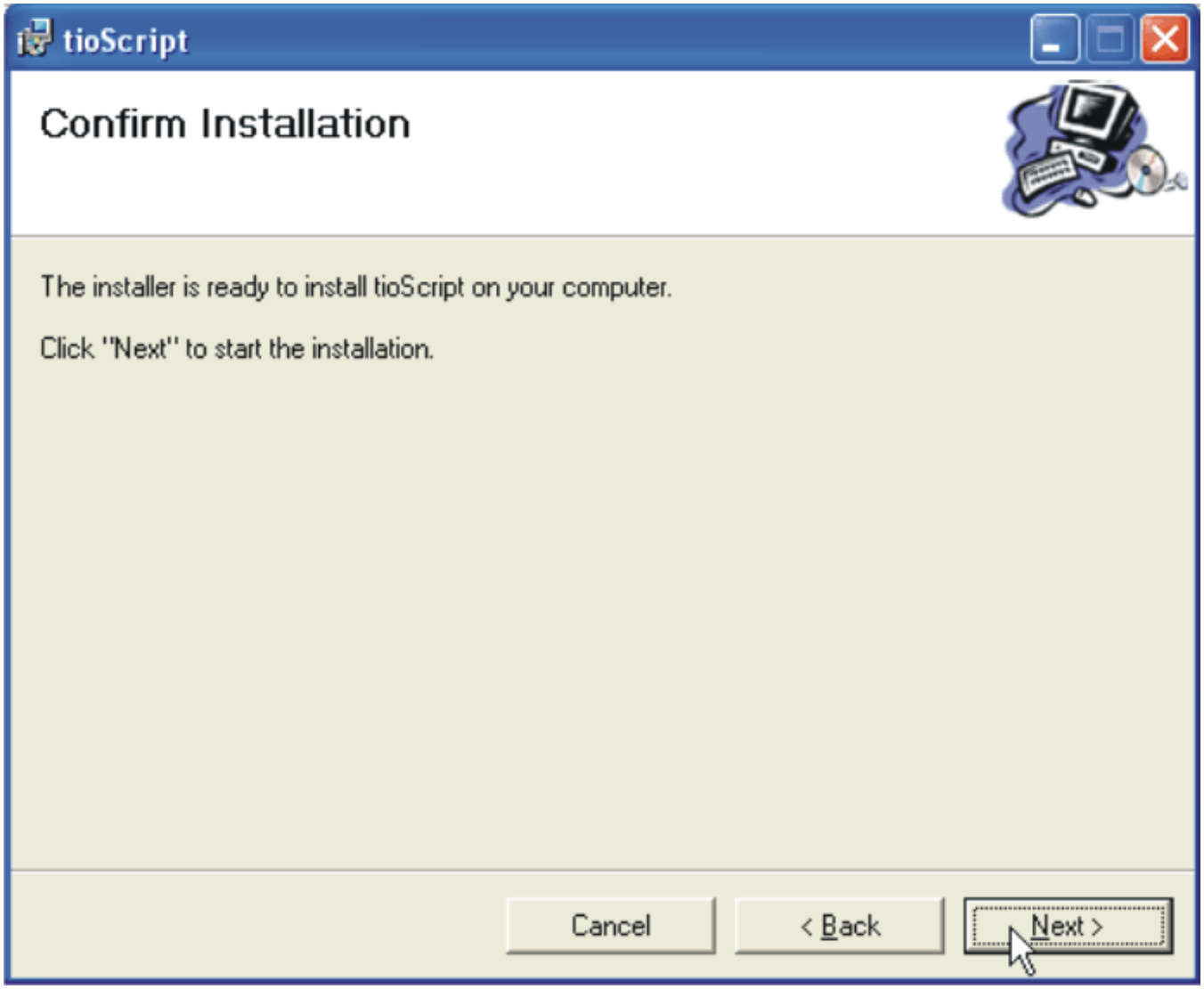


Figure 3. tioScript Installation: Confirmation

Once the installation process completes, the program displays an *Installation Complete* message (shown in Figure 4).

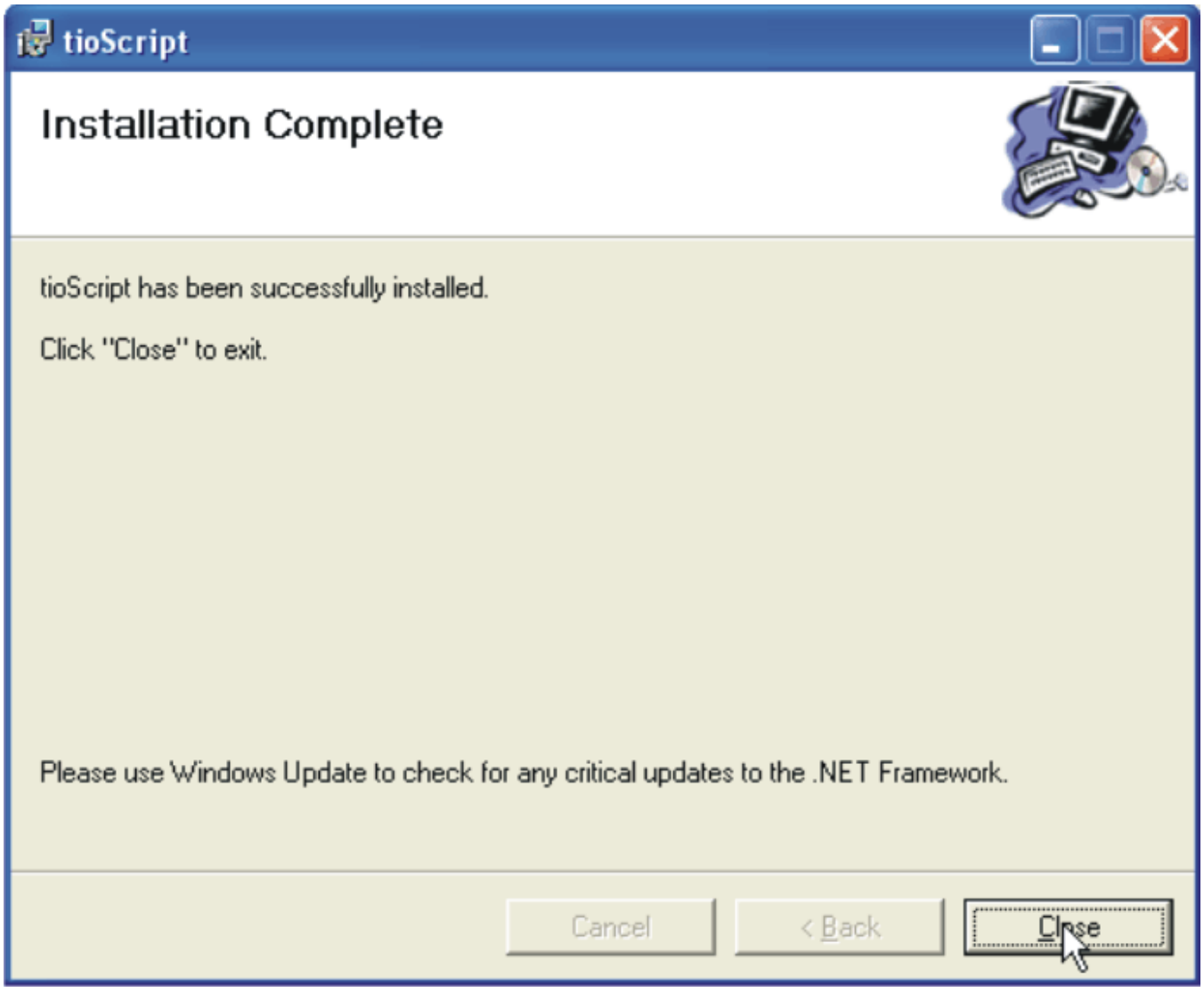


Figure 4. tioScript Installation: Successful Completion

If the user attempts to install a newer version of the Script Editor application on a PC with an existing version of the software installed, the old version must first be removed. Otherwise, the installation process indicates an error, as Figure 5 shows.

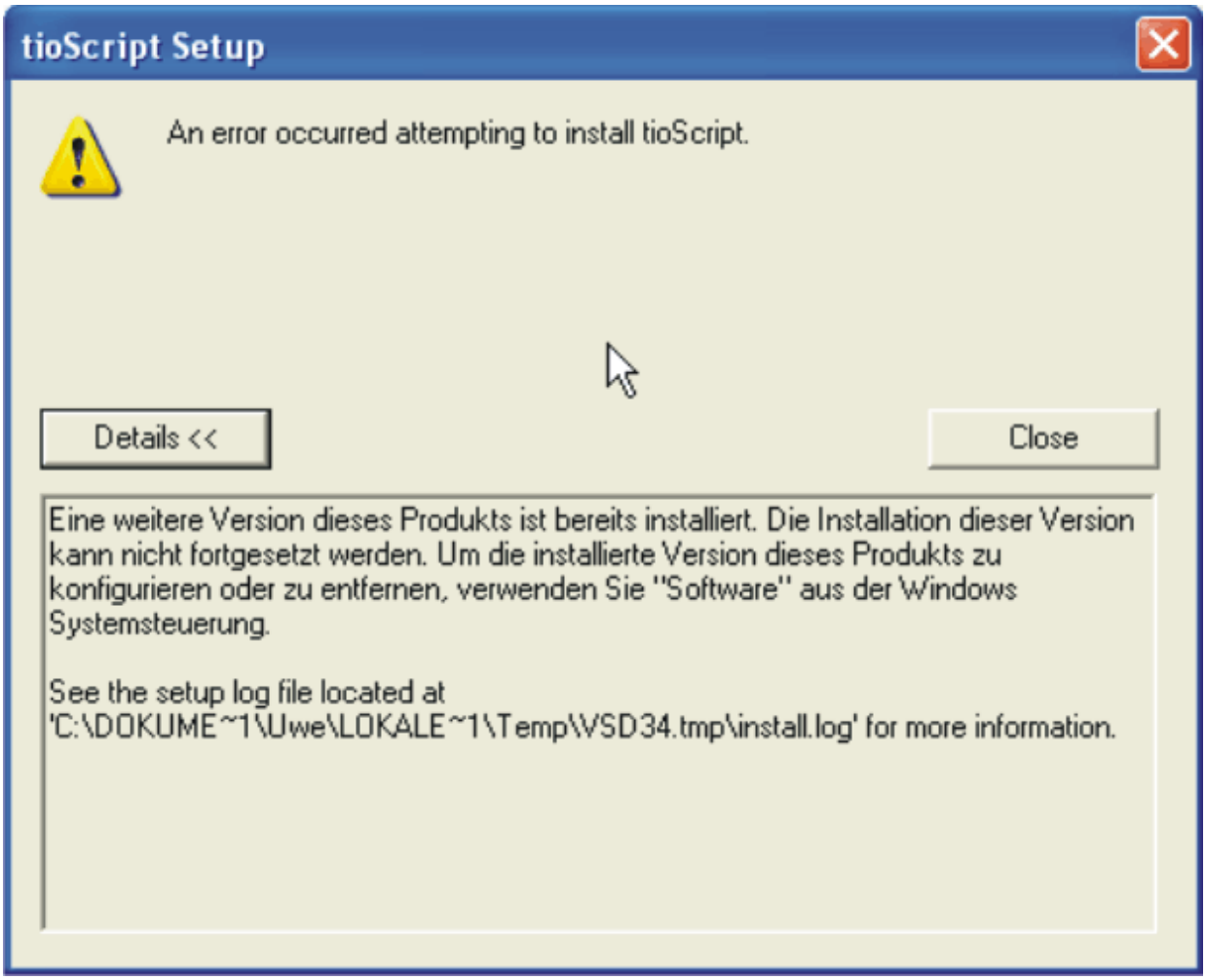


Figure 5. tioScript Installation: Setup Error (Existing Version Must Be Removed)

If this situation occurs, the software can be removed using the system Control Panel (*Start→Control Panel→Add or Remove Programs*).

