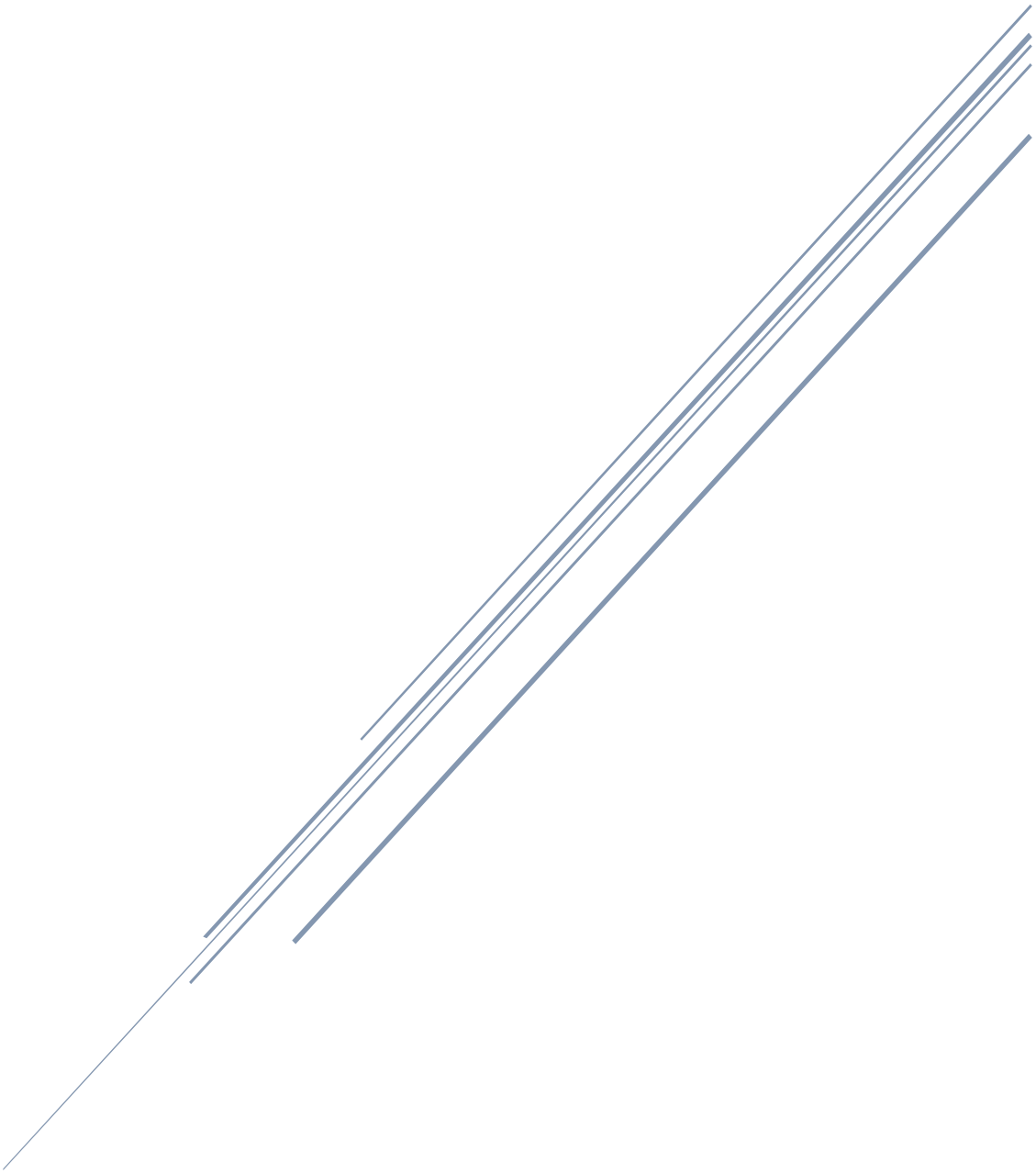
UTG1000A&UTG6000 Series Signal Source

Programming Manual



Uni-trend Technology (China) Co., Ltd.

