Version	Modification
V1.0	Official version

Introduction

This document only gives command instructions for this series of signal sources. Please refer to the document of **<UCI Help>** for the interface guideline. Please contact the technical support if the command can’t be used normally.

Reference

1. UTG4162.h:
   The basic definition of UTG4000A is given.
2. UCI related files: please refer to the document of <UCI Help>

Basic Format of Command String:

Every command is executed by string communication with device, the basic format of command string is {Field name of command: the field value of command…;}

Including：

The string before : represents the name of command
The string after : represents the value of fields
The punctuation of ; represents the end of a command, please refer to <**UCI Help**> for details.

Terminology: SG – Signal Source

General Commands：

IDN?

Get the name of device
Data format: Name% internal information #SN serial number; data volume 54Bytes.
For example: UTG8000B%**#SN005

Keys:

Name of Command	Command Parameters	Type of Command Parameters
KEY	Key value	See the key code table below

KEY	Encoding	KEY	Encoding
F1	F1	0	0
F2	F2	1	1
F3	F3	2	2
F4	F4	3	3
F5	F5	4	4
F6	F6	5	5
USER	USER	6	6
Digital	dig	7	7
Counter	cnt	8	8
MOD	mod	9	9
Sweep	swp	.	.
Burst	bst	+/-	SIGN
Knob Left	FKNL	Trigger	TG

Knob Right	FKNR	CH1	C1
Knob Click	FKN	CH2	C2
CH1 X CH2	XCH	Left	L
Sine	wsn	Right	R
Noise	wns	Preset	PST
Square	wsq	Storage	STG
Ramp	wrp	StorageL	STGL
Pulse	wps	Utility	UTIL
Arb	warb	Help	HP
Harmonic	whm	HelpL	hpl
DC	wdc	Page	PG

Item	Description	IO	Data
Lock	Lock the key	W	No data
Unlock	Unlock the key	W	No data
Lock?	Check the status of	**R**	Integer<4Bytes>： 0 – Unlock; 1 – locked
Led?	Check the status of LED	**R**	Integer<4Bytes>： 0 – LED off; 1 – Green LED on;

Example:

“KEY:c1;　　--　　CH1
“KEY:c2;”　　--　　CH2
“KEY:c2@lock;”　　--　　CH2 lock the key
“KEY:c2@unlock;”　　--　　CH2 unlock the key
“KEY:c2@lock?;”　　--　　Check the status of the key lock
“KEY:c2@led?;”　　--　　Check the status of LED, which is similar to the command of “LED;”

Notice:

Please refer to the description of uci_FormatWrite in the document of **<UCI Help>**.

uci_Read should be used to read the command with question mark, which can be accessed by buffer zone or interface return value.

Furthermore

Check the lock status utility commands of all the keys

Name of Command	Command Parameter	Data Format
lock?	Null	64 –bit integer, 0x086FFFFFFFFFFFFF, indicates full lock

Data can be accessed by the interface uci_Read, and the status of every single key can be accessed by the interface IsKeyLocked (please refer to UTG4162.h).

Local/remote status switching

Name of Command	Command Parameter	Command Parameter Types
local	Status encoding	Enum(Integer<4Bytes>):0/1{remote status/local status}
Local?	Null	Enum(Integer<4Bytes>):0/1{remote status/local status}

Example：

"local:0;" – remote status, fully lock the keys

"local:1;" –local status, fully unlock the keys

"local?;" – check the status

Notice:

Please refer to the description of uci_FormatWrite in the document of <UCI Help>.

uci_Read should be used to read the command with question mark, which can be accessed by buffer zone or interface return value.

Screenshot:

Name of Command	Command Parameter	Command Parameter Types
PrtScn	Image format	Enum(String):space/zip/bmp/png {Pixel data/Compressed pixel data/BMP file/png file}

Example:

"PrtScn:png;"	--- Save the screenshot as png file
"PrtScn:bmp;"	--- Save the screenshot as bmp file
"PrtScn;"	--- Save the screenshot as pixel data
"PrtScn:zip;"	--- Save the screenshot as compressed pixel data

Notice:

Read data by uci_Read. This command does not save the data directly to the file, but returns it to the buffer zone specified by uci_Read.