2.2 The tioScript.exe Window

Double-click the tioScript.exe icon (now located on the PC desktop) to start the tioScript Editor program. The software loads the tio32.dll file to communicate with the USB DAQ or USB DIG board. When the application window is loaded, the settings from the previous session of this program are restored from an .ini file. Figure 6 displays the window.

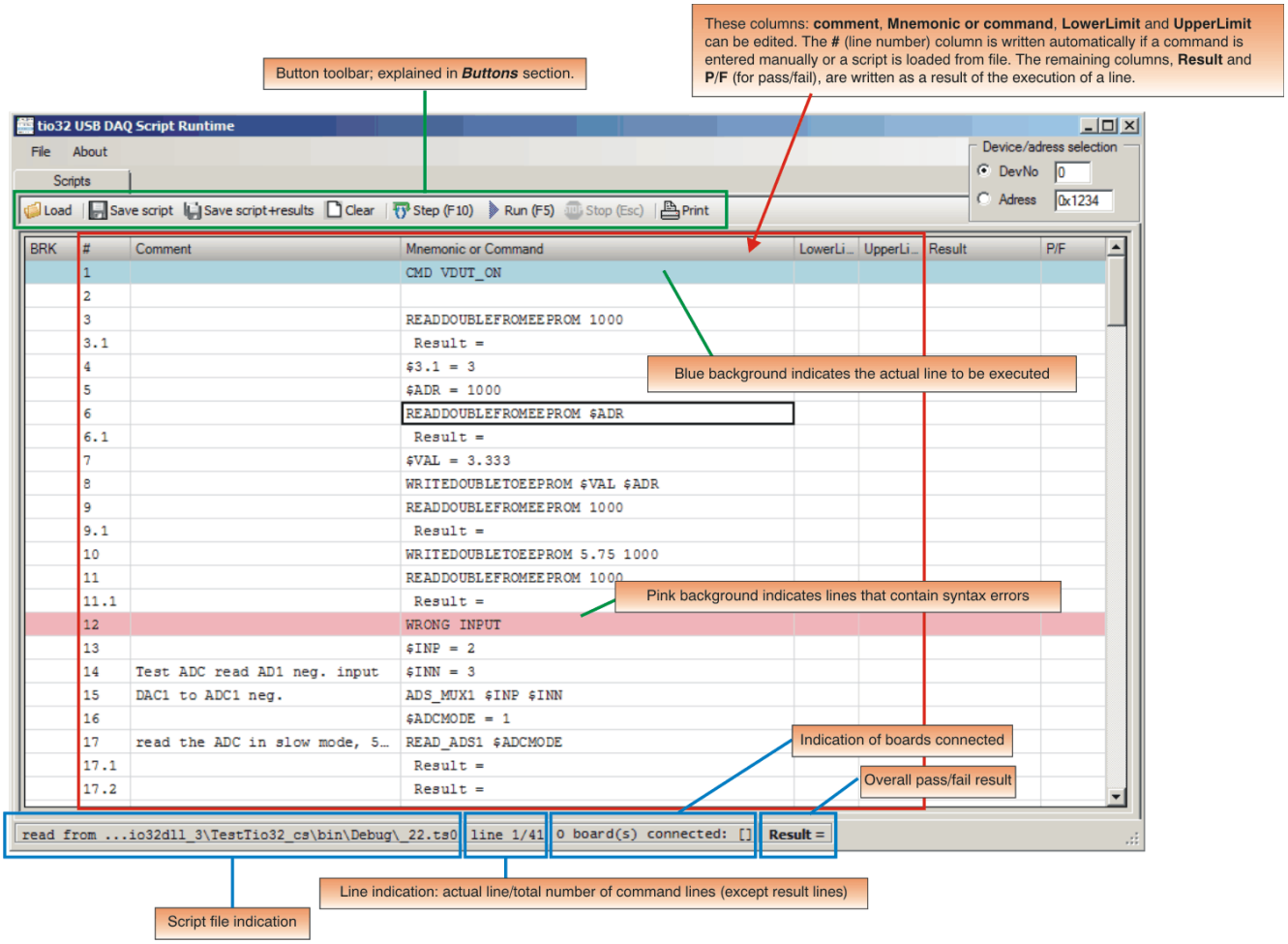


Figure 6. tioScript Editor Window

2.3 Environment Variables

2.3.1 Variable Types and Definitions

The Script Editor allows users to work with different variables. There are two types of variables:

- **user-defined variables:** these variables are declared explicitly by the user. For example: `$value`
- **implicit variables:** these variables hold the result values of any readings; items of this type are generated automatically by the software. For example: `$3.1`

Figure 7 illustrates the environment variables as seen in the Script Editor.

```
11                                  READDOUBLEFROMEEPROM 1000
11.1                                 Result =                                5.7500
12
13                                  $INP = 2
14    Test ADC read AD1 neg. input  $INN = 3
15    DAC1 to ADC1 neg.             ADS_MUX1 $INP $INN
16                                  $ADCMODE = 1
17    read the ADC in slow mode, 5… READ_ADS1 $ADCMODE
17.1                                 Result =                               -0.0119
17.2                                 Result =                               0xFFA9
```

Figure 7. Example Script with Variables

Table 1 defines the variables displayed in Figure 7.

Table 1. Variable Types in Figure 7

In Line Number	Variable Name	Explanation
11.1	$11.1	Implicit variable; generated automatically, after execution holds the double value 5.75 read from EEPROM address 1000
13	$INP	User-generated variable; type becomes INTEGER after execution, the value 2 is assigned to this variable
14	$INN	User-generated variable; type becomes INTEGER after execution, the value 3 is assigned to this variable
16	$ADCMODE	User-generated variable; type becomes INTEGER after execution, the value 1 is assigned to this variable
17.1	$17.1	Implicit variable; generated automatically, after execution holds the double value –0.0119 read from ADC1
17.2	$17.2	Implicit variable; generated automatically, after execution holds the integer value 0xFFA9, the binary code read from ADC1

User-defined variables:

User-defined variables are generated when they are first mentioned as the target of an assignment command, as shown in Example 1. The name of an user-defined variable consists of a leading **$**-sign followed by a string of characters. The first mandatory character of this string must contain a letter or an underscore (_); the remaining optional characters must contain letters, digits, or underscores. All letters are automatically converted to upper-case letters.

Example 1. User-Defined Variables

```
$VAL = 3.0
$VAL16 = $10.1 + 0x100 * $10.2
```

Implicit variables:

Implicit variables are automatically generated. They hold the values of reading results. The name of an implicit variable is also a string, with a leading **$**-sign followed by the contents of the #-column shown in the ScriptEditor window, as shown in Example 2. It is not necessary to assign values to implicit variables; they are assigned values after execution of the line to which they belong.

Example 2. Implicit Variables

```
$17.1
$17.2
```

When lines are deleted or inserted in the active script, implicit variables are re-numbered by the script software. This action also occurs in commands, where only implicit variables are used, as shown in Figure 8.

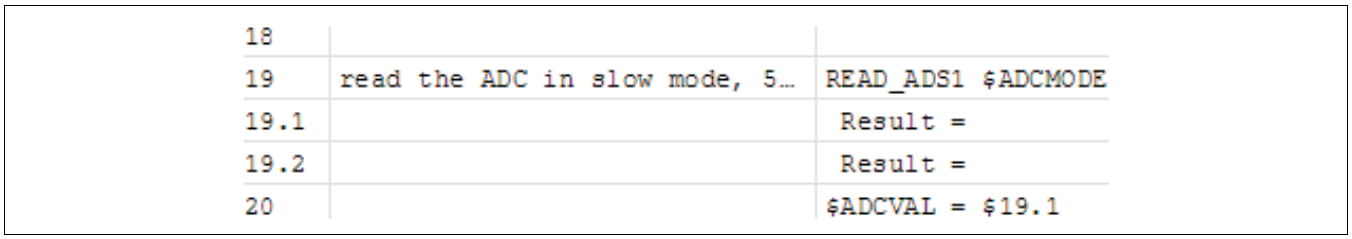


Figure 8. Implicit Variables in Command (Example 1)

After the program deletes line 18, what was formerly line 19 then becomes line 18; the respective results go from *$19.1* and *$19.2* to *$18.1* and *$18.2*. Additionally, in the previous line, *$19.1* becomes *$18.1*, as Figure 9 illustrates.

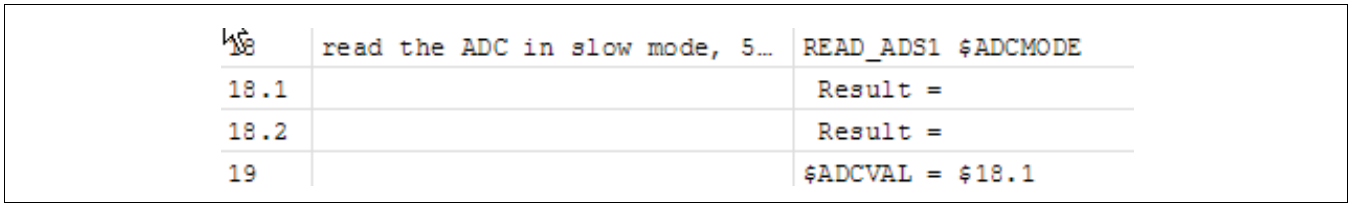


Figure 9. Implicit Variables in Command (Example 2: Deleted Line)

2.3.2 Limit Check of Variable Results

It is possible to check the value of a variable against the limit values presented in the table columns ***LowerLimit or bool*** and ***UpperLimit***. These entries are optional. If there is an entry in either of these columns, it is tested against the defined variable value. Depending on the variable data type, there are two different tests that can be performed.