Version

Version	Modified Item
V1.0	Official version
V1.0.1	Correct prtscn command description

Introduction

This document is the command description of signal source, interface refer to 《UCI Help Document》.
If the command cannot work properly, please contact with technical support.

Reference File

1. UTG4162.h ：
   The basic definition of signal source
2. UCI file：
   See 《UCI Help Document》

The basic format of command string：

Command is communicated with device through character string, the basic format of command string：
{Filed name：command file value...；}
including：
‘：’ character string before colon mark presents command name；
‘：’ character string after colon mark presents filed value；
‘；’ presents the end of a command
The detailed format see《UCI Help Document》

Term： SG - short name of signal source

Common Command

IDN?

Acquire the device name

Data Format ： displayed name % internal information #SN serial number

Data size is 54 bytes.

For example：

UTG1000A%**#SN005

Keypad：

Command Name	Command Parameter	Command Parameter Type
KEY	Key value	See the following key code table

KEY	Character Encoding	KEY	Character Encoding
Bottom function key 1	AF1	0	0
Bottom function key 2	AF2	1	1
Bottom function key 3	AF3	2	2
Bottom function key 4	AF4	3	3
Bottom function key 5	AF5	4	4
Bottom function key 6	AF6	5	5
F1	F1	6	6
F2	F2	7	7
F3	F3	8	8
F4	F4	9	9
Menu	MENU	.	.
Knob Left	FKNL	+/-	SIGN
Knob Right	FKNR	Trigger	TG
Knob Click	FKN	Utility	UTIL
Left	L	CH1	C1
Right	R	CH2	C2

Attribute Name	Meaning	IO	Data
Lock	Lock key	W	No Data
Unlock	Unlock key	W	No Data
Lock?	Query key lock status	R	Integer<4Bytes>: 0 – unlock; 1 –locked
Led?	Query LED status	R	Integer<4Bytes>: 0 – off light; 1 – on light（green）;

For example:

"KEY:c1;" -- CH1

"KEY:c2;" -- CH2

"KEY:c2@lock;" -- CH2 key lock

"KEY:c2@unlock;" -- CH2 unlock key

"KEY:c2@lock?;" -- query whether key is locked

"KEY:c2@led?;" -- query LED status, same as command "LED;"

Notes:

Refer to the description of uci_FormatWrite in 《UCI Help Document》.

Command with question mark need use interface uci_Read to read. It can get it through buffer area or return value by interface.

In addition:

The command for query key lock status:

Command Name	Command Parameter	Data Format
lock?	None	64 bit integer, 0x086FFF FFFFFFFF presents locked full qwerty

Use interface uci_Read get data, call interface IsKeyLocked（see UTG4162.h）to acquire single key status.

Switch Local/Remote Status:

Command Name	Command Parameter	Command Parameter Type
local	Status encoding	Enum(Integer<4Bytes>):0/1{remote status/local status}
Local?	None	Enum(Integer<4Bytes>):0/1{remote status/local status}

For example:

"local:0;" -- remote status, lock full qwerty

"local:1;" -- local status, unlock full qwerty

"local?;" -- query status

Notes:

Refer to the description of uci_FormatWrite in 《UCI Help Document》.

Command with question mark need use interface uci_Read to read. It can get it through buffer area or return value by interface.

Capture Screen:

Command Name	Command Parameter	Command Parameter Type
PrtScn	Image format	Enum(String):null/zip/bmp {pixel data/packed pixel data/BMP file}

For example:

"PrtScn:bmp;" --- screenshot saved as bmp file;

"PrtScn;" --- screenshot saved as pixel data;

"PrtScn:zip;" --- screenshot saved as packed pixel data;

Notes:

Use uci_Read to read data, the command has no saved data file, it should return to the specified buffer area of uci_Read . If you want to buffer a local file, please save it by yourself.