Please save first if it needs to buffer to local files.

1. If the command of **PrtScn;** is used, the size of buffer zone must be 800*480*4=1536000, can read 32 bits pixel data
2. If the command of **PrtScn:bmp;** is used, the size of buffer zone must be 800*480*3+54=1152054, which is the size of bitmap file.
3. If the command of **PrtScn: png;** is used, the size of buffer zone can be set as 800*480*4=1536000, this is the maximum value, and PNG is data of image compression, will be less than this (1536000), valuable data length can be obtained by the return value of uci_Read interface.
4. If the command of **PrtScn:zip;** is used, the size of buffer zone can be set as 800*480*4=1536000 (maximum value). The compressed pixel data is read, then the data should be decompressed by alg_UnCompressPixels interface. Please be noticed the compressed data is returned by uci_Read.
5. The bitmap can be saved to the files by uci_ReadToFile interface and the command of **prtscn:bmp;**.

The comparison of efficiency and stability:

PrtScn; has the highest efficiency and stability. If frequent refreshes are needed, this interface can be considered. **PrtScn:bmp;** is less efficient because its bitmap file is larger.

The transmission of **PrtScn:png;** is stable but less efficient, because the size of compressed data is small, but the compression itself is time-consuming.

Write arbitrary waveform files:

Name of Command	Command Parameter	Command Parameter Types
WARB	Null	Null

Char Name	Significance	IO	Data
CH	Channel Number	W	Enum(Integer) : 0/1{ CH1/ CH2 }
Mode	Load Mode	W	Enum(Integer): 0/1/R {Carrier/Mod}
Disk	Storage Medium	W	Enum(Integer): 0/1/2/3{RAM/ROM/TF/U-DISK}

Example:

“WARB@CH:0@MODE:0@DISK:0;”

Load the waveform files in CH1 by saving in RAM with the form of carrier , which will lose after restart.

Notice:

Write arbitrary waveform files by uci_WriteFromFile. Set the time-out to be 1000.

Read and write the configuration files

Name of Command	Command Parameter	Command Parameter Types
dconfig	Null	Null

Example：

“dconfig;”

Notice：

Read the the configuration files by uci_ReadToFile, and write configuration files by uci_WriteFromFile. The configuration files are suffixed by “.set”, so please ensure the suffix of documents are .set during reading and writing.

Versions:

Name of	Command	Command Parameter Types
Ver?;	Null	Null

Example：

“Ver?;”

Interface：

Read the data by the interface of uci_Read

The data size is 54 bytes.

Communication protocol version:

Check the version number of system information to check if the protocol interface is supported.

Name of Command	Command Parameter	Command Parameter Types
CVer?;	Null	Null

Example：

“CVer?;”

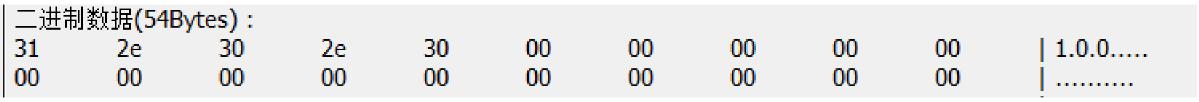
Interface：

Read the data by uci_Read

The data size is 54 bytes.

Data：

The binary data (54Bytes)



```
二进制数据(54Bytes)：
31    2e    30    2e    30    00    00    00    00    00    | 1.0.0.....
00    00    00    00    00    00    00    00    00    00    | ..........
```

Name of Command	Command Parameter	Command Parameter Types
rp	Null	Null

Char Name	Significance	IO	Data
CH	Channel Number	W	Enum(Integer) : 0/1{ CH1/ CH2 }
addr	Parameter Address	W	Enum(ERemoteMessage): check the significance of parameter address.

Example：

“rp@CH:0@addr:0x8009;” - read the frequency of CH1

Notice：

The interface of UCI corresponds to uci_Read, and the corresponding data size is 8 bytes, dobule types

Interface:

Read the parameters by uci_Read or uci_ReadX;

Write the parameters:

Name of Command	Command Parameter	Command Parameter Types
wp	Null	Null

Char Name	Significance	IO	Data
CH	Channel Number	W	Enum(Integer) : 0/1{ CH1/ CH2 }
addr	Parameter Address	W	Enum(ERemoteMessage): check the significance of parameter address
v	Parameter Value	W	The physical unit depends on the significance of each parameter.