Boolean variable

The optional entry in *LowerLimit or bool* may be TRUE or FALSE. It is compared to the variable. If it equals the value, the test passes; otherwise, it fails. An entry in *UpperLimit* is ignored in boolean mode. If there is no entry in *LowerLimit or bool*, the test passes.

Arithmetic variable

With arithmetic variables, the following test will be performed.

PASS == *LowerLimit or bool* ≤ (variable value) ≤ *UpperLimit*

Both limit values are optional. If present, they are used in the test. The limit entries may be:

HEX-Values with prefix 0x	For example: 0x1234
decimal values	For example: 1000 ; –300
or float values	For example: 0.0123 ; 3.33E5 ; 2334.567 ; –0.25

The test result affects the P/F column entry of the table. The entry will be PASS or FAIL, depending on the test result.

3 Buttons and Keys

3.1 Load Button

The *Load* icon opens a file load dialog to load a script file. Figure 10 shows the Load button.

Figure 10. Load Button

3.2 Save Script Button

The *Save Script* button, illustrated in Figure 11, opens a file save dialog to save the script into a file. The script is saved without the contents of the Result and P/F columns.

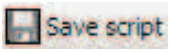


Figure 11. Save Script Button

3.3 Save Script + Results Button

This icon (shown in Figure 12) opens a file save dialog to save the script with all its columns into a file.

Figure 12. Save Script + Results Button

3.4 Clear Button

This button clears the table so that the script is empty. Figure 13 shows the Clear button.

Figure 13. Clear Button

3.5 Step Button

Pressing the Step icon (illustrated in Figure 14) executes one line of the script (that is, a single step of the actual row). Function key F10 performs the same function.

Figure 14. Step Button

3.6 Run Button

Press the Run button, shown in Figure 15, executes commands from actual row until the next BREAK position, or STOP or END command.

Figure 15. Run Button

When running a script, the Stop button is enabled, so that the user has the option to interrupt the execution. Function key F5 performs the same function.

3.7 Stop Button

The Stop icon interrupts the execution. This button (shown in Figure 16) is only enabled if a script is executed. The Esc key performs the same function.

Figure 16. Stop Button

3.8 Print Button

Press the Print button to print all rows and columns of the script, including results and pass/fail information. Figure 17 shows the Print button.

Figure 17. Print Button

4 Context Menu

When the mouse cursor is in the script grid area, one can activate the a context pop-up menu (as shown in Figure 18) by clicking the right mouse button.

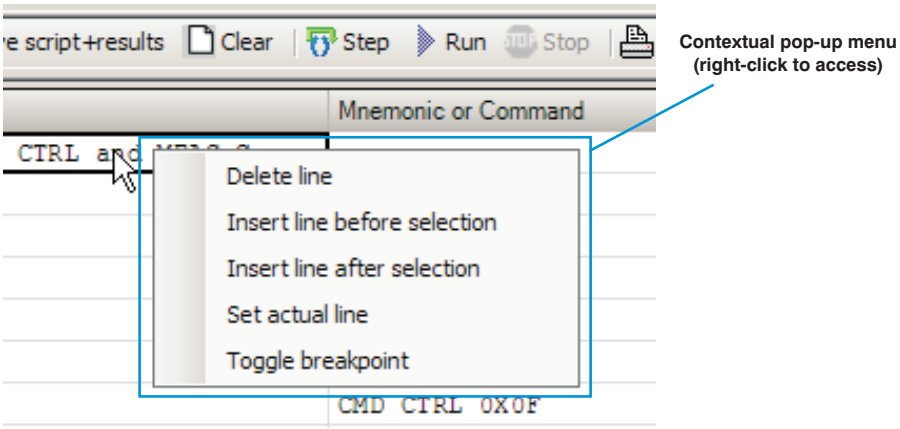


Figure 18. Context Pop-Up Menu

With the left mouse button, a user can select specific table cells for editing or to mark it as the row that is actually referred to. The marked cells are then bordered in black, as Figure 19 shows. Here, row 3 is marked.

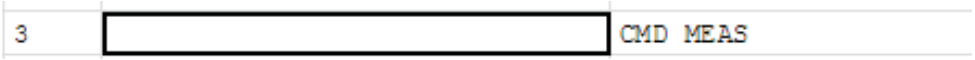


Figure 19. Selected Table Cells

4.1 Delete Line

This command deletes the line that is presently selected. Figure 20 shows the Delete Line command.

Figure 20. Delete Line Command

4.2 Insert Line Before Selection

To insert an empty line before the actual selection, choose this right-click menu option (shown in Figure 21).

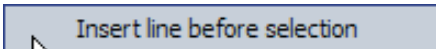


Figure 21. Insert Line Before Selection Command

4.3 Insert Line After Selection

This command inserts an empty line after the actual selection, as Figure 22 shows.

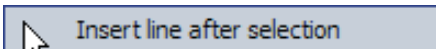


Figure 22. Insert Line After Selection Command

4.4 Set Actual Line

This command (illustrated in Figure 23) sets the line that is selected as the next line to be executed. The *actual* line means that this line is to be executed when either the Step or Run button is pressed.

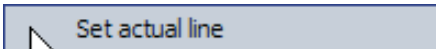


Figure 23. Set Actual Line

The actual line becomes highlighted with a light blue background, as Figure 24 shows; in this case, line 11 with the PAUSE command is the actual line to be executed.

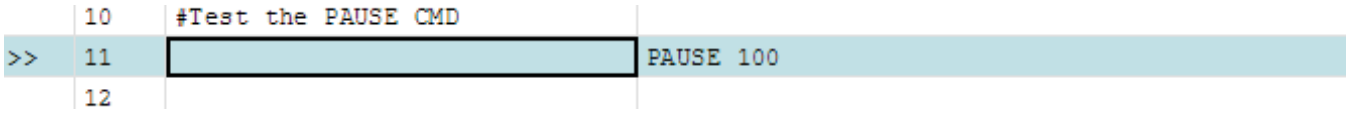


Figure 24. Selected Actual Line

4.5 Toggle Breakpoint

The Toggle Breakpoint command, as shown in Figure 25, toggles the breakpoint in the line that is actually selected. A breakpoint is indicated with *BRK* in the first left column.

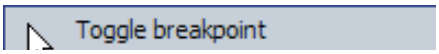


Figure 25. Toggle Breakpoint Command

In the example shown in Figure 26, a breakpoint is set at line 11 with the PAUSE command.

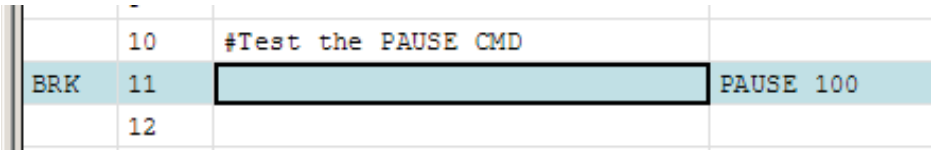


Figure 26. Toggle Breakpoint Example

It is possible to set as many breakpoints as the script has command lines. Breakpoints cannot be set in result lines.

4.6 Debug Window

Pressing the F11 function key allows users to print a list of all known variables with the respective actual values into the debug window and then display that window. Figure 27 illustrates the Debug window.

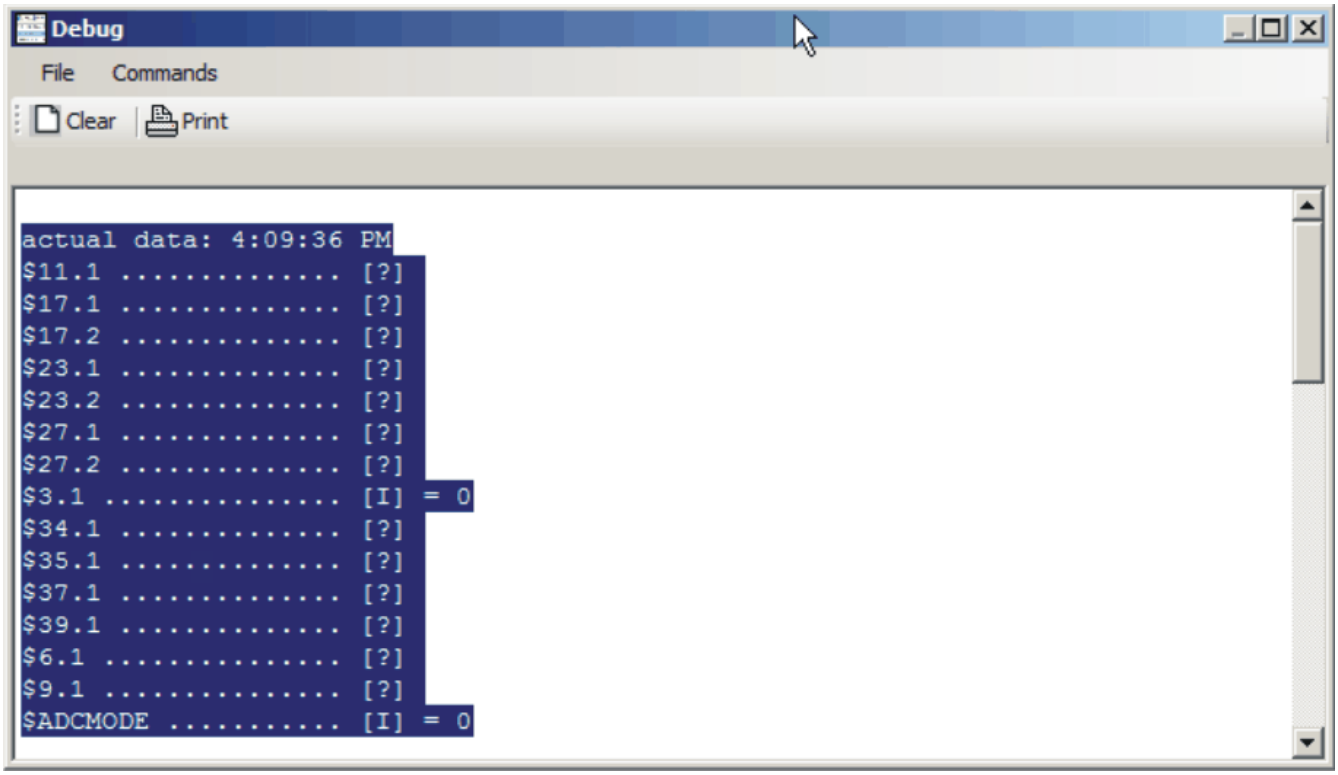


Figure 27. Debug Window

The contents of the debug window are deleted by pressing the *Clear* button (see Section 3.4). The Print button (explained in Section 3.8) prints a list of all known variables with the respective actual values.