1. If use command: "PrtScn;", buffer area size must be 480 * 272* 4 = 522240,readout is 32bits of pixel data;;
2. If use command: "PrtScn:bmp;", buffer area size must be 480 * 272* 4 + 54 =522294, that is the bitmap;
3. If use command: "PrtScn:zip;", buffer area size can set 480 * 272* 4 = 522240 (maximum data volume), readout is packed pixel data. And then use the interface: alg_UnCompressPixels_1000A to unzipping data. Note: uci_Read return value is packed data volume.
4. Use the interface uci_ReadToFile to add the command "prtscn:bmp;", it can save bitmap to disk file.

Write Arbitrary Wave File:

Command Name	Command Parameter	Command Parameter Type
WARB	none	none

Attribute Name	Meaning	IO	Data
CH	Channel number	W	Enum(Integer) : 0/1{ CH1/ CH2 }
Mode	Loading mode	W	Enum(Integer): 0/1/R {Carrier/Mod}
Disk	memory medium	W	Enum(Integer): 0/1/2/3{RAM/ROM/TF/U-DISK}

For example:

"WARB@CH:0@MODE:0@DISK:0;"

Loading waveform file as the carrier wave into CH1, saved in RAM, it will lost when restart.

Notes:

Use interface uci_WriteFromFile to write arbitrary wave, timeout set to 1000.

Read and Write Configuration File

Command Name	Command Parameter	Command Parameter Type
dconfig	none	none

For example:

"dconfig;"

Notes:

use interface uci_ReadToFile to read configuration file; use interface uci_WriteFromFile to write configuration file

Configuration file suffix is ".set", please make sure the suffix is the same when read and save file.

Version:

Command Name	Command Parameter	Command Parameter Type
Ver?;	none	none

For example:

"Ver?;"

Interface:

Use interface uci_Read to read data.

Data size is 54 bytes.

Communication Protocol Version:

Query version number of system information, to check protocol interface whether support.

Command Name	Command Parameter	Command Parameter Type
CVer?;	none	none

For example:

"CVer?;"

Interface:

Use interface uci_Read to read data.

Data size is 54 bytes.

Data:

```
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    | ..........
```

Read Parameter：

Command Name	Command Parameter	Command Parameter Type
rp	None	None

Attribute Name	Meaning	IO	Data
CH	Channel number	W	Enum(Integer) : 0/1{ CH1/ CH2 }
addr	Parameter address	W	Enum(ERemoteMessage): see the definition in Parameter address

For example：

"rp@CH:0@addr:0x8009;" - read frequency of CH1；

Notes：

UCIInterface corresponding to：uci_Read，the corresponding data size is 8 bytes，double type！

Interface：

Read parameter interface use uci_Read or uci_ReadX；

Write Parameter：

Command Name	Command Parameter	Command Parameter Type
wp	None	None

Attribute Name	Meaning	IO	Data
CH	Channel number	W	Enum(Integer) : 0/1{ CH1/ CH2 }
addr	Parameter address	W	Enum(ERemoteMessage): see the definition in Parameter address
v	Parameter value	W	Physical unit please refer to parameter definition

For example：

"wp@CH:0@addr:0x8009@v:2000;" - set CH1 frequency as 2kHz；

Notes：

UCIInterface corresponding to：uci_Write, data type is double type, 8 bytes.

Interface:

Write parameter interface use uci_Write , uci_WriteX or uci_FormatWrite ;

Query LED Status:

Command Name	Command Parameter	Command Parameter Type
LED	none	none

For example:

"LED;"

Notes:

Use interface uci_Read to read status data, data format is LEDStatus (see the definition in file UTG4162.h）

Sync Reconnection:

Command Name	Command Parameter	Command Parameter Type
Reconnect;	none	none

For example:

"Reconnect;"

Notes:

Use interface *uci_SendCommand* to send command.

In reconnecting, the instrument is break off, send this command to reconnect it.