Example:

“wp@CH:0@addr:0x8009@v:2000;” - set the frequency of CH1 to be 2kHz;

Notice:

The interface of UCI corresponds to uci_Write, the types of all the data are double, and the data size is 8Bytes.

Interfaces:

Write parameters by uci_Write, uci_WriteX or uci_FormatWrite

Check the status of LED:

Name of Command	Command Parameter	Command Parameter Types
LED	Null	Null

Example:

LED;

Notice:

Read the status data by uci_Read, and the format of data is LEDStatus (please refer to the definition in UTG4162.h file)

Auto Reconnection:

Name of Command	Command Parameter	Command Parameter Types
Reconnect;	Null	Null

Example:

Reconnect;

Notice:

Send the command by *uci_SendCommand*. After connecting, the detected device is disconnected and you can try to reconnect by sending the command. The way to check the equipment online status is to regularly check the LED status, which is to execute the command of LED; and determine the line state by interface return value.

Appendix

Parameter Definition:

The following definitions come from: include\UTG4162.h **Explanatory Note**

Such as: {IO:WR}{DATA:0-OFF，1-ON，2-reverse}

IO defines: the read-write type of parameters: W writable parameter; R readable parameter

Data defines: the value of data

```
////////////////////////////////Parameters AddressDefine////////////////////////////
        //Work Mode
        typedef enum _EWorkMode : long long{
            //@brief : Fundamental Wave Mode
            WM_BASE = 0,
            //@brief : Modulation Wave Mode
            WM_MODE,
            //@brief : Frequency Sweep Mode
            WM_SWEEP,
            //@brief : Burst
            WM_BURST,
        }EWorkMode;

        //Basic Waveform
        typedef enum _EBaseWave : long long{
            //@brief : Sine Wave
            BASE_SINE = 0,
            //@brief : Square Wave
            BASE_SQUARE = 1,
            //@brief : Ramp Wave
            BASE_RAMP = 3,
            //@brief : Pulse Wave
            BASE_PULSE = 4,
            //@brief : Noise
            BASE_NOISE = 5,
            //@brief : Arbitrary Wave
            BASE_ARB = 6,
            //@brief : Harmonic Wave
            BASE_HARMONIC = 7,
            //@brief : DC
            BASE_DC = 8
        }EBaseWave;
```

```
typedef enum _EModeType : long long { MT_AM = 0,
        MT_FM = 1,
        MT_PM = 2,
        MT_ASK = 3,
        MT_FSK = 4,
        MT_PSK = 5,
        MT_BPSK = 6,
        MT_QPSK = 7,
        MT_OSK = 8,
        MT_QAM = 9,
        MT_PWM = 10,
        MT_SUM = 11,
}EModeType;

//Modulation Waveform
typedef enum _EModeWaveType : long long
        { MOD_WAVE_SINE = 0,
        MOD_WAVE_SQUARE = 1,
        MOD_WAVE_UPRAMP = 2,
        MOD_WAVE_DNRAMP = 3,
        MOD_WAVE_NOISE = 4,
        MOD_WAVE_ARB = 5,