5 Mnemonic Commands

The Script Editor uses three categories of commands:

- Mnemonic Commands
- High-Level Commands
- Script Editor Commands

A *mnemonic command* is a series of test strings that can be used to generate digital signals with either the USB DAQ Platform or the USB DIG Platform. For example, the mnemonic command `I2C CH1 S 80 ACKS 16 ACKS P` generates an I²C™ signal. Mnemonic commands can also be used to configure the USB DAQ and USB DIG Platforms. For example, `CMD VDUT_ON` turns on the V_{DUT} power supply.

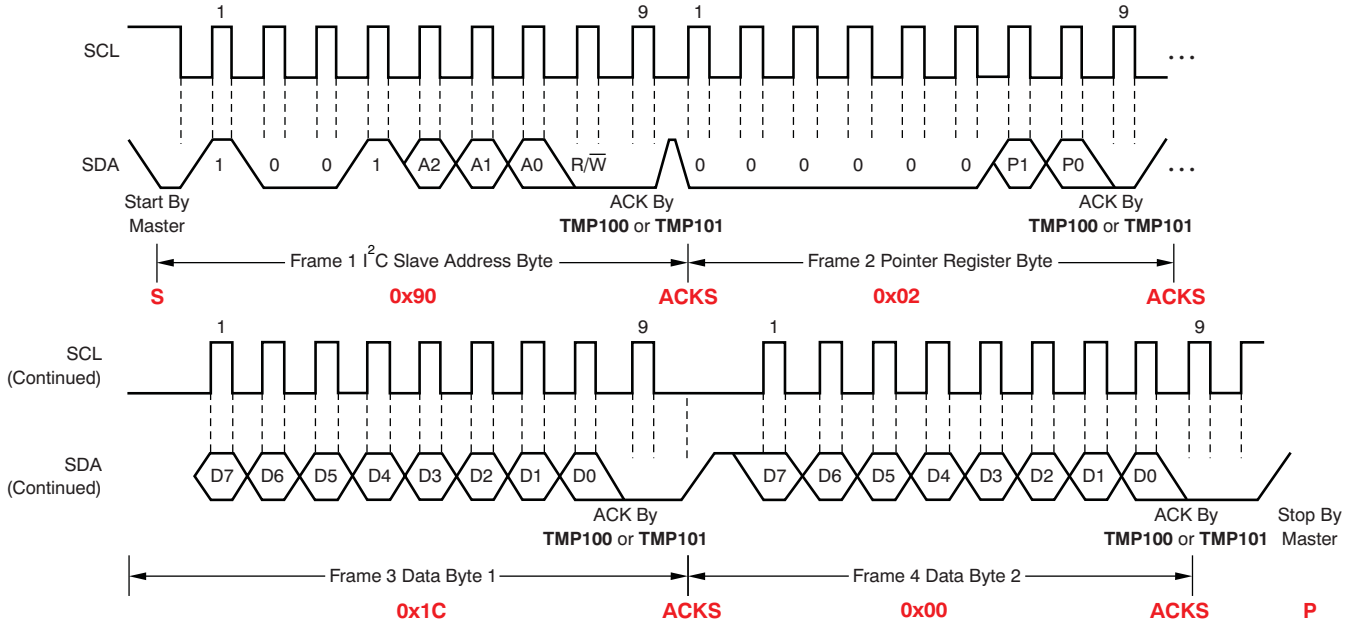
Mnemonic commands are different from high-level commands in that they are a single transaction with the USB DAQ Platform. High-level commands are composed of multiple mnemonic commands. An example of a mnemonic command is a single I²C write. An example of a high-level command is a DAC write. The DAC write consists of multiple I²C writes.

5.1 I²C Mnemonic Commands

Table 2 summarizes the I²C mnemonic commands. Figure 28 and Figure 30 illustrate how an I²C transaction can be translated into a mnemonic command using the strings from Table 2.

Table 2. List of I²C Mnemonic Commands

Command	Explanation
I2C	This sequence is always the first mnemonic in an I²C command.
CH1 CH2	This sequence is always the second mnemonic in an I²C command. This mnemonic identifies which I²C channel is used. The EVM contains two I²C channels to allow for the connection of two different I²C devices.
S	Start condition. This command means: ask all I²C devices on bus to listen.
P	Stop condition. Thsis command means: let the I²C bus know the transaction is completed.
R	Read an 8-bit word
ACKS	Checks to see if the slave acknowledged (that is, SDA is read during the acknowledge pulse to see if the slave acknowledged).
ACLM	Drives the acknowledge bit low (that is, SDA is driven low by the EVM).
Example: **0x3A 0x21 0xBA**	Write an 8-bit word (0x00 to 0xFF for hex, and 00 to 255 for decimal)



I2C CH1 S 0x90 ACKS 0x02 ACKS 0x1C ACKS 0x00 ACKS P

Figure 28. Generation of an I²C Mnemonic Command with an I²C Digital Pattern

Figure 28 shows how the I²C mnemonic command is generated using an I²C digital pattern. Note that the mnemonic *I²C* is always first in I²C commands. The mnemonic *CH1* indicates which channel is used. The USB DAQ Platform has two I²C channels available. The remainder of the command (that is, *S 0x90 ACKS 0x02 ACKS 0x1C ACKS 0x00 ACKS P*) depends on the specific I²C signal the user wants to generate.

Figure 29 shows how an I²C command appears when entered into the Script Editor. Note that lines 4.1 to 4.3 are automatically generated. These lines display the results. They can also be used as implicit variables *$4.1*, *$4.2*, and *$4.3*. Limits can be used to test the results. A '0' for the Acknowledge result indicates that the device acknowledges. The limits are both set to 0 so that an the test passes when the device acknowledges.

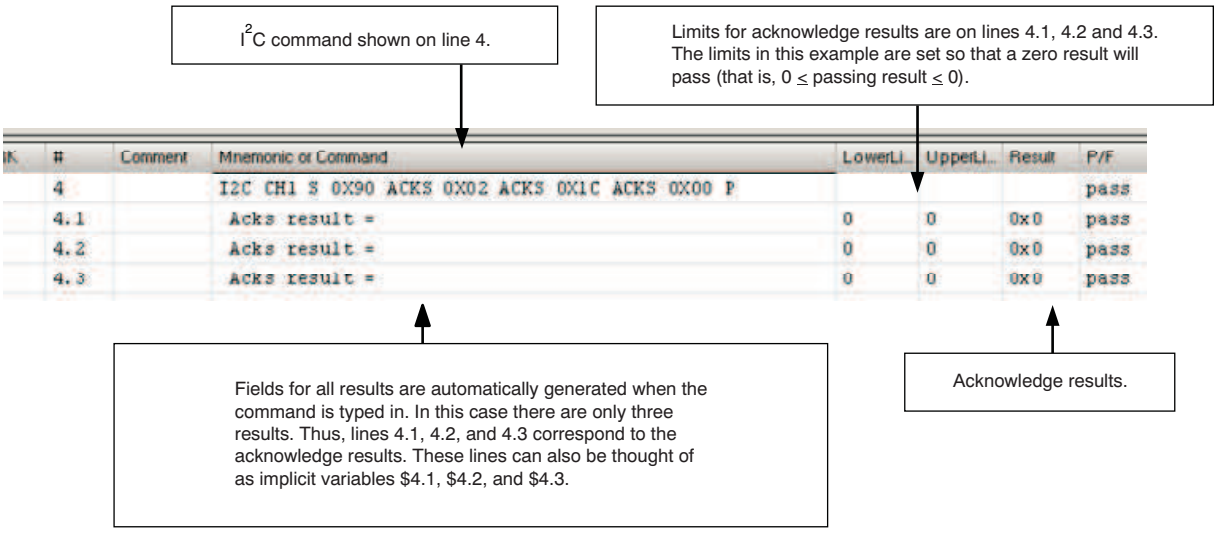


Figure 29. I²C Mnemonic Command in Script Editor (Example 1)

Figure 30 illustrates another example of how the I²C mnemonic command is generated using an I²C digital pattern. In this case, the command contains an I²C read (that is, *R*).