Check the instrument whether is online, query LED status on timing, that is execute "LED" command, judge the line status by interface return value.

Appendix:

Parameter Definition:

The following definitions comes from file: include\UTG4162.h

explanatory note

For example : {IO:WR}{DATA: 0-OFF, 1-ON, 2-reverse}

IO assignment: read and write attribute of parameter, W presents parameter can be write; R presents parameter can be read.

Data assignment: Data value

```
////////////////////////////////Parameters Address Define////////////////////////////
        //operating mode
        typedef enum _EWorkMode : long long {
            //@brief : fundamental waveform mode
            WM_BASE = 0,
            //@brief : modulating waveform mode
            WM_MODE,
            //@brief : sweep frequency mode
            WM_SWEEP,
            //@brief : burst
            WM_BURST,
        }EWorkMode;

        // fundamental waveformform
        typedef enum _EBaseWave : long long {
            //@brief : sine wave
            BASE_SINE = 0,
            //@brief : square wave
            BASE_SQUARE = 1,
            //@brief : slope 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;

// modulating 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 : parameter encoding of signal source
//@remark: read parameter use command rp, write parameter use command wp.
//@example: wp@ch:0@addr:0x8000@v:0;
//          rp@ch:0@addr:0x8000;
// the corresponding data volume of all parameter are 8 bytes, double type
typedef enum _enumRemoteMessage {
    //@brief : operating mode
    //@remark: {IO:WR}{DATA:EWorkMode}
    RM_WORK_MODE = 0x8000,

    //{ fundamental waveform
    //@brief : channel switch
    //@remark: {IO:WR}{DATA: 0-OFF, 1-ON}
    RM_CH_SW = 0x8001,
    //@brief : sync switch
    //@remark: {IO:WR}{DATA: 0-OFF, 1-ON, 2-reverse}
    RM_CH_SYNC_SW,
    //@brief : channel inversion
```

```
            //@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 enable function
            //@remark: {IO:WR}{DATA:   0：OFF，1：ON}
            RM_CH_OUTPUT_LIMIT_ENABLE,
            //@brief : low level of channel output limit
            //@remark: {IO:WR}{DATA<?>:   -10V~MAX_LEVEL}
            RM_CH_OUTPUT_LIMIT_MIN_LEVEL,
            //@brief : upper level of channel output limit
            //@remark: {IO:WR}{DATA:   MIN_LEVEL~10V}
            RM_CH_OUTPUT_LIMIT_MAX_LEVEL,

            //@brief : fundamental waveform
            //@remark: {IO:WR}{DATA:EBaseWave}
            RM_BASE_WAVE_TYPE = 0x8008,
            //@brief : frequency of fundamental waveform (unit：Hz)
            //@remark: {IO:WR}{DATA:1uHz~the maximum frequency of current fundamental waveform}
            RM_BASE_FREQ,
            //@brief : phase of fundamental waveform (unit：°)
            //@remark: {IO:WR}{DATA:-360~360}
            RM_BASE_PHASE = 0x800A,
            //@brief : amplitude of fundamental waveform (unit：VPP)
            //@remark: {IO:WR}{DATA:1mVpp~10Vpp（50 ohm）}
            RM_BASE_AMP_VPP,
            //@brief : amplitude of fundamental waveform (unit：VRMS)
            //@remark: {IO:WR}{DATA:1mVRMS~5Vpp（50 ohm）}
            RM_BASE_AMP_VRMS,
            //@brief : amplitude of fundamental waveform (unit：VDBM)
            //@remark: {IO:WR}{DATA:-53.010VDBM~26.99VDBM（50 ohm）}
            RM_BASE_AMP_VDBM,