}EModeWaveType;

//@brief : Signal Source Parameter Coding
//@remark: read parameters by command of rp, write parameters by command of wp
//@example: wp@ch:0@addr:0x8000@v:0;
//                    rp@ch:0@addr:0x8000;
//The corresponding data amount for all parameters is 8bytes, double type
typedef enum _enumRemoteMessage {
        //@brief : Work Mode
        //@remark: {IO:WR}{DATA:EWorkMode}
        RM_WORK_MODE = 0x8000,

        //{ Fundamental Wave
        //@brief : Channel Switch
        //@remark: {IO:WR}{DATA: 0-OFF，1-ON}
        RM_CH_SW = 0x8001,
        //@brief : Synchronous Switch
        //@remark: {IO:WR}{DATA: 0-OFF，1-ON，2-reverse} RM_CH_SYNC_SW,
        //@brief : Channel Reverse
        //@remark: {IO:WR}{DATA:  0：OFF，1：ON}
```

```
RM_CH_REVERTSE,
//@brief : Channel Load Impedance
//@remark: {IO:WR}{DATA: 1~1000}
RM_CH_LOAD,
//@brief : Channel Output Limit Enablement
//@remark: {IO:WR}{DATA:            0： OFF， 1： ON}
RM_CH_OUTPUT_LIMIT_ENABLE,
//@brief :  Lowest Limit of Channel Output Level
//@remark: {IO:WR}{DATA<?>:              -10V~MAX_LEVEL}
RM_CH_OUTPUT_LIMIT_MIN_LEVEL,
//@brief : Highest Limit of Channel Output Level
//@remark: {IO:WR}{DATA:             MIN_LEVEL~10V}
RM_CH_OUTPUT_LIMIT_MAX_LEVEL,

//@brief : Type of Fundamental Wave
//@remark: {IO:WR}{DATA:EBaseWave}
RM_BASE_WAVE_TYPE = 0x8008,
//@brief : Frequency of Fundamental Wave (Unit: Hz)
//@remark: {IO:WR}{DATA:1uHz~Maximum frequency of current fundamental wave }
RM_BASE_FREQ,
//@brief : Phase of Fundamental Wave (Unit: °)
//@remark: {IO:WR}{DATA:-360~360}
RM_BASE_PHASE = 0x800A,
//@brief : Range of Fundamental Wave (Unit: VPP)
//@remark: {IO:WR}{DATA: 1mVpp~10Vpp (50Ω) } RM_BASE_AMP_VPP,
//@brief : Range of Fundamental Wave (Unit: VRMS)
//@remark: {IO:WR}{DATA:1mVRMS~5Vpp (50Ω) } RM_BASE_AMP_VRMS,
//@brief : Range of Fundamental Wave (Unit: VDBM)
//@remark: {IO:WR}{DATA:-53.010VDBM~26.99VDBM (50Ω) }
RM_BASE_AMP_VDBM,
//@brief : Offset of Fundamental Wave (Unit: V)
//@remark: {IO:WR}{DATA:-5VRMS~5Vpp (50Ω) } RM_BASE_OFFSET,
//@brief : High Fundamental Wave Level (Unit: V)
//@remark: {IO:WR}{DATA:BASE_LOW~5Vpp (50Ω) } RM_BASE_HIGHT = 0x800F,
//@brief : Low Fundamental Wave Level (Unit: V)
//@remark: {IO:WR}{DATA:-5VRMS~BASE_HIGHT (50Ω) } RM_BASE_LOW = 0x8010,
//@brief : Duty Cycle of Fundamental Wave : Duty Cycle of Square Wave
//@remark: {IO:WR}{DATA:0~100}
RM_BASE_DUTY,
//@brief : Pulse Wave Rise Time (Unit: s)
//@remark: {IO:WR}{DATA:min ~ cycle*0.4}
RM_BASE_RISETIME,
//@brief : Pulse Wave fALL Time (Unit: s)
//@remark: {IO:WR}{DATA:min ~ cycle*0.4}
RM_BASE_FALLTIME,
//@brief : Arbitrary Wave Play Mode Switch
//@remark: {IO:WR}{DATA:0: OFF, 1: ON}
RM_BASE_ARB_PLAY_ENABLE,
//@brief : Symmetry Degree of Ramp Wave
//@remark: {IO:WR}{DATA:0~100}