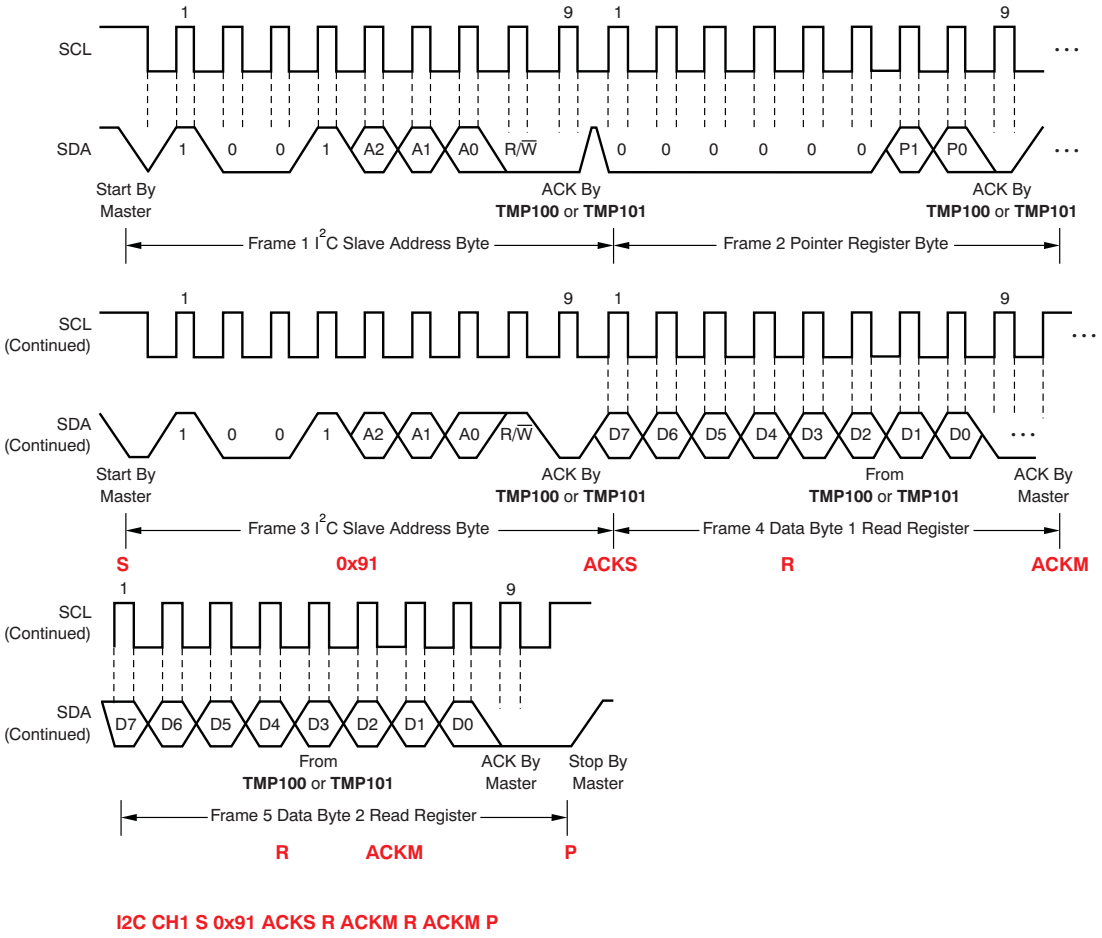


Figure 30. Generation of an I²C Mnemonic Command with an I²C Digital Pattern (with Read Command)

Figure 31 shows another example of how an I²C command appears when entered into the Script Editor. Note that lines 4.1 to 4.3 are automatically generated. These lines display the results. They can also be used as implicit variables *$4.1*, *$4.2*, and *$4.3*. Note that lines 4.2 and 4.3 are the results associated with the two 8-bit reads. The limits for lines 4.2 and 4.3 are set to pass all values from 0 to 0xFF.

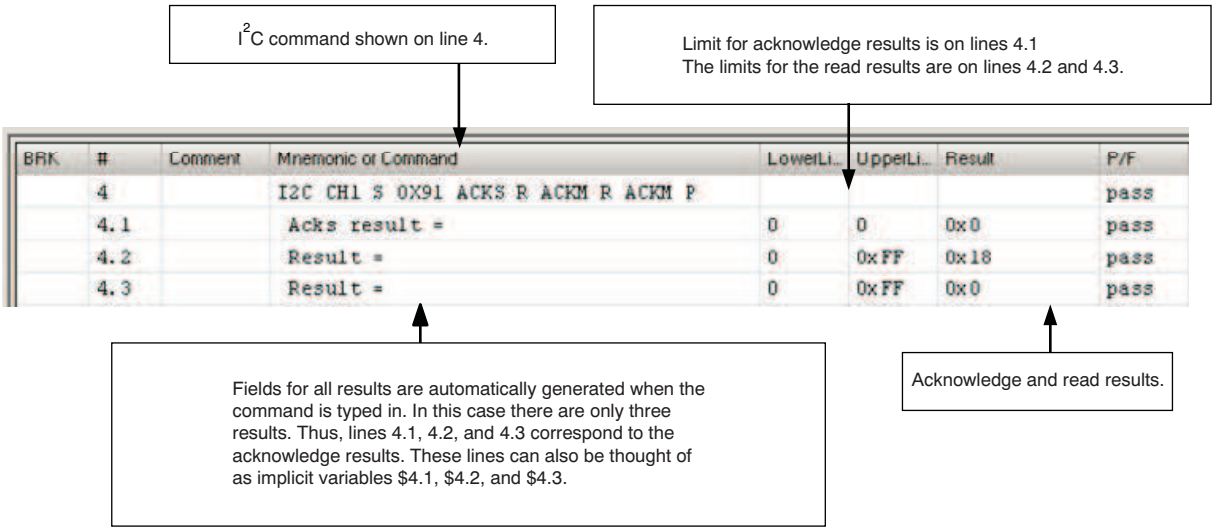


Figure 31. I²C Mnemonic Command in Script Editor (Example 2)

Table 3 lists the general commands that are associated with I²C commands. The most important of these commands is the *EN_I2C ON* command. This command enables I²C Channel 1 for communication.

Table 3. General Commands Associated with I²C Commands

Command	Explanation
CMD	This sequence is always the first mnemonic in a general command.
EN_I2C ON (or OFF)	This command is used to enable (connect) I²C channel 1.
CH1_SCL_LOW	Drive SCL low: useful in debugging fault conditions (such as a timeout)
CH1_SCL_HIGH	Drive SCL high: useful in debugging fault conditions (such as a timeout)
CH1_SDA_LOW	Drive SDA low: useful in debugging fault conditions (such as a timeout)
CH1_SDA_HIGH	Drive SDA high: useful in debugging fault conditions (such as a timeout)

5.2 SPI Mnemonic Commands

Table 4 summarizes the SPI commands. Figure 32 shows how an SPI transaction can be translated into a mnemonic command using the strings from Table 4.

Table 4. SPI Mnemonic Commands

Command	Explanation
SPI	This sequence is always the first mnemonic in an SPI command.
CH1 CH2	This sequence is always the second mnemonic in an SPI command. This mnemonic identifies which SPI channel is used. The EVM contains two SPI channels to allow for the connection of two different SPI devices.
BREH BREL AFEH AFEL	This sequence is always the third mnemonic in an SPI command. This mnemonic identifies different modes of data capture for SPI. For example, *BREL* means that the data is captured before the rising edge of the clock and the clock is active low. Further details, including timing diagrams, are shown later.
L	Drive chip select low
H	Drive chip select high
R	Read an 8-bit word
Example: **0x3A 0x21 0xBA**	Write an 8-bit word (any hex two-character byte)

Table 5 describes the four different SPI clock phase and polarity modes of operation. The USB DAQ Platform can operate in all four SPI modes.

Table 5. Different SPI Clock Phase and Polarity Modes

Mode	Clock Polarity Description (CPOL)	Clock Transition for Data Polarity Description (CPHA)
BREH	Clock idles low CPOL = 0	Data are read by the USB DAQ EVM on *rising* edge of clock. Data are changed by the USB DAQ EVM on *falling* edge of clock. Data are sampled on first clock edge. CPHA = 0
BREL	Clock idles low CPOL = 0	Data are read on *falling*e dge of clock. Data change on *rising* edge of clock. Data are sampled on second clock edge. CPHA = 1
AFEH	Clock idles high CPOL = 1	Data are read on *falling* edge of clock. Data change on *rising* edge of clock. Data are sampled on first clock edge. CPHA = 0
AFEL	Clock idles HIGH CPOL = 1	Data are read on *rising* edge of clock. Data change on *falling* edge of clock. Data are sampled on second clock edge. CPHA = 1

Figure 32 illustrates how the SPI mnemonic command is generated using an SPI digital pattern. Note that the mnemonic *SPI* is always first in SPI commands. The mnemonic *CH1* indicates which channel is used. The USB DAQ Platform has two available SPI channels. The mnemonic *BREH* sets the SPI mode (for example, the SPI clock polarity and phase). The remainder of the command (that is, *L R R 0x80 0x08 R R H)* depends on the specific SPI signal the user wants to generate.

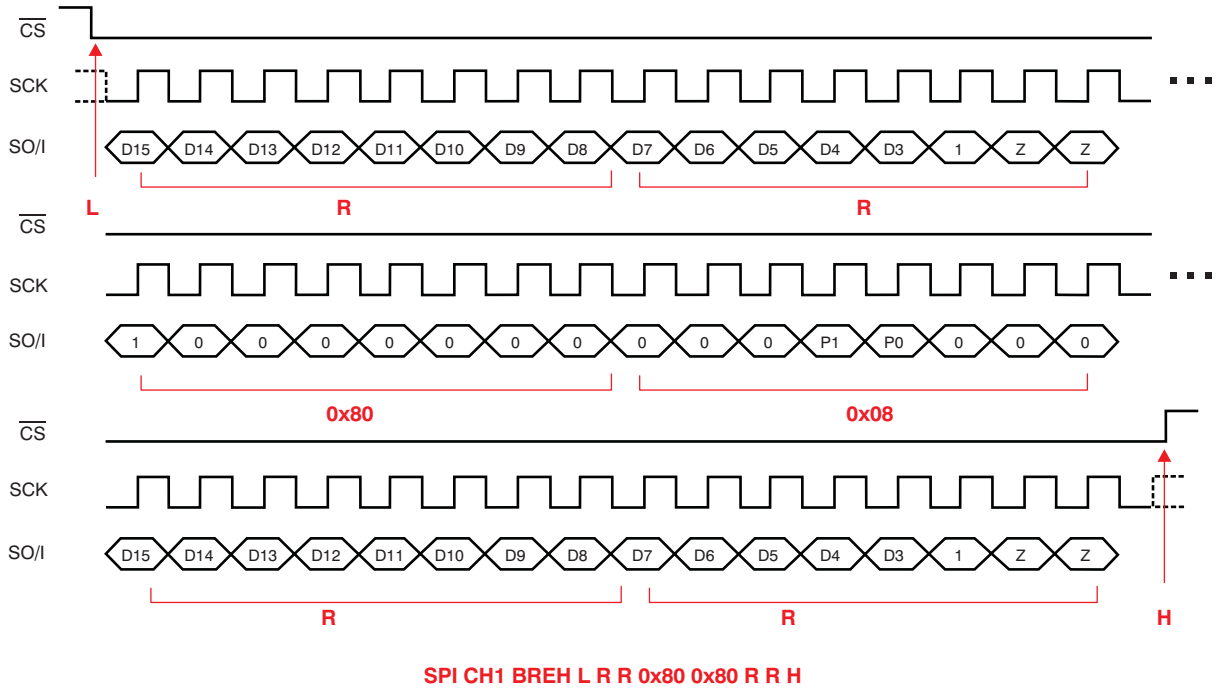


Figure 32. Generation of an SPI Mnemonic Command with an SPI Digital Pattern

Figure 33 shows how an SPI command appears when entered into the Script Editor. Note that lines 3.1 through 3.4 are automatically generated. These lines display the results. They can also be used as implicit variables *$3.1* through *$3.4*. The limits are used to test each result. In this example, line 3.4 fails the limit test while all other lines pass.

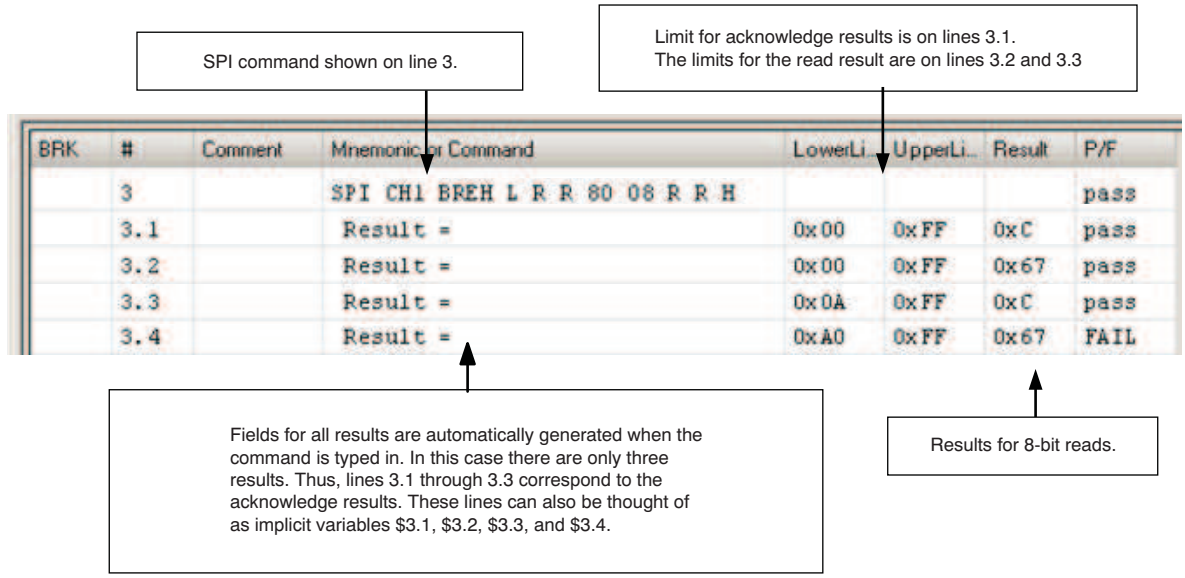


Figure 33. SPI Mnemonic Command in Script Editor

Figure 34 to Figure 37 show timing diagrams for the different SPI clock phase and polarity modes. The USB DAQ Platform can operate in all four SPI modes.

BREH: SPI Mode 00

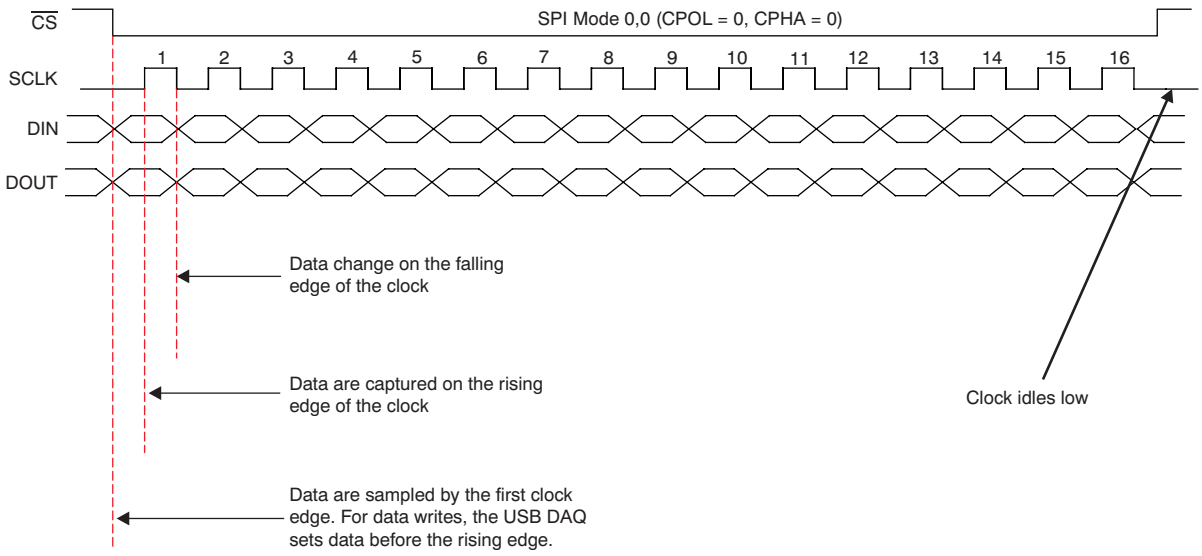


Figure 34. Clock Phase and Polarity: Mode 00

AFEH: SPI Mode 01

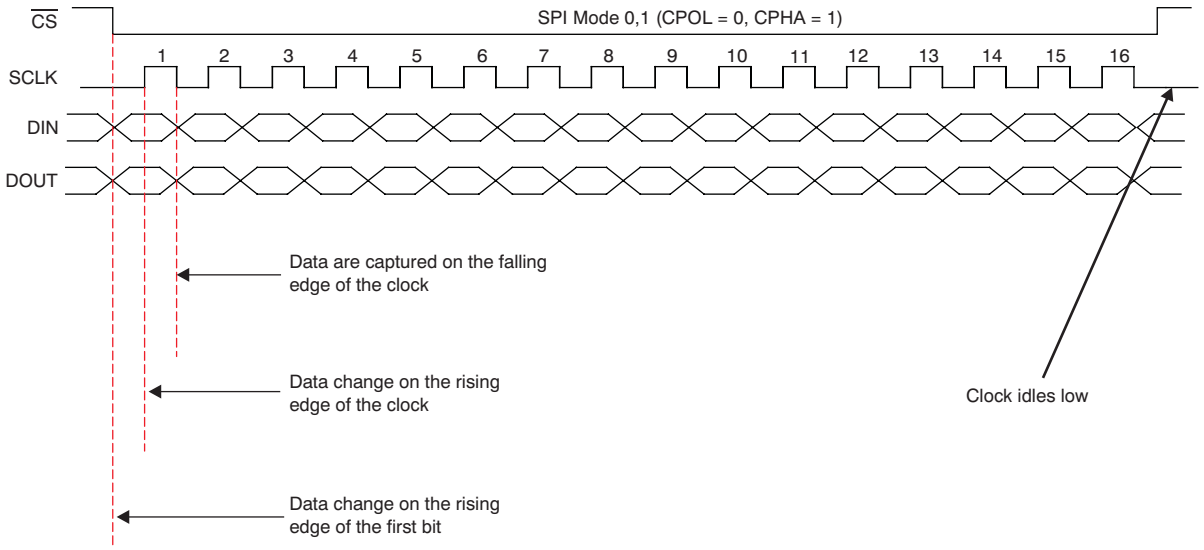


Figure 35. Clock Phase and Polarity: Mode 01

BREH: SPI Mode 10

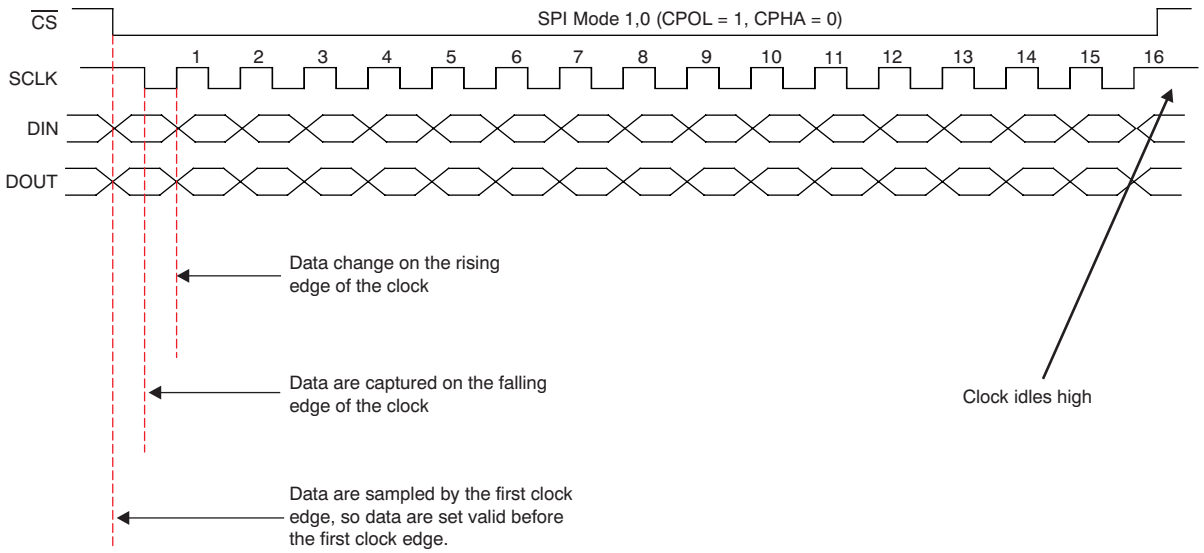


Figure 36. Clock Phase and Polarity: Mode 10

AFEH: SPI Mode 11

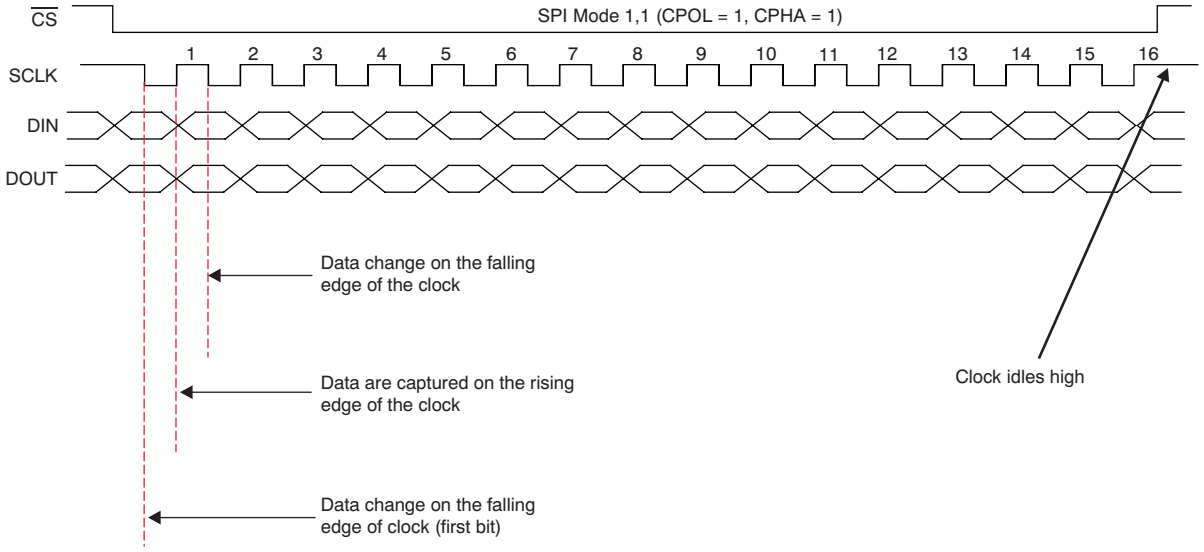


Figure 37. Clock Phase and Polarity: Mode 11

5.3 One-Wire Mnemonic Commands

Table 6 summarizes the One-Wire commands. Figure 38 and Figure 39 demonstrate how a One-Wire transaction can be translated into a mnemonic command using the strings from Table 6.