RM_BASE_SYMMETRY,,

//@brief : Harmonic Wave - Work Mode
//@remark: {IO:WR}
//{DATA:
//0:        Odd-order,
//1:        Even-order,
//2:        All,
//3:        USER, user-defined
// }
RM_BASE_HARMOIC_TYPE = 0x8080,
//@brief: Harmonic wave switch, which is valuable when RM_BASE_HARMOIC_TYPE=USER
//@remark: {IO:WR}{DATA: BIT14-BIT0  correspond to the switches of 2~16 harmonic wave respectively, BIT15 is forced open corresponds to fundamental wave} RM_BASE_HARMONIC_ONOFF,
//There are 2 ways to set N(2~16) harmonic wave:
//EG:N=3 (Third Harmonic Wave), AMP = 1Vpp, PHASE = 90°
//方法 1:
//（1）Specified harmonic number, RM_HARMONIC_NUM = N(2~16),
//（2）Set the range and phase of harmonic wave,
//          RM_HARMONIC_SN_AMP_N = 1.0; RM_HARMONIC_SN_PHASE_N = 90.0;
//Method 1: set the range and phase of the third harmonic directly
//RM_HARMONIC_SN_AMP_3 = 1.0, RM_HARMONIC_SN_PHASE_3 =90.0,

//@brief : Number of harmonic wave
//@remark: {IO:WR}{DATA:1~15}
RM_HARMONIC_NUM,
//@brief : Range of harmonic wave Vpp
//@remark: {IO:WR}{DATA:0~ Range of fundamental wave}
RM_HARMONIC_SN_AMP_N,
//@brief : Phase of harmonic wave (Unit: °)
//@remark: {IO:WR}{DATA:-360~360}
RM_HARMONIC_SN_PHASE_N = 0x8084,
//@brief : Range of harmonic wave is as same as RM_HARMONIC_SN_AMP_N
```

```
RM_HARMONIC_SN_AMP_2,
RM_HARMONIC_SN_AMP_3,
RM_HARMONIC_SN_AMP_4,
RM_HARMONIC_SN_AMP_5,
RM_HARMONIC_SN_AMP_6,
RM_HARMONIC_SN_AMP_7 = 0x808A ,
RM_HARMONIC_SN_AMP_8,
RM_HARMONIC_SN_AMP_9,
RM_HARMONIC_SN_AMP_10,
RM_HARMONIC_SN_AMP_11,
RM_HARMONIC_SN_AMP_12 = 0x808F,
RM_HARMONIC_SN_AMP_13 = 0x8090,
RM_HARMONIC_SN_AMP_14,
RM_HARMONIC_SN_AMP_15,
RM_HARMONIC_SN_AMP_16,
//@brief : Harmonic Wave Phase RM_HARMONIC_SN_PHASE_N
RM_HARMONIC_SN_PHASE_2,
RM_HARMONIC_SN_PHASE_3,
RM_HARMONIC_SN_PHASE_4,
RM_HARMONIC_SN_PHASE_5,
RM_HARMONIC_SN_PHASE_6,
RM_HARMONIC_SN_PHASE_7,
RM_HARMONIC_SN_PHASE_8 = 0x809A,
RM_HARMONIC_SN_PHASE_9,
RM_HARMONIC_SN_PHASE_10,
RM_HARMONIC_SN_PHASE_11,
RM_HARMONIC_SN_PHASE_12,
RM_HARMONIC_SN_PHASE_13 = 0x809F,
RM_HARMONIC_SN_PHASE_14 = 0x80A0,
RM_HARMONIC_SN_PHASE_15,
RM_HARMONIC_SN_PHASE_16,
//}

//{MOD
//@brief : Modulation Mode
//@remark: {IO:WR}{DATA:EModeType}
RM_MOD_TYPE = 0x8100,

//Modulation Waveform
//0:  MOD_SINE,
//1:  MOD_SQUARE,
//2:  MOD_UPRAMP,
//3:  MOD_DNRAMP,
//4:  MOD_NOISE,
//5:  MOD_ARB,
//@brief : Modulation Waveform
//@remark:{IO:WR}{DATA:EModeWaveType}
RM_MOD_WAVE,
//@brief : Frequency of Modulation Wave (Unit: Hz)
//@remark: {IO:WR}{DATA: 1uHz~200KHz}
RM_MOD_FREQ,
//@brief : Rate of modulation wave (Unit: s)
//@remark: {IO:WR}{DATA<double>: 2ms~1Ms}
RM_MOD_RATE,
//@brief : Modulation depth (Unit: %)
//@remark: {IO:WR}{DATA: 0~120}
RM_MOD_SCOPE,
//@brief : Modulation source
//@remark: {IO:WR}{DATA: 0-Internal,1-External} RM_MOD_SOURCE,