Table 6. One-Wire Mnemonic Commands

Command	Explanation
One	This sequence is always the first mnemonic in an ONE command.
R	Read an 8-bit word
Example: **3A 21 BA**	Write an 8-bit word (any hex two-character byte)

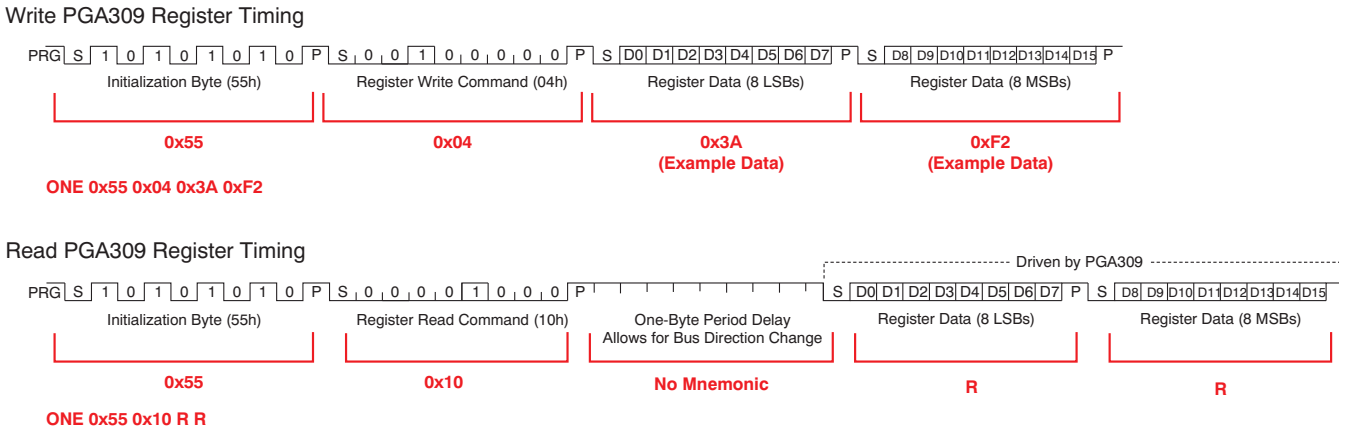


Figure 38. Generation of a One-Wire Mnemonic Command

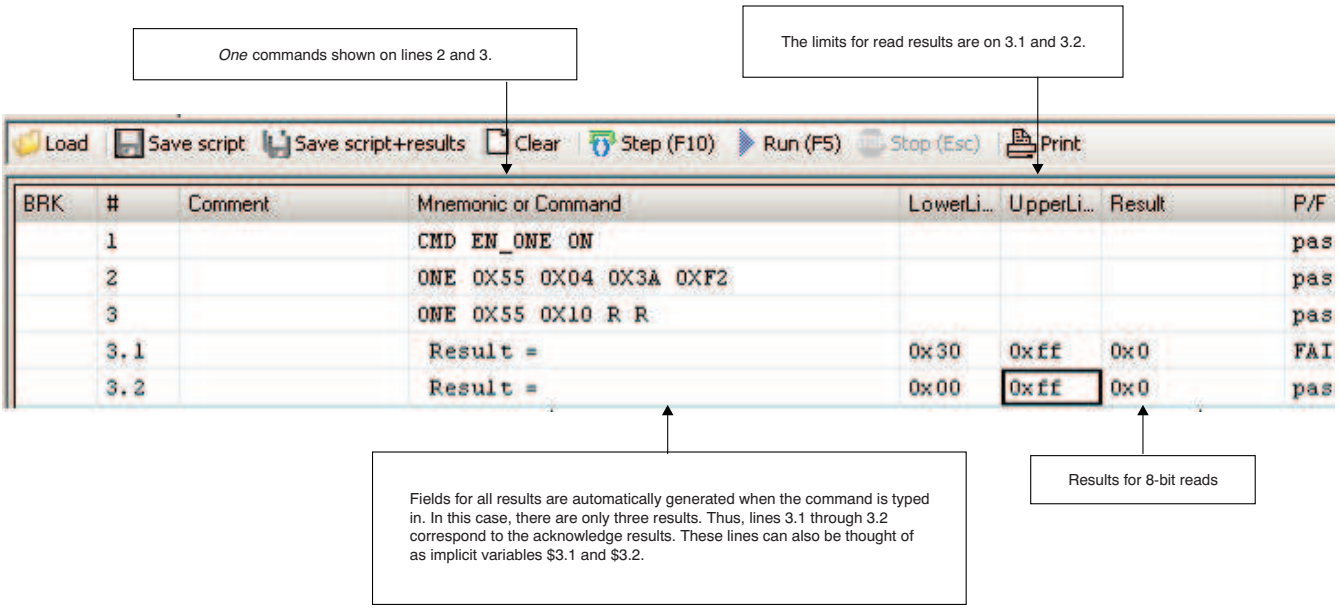


Figure 39. One-Wire Command in Script Editor

Table 7 shows some general commands that are related to the One-Wire interface.

Table 7. General One-Wire Commands

Command	Explanation
CMD	This sequence is always the first mnemonic in a general command.
EN_ONE ON (or OFF)	This command is used to enable the One-Wire interface.
WAKE_UP_ONE	This command cycles the power, waits a few milliseconds, and then issues a one-wire command. This command was developed for a special mode on specific products (for example, the PGA308 and PGA309). This mode allows for the One-Wire line and output voltage line to be connected together.

5.4 General-Purpose Mnemonic Commands

This section discusses general-purpose mnemonic commands. All general-purpose commands start with the mnemonic *CMD*. Some general-purpose commands are used in conjunction with the different interfaces. For example, there is a general-purpose command that enables and disables the I^2C interface. Other general-purpose commands perform unique functions such as turning the device power on or off.

Table 8 summarizes the general-purpose mnemonic commands.

Table 8. General-Purpose Mnemonic Commands

Command	Meaning
EN_ADS ON (or OFF)	Connects or disconnects the ADS to the DSUB connector. EN_ADS ON connects the ADS, and EN_ADS OFF disconnects the ADS. This mode of operation allows the ADS to be disconnected from the circuit under test when the DAC goes through its self calibration.
EN_ONE ON (or OFF)	Connects or disconnects the One-Wire interface from the DSUB connector (U21). EN_ONE ON connects the interface and EN_ONE OFF disconnects it.
EN_I2C ON (or OFF)	Connects or disconnects I^2C CH1 from the DSUB connector (U23 and U27). Note that I^2C CH2 is not affected by this command (this channel is always connected).
ILOOP ON (or OFF)	Connects the current loop receiver instrumentation amplifier (U32) to the ADS (U30) using a switch (U31).
RS232 ON (or OFF)	Turns the charge pump on the RS-232 translator on or off. The RS-232 port is used for firmware debug purposes. When it is not in use, the charge pump should be turned off because the charge pump generates noise.
WAKE_UP_ONE	This command cycles the power, waits a few milliseconds, and issues a One-Wire command. This command was developed for a special mode on specific products (for example, the PGA308 and PGA309). This mode allows for the One-Wire line and output voltage line to be connected together.
MEAS	Measures the digital signal on the 8 input bit port.
CTRL	Sets the digital output on the 8-bit output port.

Figure 40 shows the syntax of different general-purpose commands entered into the script editor.

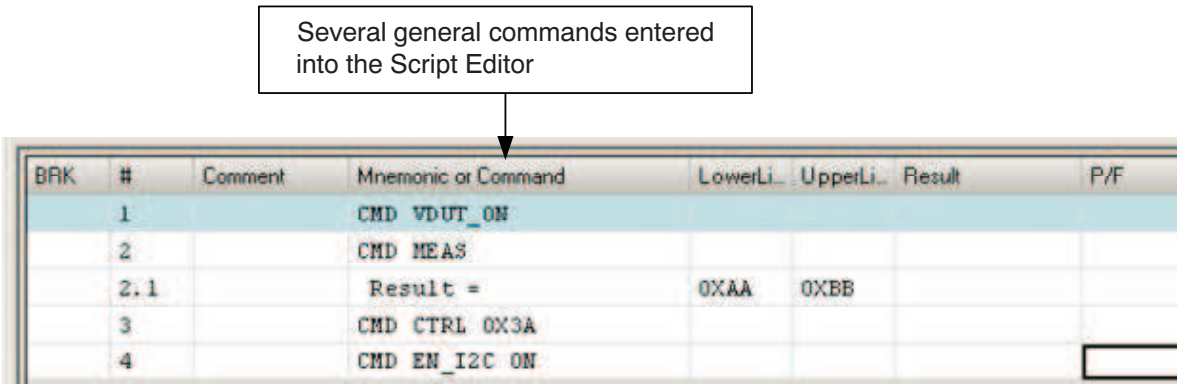


Figure 40. General-Purpose Command in Script Editor

6 High-Level Commands

The script editor uses three categories of commands: Mnemonic commands, High-Level Commands, and Script Editor Commands. This section covers High Level Commands commands.

High-level commands are typically composed of several mnemonic commands. For example, a *Write_DAC* command is composed of several I^2C writes. However, it should be noted that this distinction is most important to software developers. Script Editor users can consider the High-Level Commands and Mnemonic Commands to represent two different categories. The commands are not case-sensitive; internally, all commands convert to upper case. This case conversion also happens when a command is entered with the Script Editor.

6.1 WRITE_DAC Commands

These commands set the DAC output voltage of one of the four DAC channels. In the slow modes, the output value is measured with the ADC and the deviation is corrected. In fast modes, these commands only output the value.