//@brief : FM Frequency difference (Unit: Hz)
//@remark: {IO:WR}{DATA: 0~current frequency of carrier wave}
RM_MOD_FRE_DEV,
//@brief : PM phase difference          (Unit: °)
//@remark: {IO:WR}{DATA:-360~360}
RM_MOD_PHASE_DEV,
//@brief : FSK frequency hopping (Unit: Hz)
//@remark: {IO:WR}{DATA: 0~maximum frequency of carrier
wave} RM_MOD_HOP_FREQ,
//@brief : BPSK modulation data source
//@remark: {IO:WR}
//{DATA<double>:
//0:   PN7,
//1:   PN9,
//2:   PN15,
//3:   PN21,
//}
RM_MOD_DATA_SOURCE = 0x8109,
//@brief : BPSK phase position, QPSK phase position 1, (Unit: °)
//@remark: {IO:WR}{DATA:-360~360}}
RM_MOD_PSK_PHASE1,
//@brief : QPSK phase position 2 (Unit: °)
//@remark: {IO:WR}{DATA:-360~360}}
RM_MOD_PSK_PHASE2,
//@brief : QPSK phase position 3 (Unit: °)
//@remark: {IO:WR}{DATA:-360~360}
RM_MOD_PSK_PHASE3,
//@brief : OSK vibration time (Unit: s)
//@remark: {IO:WR}{DATA: 8ns~200s}
```

```
RM_MOD_OSC_TIME,
//@brief : QAM modulate IQ MAM
//@remark: {IO:WR}{DATA:0-4QAM,1-8QAM,2-16QAM,3-32QAM,4-64QAM,5-128QAM,6-256QAM,}
RM_MOD_IQ_MAP,
//@brief : PWM difference of modulation duty cycle
//@remark: {IO:WR}{DATA: 0~PULS DUTY}
RM_MOD_DUTY_DEV,
//}

//{SWEEP
//@brief : Sweep Type
//@remark: {IO:WR}{DATA: 0: Linear, 1: Logarithm}
RM_SWEEP_TYPE = 0x8200,
//@brief : Trigger source of frequency sweep
//@remark: {IO:WR}{DATA: 0: Internal, 1: External, 2: Manual}
RM_SWEEP_SOURCE,
//@brief : Sweep Time (Unit: s)
//@remark: {IO:WR}{DATA: 1ms~500s}
RM_SWEEP_TIME,
//@brief : Sweep starting frequency (Unit: Hz)
//@remark: {IO:WR}{DATA: 1uHz~Maximum carrier wave frequency}
RM_SWEEP_START_FREQ,
//@brief : Sweep ending frequency (Unit: Hz)
//@remark: {IO:WR}{DATA: 1uHz~Maximum carrier wave frequency }
RM_SWEEP_STOP_FREQ,
//@brief : Synchronous output trigger frequency (Unit: Hz)
//@remark: {IO:WR}{DATA:RM_SWEEP_START_FREQ~RM_SWEEP_STOP_FREQ} RM_SWEEP_SYNC_FREQ,
//@brief : Frequency sweep trigger output
//@remark: {IO:WR}{DATA: 0-OFF，1-ON}
RM_SWEEP_TIRG_OUT,
//}

//{BURST
//@brief : Burst Type
//@remark: {IO:WR}{DATA:
//0:   N Cycle,
//1:   Unlimited,
//2:   Gating
//}
RM_BURST_TYPE = 0x8300,
//@brief : Burst trigger source
//@remark: {IO:WR}{DATA: 0: Internal, 1: External, 2: Manual }
RM_BURST_SOURCE,
        //@brief : Trigger Output
        //@remark: {IO:WR}{DATA: 0-OFF，1-ON}
        RM_BURST_TIRG_OUT,
        //@brief : Burst Period (Unit: s)
        //@remark: {IO:WR}{DATA: 1~500}
        RM_BURST_PERIOD,
        //@brief : Burst Phase (Unit: °)
        //@remark: {IO:WR}{DATA:-360~360}
        RM_BURST_PHASE,
        //@brief : Burst Cycles
        //@remark: {IO:WR}{DATA: 1~50000}
        RM_BURST_CYCLES,
        //@brief : Trigger Edge
        //@remark: {IO:WR}{DATA: 0-Rise，1-Fall}
        RM_BURST_TIRG_EDGE
        //}
    }ERemoteMessage;
```