Syntax:

```
WRITE_DAC_A dval mode [ref]
WRITE_DAC_B dval mode [ref]
WRITE_DAC_C dval mode [ref]
WRITE_DAC_D dval mode [ref]
```

Parameter	Description
dval	DAC output voltage value to be set. Valid range:
mode	The operation mode. Supported modes: 1. Slow mode with 5V reference, calibrated and corrected 2. Slow mode with 3V reference, calibrated and corrected 3. Slow mode with user reference, not calibrated but corrected 4. Slow mode with code output, not calibrated 5. Fast mode with 5V reference, not calibrated 6. Fast mode with 3V reference, not calibrated
ref	Optional user reference voltage used in mode 3 only

6.2 read_ads Commands

The read_ads commands read an ADC value and deliver a result depending on the operating mode.

Syntax:

```
READ_ADS1 mode [ref]
READ_ADS2 mode [ref]
```

Parameter	Description
mode	The operation mode. Supported modes: 1. Slow mode with 5V reference, calibrated and corrected 2. Slow mode with 3V reference, calibrated and corrected 3. Slow mode with user reference, not calibrated but corrected 4. Slow mode with code output, not calibrated 5. Fast mode with 5V reference, not calibrated 6. Fast mode with 3V reference, not calibrated
ref	Optional user reference voltage used in mode 3 only

6.3 ReadDoubleFromEEPROM Command

Reads a double value from the EEPROM. The value is read from three consecutive 8-bit memory cells. The double result is stored in the implicit result value.

Syntax:

```
READDOUBLEFROMEEPROM adr
```

Parameter	Description
adr	The address to be read from

6.4 WriteDoubleToEEPROM Command

This instruction writes a double value into the EEPROM. The value is written as three consecutive 8-bit values into the memory cells.

Syntax:

```
WRITEDOUBLETOEEPROM val adr
```

Parameter	Description
Val	The double value to be written
adr	The address to be read from

6.5 ads_mux Commands

These commands set the MUX1/MUX2 input lines.

Syntax:

```
ADS_MUX1 inp inn
ADS_MUX2 inp inn
```

Parameter	Description
inp	The in_p option (1-4)
inn	The in_n option (1-4)

6.6 SetCurrVoltMode Command

This command sets either current or voltage mode.

Syntax:

```
SETCURRVOLTMODE val
```

Parameter	Description
val	1. Current mode 2. Voltage mode

6.7 ReadILoop Command

This function reads the 4mA to 20mA current loop. It does this by reading the ADS1100 (U24) and applies the appropriate calibration constants. Before this function is used, the *SetCurrVoltMode* must be set to current mode.

Syntax:

```
READILOOP
```

Parameter	Description
none	

7 Script Editor Commands

Script Editor commands do not affect the USB DAQ Platform. Script Editor commands control the execution of the script. Stop, for example, stops the execution of a script in progress.

7.1 Pause Command

Pauses the execution for *x* milliseconds.

Syntax:

```
PAUSE X
```

Parameter	Description
X	Duration of the pause in milliseconds

7.2 Stop Command

Stops the execution. The buttons on the toolstrip are enabled again. It is possible to continue execution with the *Step* or *Run* button starting from the actual (blue marked) line.

Syntax:

```
STOP
```

Parameter	Description
none	

8 Mathematical Calculations

The Script Editor contains a means for interpreting mathematical equations. Equations are formulated according to general mathematical principles. Arithmetical, relational, and logical operators are supported and handled correctly according to the respective range. Operators use the same symbols as in the C language.

Braces can be used to affect the order of the calculation and can be nested to any level. Syntax errors as missing arguments, missing operators, and braces match imbalance are recognized.

Here are several example mathematical calcuations:

```
$FORCE = $F_SLOPE * $V_PGA + $F_OFFSET
$VAL1 = ($VAL2 | $VAL3) & !$VAL4
$VALID = ($4.1 > $L_LEVEL) && ($4.1 < $H_LEVEL)
$REG2 = (($MSB << 8) | $LSB) & 0XFFFF
$BIT4 = $10.2 & 0X10
```

8.1 Operands

Operands can be variables, handled by their names or numerical and boolean literals.

8.1.1 Variables

For the format of variable names see Figure 7. User-defined and implicit variables may be used. Supported data types are bool, int and double.

Variables at the right side of the equation must already have a valid data entry. If not the calculation will fail, giving an invalid result, reported as an error. Calculations with a failure give an invalid variable result type, also reported as an error.

8.1.2 Numerical and Boolean Literals

Boolean literals are FALSE and TRUE.

Int Literals are accepted as hex value by prefix 0x, e.g. 0X1234, or decimal, e.g. 10, 3456, 300 , there must not be a period or a comma.

Double literals are recognized by a number containing a period. Examples are:

.725 ; 1.0 ; 2.5 ; 0.00033 ; 5400.375

The exponential format is also supported. Exampls are:

.333E2 ; 1.25E+3 ; 3.25E-6

Note that these literals are always positive numbers because a preceding minus would be handled separately as a unary minus before recognizing the literal.

8.2 Supported Operators

Table 9 lists all suported operators sorted by their range with their supported data types, the resulting data type and the association (left | right).

Table 9. Summary of Supported Operators

Sign	Name	Supported Operands Data Types	Result Data Type	Range	Association
(	left brace	—	—	15	L
)	right brace	—	—	15	L
!	not	bool	bool	14	R
~	1's complement	int	int	14	R
-	unary minus	int, double	int, double	14	R
*	multiplication	int, double	int, double	13	L
/	division	int, double	int, double	13	L
%	modulo	int	int	13	L
+	addition	int, double	int, double	12	L
-	subtraction	int, double	int, double	12	L
<<	left shift	int	int	11	L
>>	right shift	int	int	11	L
<	less than	int, double	bool	10	L
<=	less or equal	int, double	bool	10	L
>=	greater or equal	int, double	bool	10	L
>	greater than	int, double	bool	10	L
= =	equals	bool, int, double	bool	9	L
!=	not equal	bool, int, double	bool	9	L
&	and	bool, int	bool, int	8	L
^	xor	bool, int	bool, int	7	
|	or	bool, int	bool, int	6	L
&&	conditional and	bool	bool	5	L
||	conditional or	bool	bool	4	L

8.3 Resulting Numerical Data Types

If an operator supports different numerical data types (such as arguments) and the result is numerical, then the data type of the result follows the argument with the highest resolution. For example:

int *op* **int**	results in	**int**
int *op* **double**	results in	**double**
double *op* **int**	results in	**double**
double *op* **double**	results in	**double**

9 Script File Format

The script is stored in .CSV format files with the colon (**;**) as a separator. The contents of the breakpoint column are not stored, so a line starts with the line number column (#).

	1	# Test the CTRL and MEAS Comm…		
	2		CMD CTRL 0XF0	
	3		CMD MEAS	
	3.1		result value =	0
	4	comment	CMD MEAS	
	4.1		result value =	1

Figure 41. Example Script with Variables

The output in Figure 41 corresponds to:

```
1;# Test the CTRL and MEAS Command;;;
2;;CMD CTRL 0XF0;;
3;;CMD MEAS;;
3.1;; result value =;0;
4;comment;CMD MEAS;;
4.1;; result value =;1;
```

The script files are stored with a *.tsc* file extension for a tio32 script file.

One can edit these files with any text editor such as Notepad. It is not necessary to write the result line; they are inserted automatically when the script is loaded.

As noted earlier in this document, it is possible to associate .tsc files with the tioScript.exe application so that double-clicking on a script file launches the tioScript.exe program automatically.

EVALUATION BOARD/KIT IMPORTANT NOTICE

Texas Instruments (TI) provides the enclosed product(s) under the following conditions:

This evaluation board/kit is intended for use for **ENGINEERING DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY** and is not considered by TI to be a finished end-product fit for general consumer use. Persons handling the product(s) must have electronics training and observe good engineering practice standards. As such, the goods being provided are not intended to be complete in terms of required design-, marketing-, and/or manufacturing-related protective considerations, including product safety and environmental measures typically found in end products that incorporate such semiconductor components or circuit boards. This evaluation board/kit does not fall within the scope of the European Union directives regarding electromagnetic compatibility, restricted substances (RoHS), recycling (WEEE), FCC, CE or UL, and therefore may not meet the technical requirements of these directives or other related directives.

Should this evaluation board/kit not meet the specifications indicated in the User's Guide, the board/kit may be returned within 30 days from the date of delivery for a full refund. THE FOREGOING WARRANTY IS THE EXCLUSIVE WARRANTY MADE BY SELLER TO BUYER AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

The user assumes all responsibility and liability for proper and safe handling of the goods. Further, the user indemnifies TI from all claims arising from the handling or use of the goods. Due to the open construction of the product, it is the user's responsibility to take any and all appropriate precautions with regard to electrostatic discharge.

EXCEPT TO THE EXTENT OF THE INDEMNITY SET FORTH ABOVE, NEITHER PARTY SHALL BE LIABLE TO THE OTHER FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

TI currently deals with a variety of customers for products, and therefore our arrangement with the user **is not exclusive.**

TI assumes **no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein.**

Please read the User's Guide and, specifically, the Warnings and Restrictions notice in the User's Guide prior to handling the product. This notice contains important safety information about temperatures and voltages. For additional information on TI's environmental and/or safety programs, please contact the TI application engineer or visit www.ti.com/esh.

No license is granted under any patent right or other intellectual property right of TI covering or relating to any machine, process, or combination in which such TI products or services might be or are used.

FCC Warning

This evaluation board/kit is intended for use for **ENGINEERING DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY** and is not considered by TI to be a finished end-product fit for general consumer use. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

EVM WARNINGS AND RESTRICTIONS

It is important to operate this EVM within the input voltage range of 6V to 9V and the output voltage range of 0V to 5V.

Exceeding the specified input range may cause unexpected operation and/or irreversible damage to the EVM. If there are questions concerning the input range, please contact a TI field representative prior to connecting the input power.

Applying loads outside of the specified output range may result in unintended operation and/or possible permanent damage to the EVM. Please consult the EVM User's Guide prior to connecting any load to the EVM output. If there is uncertainty as to the load specification, please contact a TI field representative.

During normal operation, some circuit components may have case temperatures greater than +85°C. The EVM is designed to operate properly with certain components above +85°C as long as the input and output ranges are maintained. These components include but are not limited to linear regulators, switching transistors, pass transistors, and current sense resistors. These types of devices can be identified using the EVM schematic located in the EVM User's Guide. When placing measurement probes near these devices during operation, please be aware that these devices may be very warm to the touch.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265