            //@brief : fundamental waveform offset(unit：V)
            //@remark: {IO:WR}{DATA:-5VRMS~5Vpp（50 ohm）}
            RM_BASE_OFFSET,
            //@brief : high level of fundamental waveform (unit：V)
            //@remark: {IO:WR}{DATA:BASE_LOW~5Vpp（50 ohm）}
            RM_BASE_HIGHT = 0x800F,
            //@brief : low level of fundamental waveform (unit：V)
            //@remark: {IO:WR}{DATA:-5VRMS~BASE_HIGHT（50 ohm）}
            RM_BASE_LOW = 0x8010,
            //@brief : fundamental waveform share square duty ratio, pulse duty ratio and
triangular wave symmetry
```

```
            //@remark: {IO:WR}{DATA:0~100}
            RM_BASE_DUTY,
            //@brief : rise time of pulse wave (unit: s)
            //@remark: {IO:WR}{DATA:min ~ cycle*0.4}
            RM_BASE_RISETIME,
            //@brief : fall time of pulse wave (unit: s)
            //@remark: {IO:WR}{DATA:min ~ cycle*0.4}
            RM_BASE_FALLTIME,
            //@brief : switch of arbitrary wave play mode
            //@remark: {IO:WR}{DATA:0: OFF, 1: ON}
            RM_BASE_ARB_PLAY_ENABLE,
            //@brief : harmonic wave - operating mode
            //@remark: {IO:WR}
            //{DATA:
            //0:    ODD,
            //1:    EVEN,
            //2:    ALL,
            //3:    USER, custom
            // }
            RM_BASE_HARMOIC_TYPE = 0x8080,
            //@brief :harmonic switch, only valid in RM_BASE_HARMOIC_TYPE=USER
            //@remark: {IO:WR}{DATA:BIT14 and BIT0 is respectively correspond to 2~16
subharmonic wave switch, BIT15 is correspond to fundamental wave force open }
            RM_BASE_HARMONIC_ONOFF,
            //set N(2~16) subharmonic wave, two methods to set:
            //EG:N=3(third harmonic),AMP = 1Vpp, PHASE = 90°
            //method 1:
            //（1）the specified harmonic order, RM_HARMONIC_NUM = N(2~16),
            //（2）set amplitude and phase of harmonic wave,
            //      RM_HARMONIC_SN_AMP_N = 1.0; RM_HARMONIC_SN_PHASE_N = 90.0;
            //method 2:directloy set amplitude and phase of third harmonic
            //RM_HARMONIC_SN_AMP_3 = 1.0, RM_HARMONIC_SN_PHASE_3 = 90.0,

            //@brief : harmonic order
            //@remark: {IO:WR}{DATA:1~15}
            RM_HARMONIC_NUM,
            //@brief : amplitude of harmonic wave Vpp
            //@remark: {IO:WR}{DATA:0~amplitude of fundamental waveform }
            RM_HARMONIC_SN_AMP_N,
            //@brief : phase of harmonic wave (unit: °)
            //@remark: {IO:WR}{DATA:-360~360}
            RM_HARMONIC_SN_PHASE_N = 0x8084,
            //@brief : amplitude of harmonic wave 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 : phase of harmonic wave 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 : modulating mode
//@remark: {IO:WR}{DATA:EModeType}
RM_MOD_TYPE = 0x8100,
//modulating waveform
//0:  MOD_SINE,
//1:  MOD_SQUARE,
//2:  MOD_UPRAMP,
//3:  MOD_DNRAMP,
//4:  MOD_NOISE,
//5:  MOD_ARB,
//@brief : modulating waveform
```

```
//@remark: {IO:WR}{DATA:EModeWaveType}
RM_MOD_WAVE,
//@brief : frequency of modulating waveform（unit：Hz）
//@remark: {IO:WR}{DATA: 1uHz~200KHz}
RM_MOD_FREQ,
//@brief : modulating waveform rate（unit：s）
//@remark: {IO:WR}{DATA<double>: 2ms~1Ms}
RM_MOD_RATE,
//@brief : modulating depth（unit：%）
//@remark: {IO:WR}{DATA: 0~120}
RM_MOD_SCOPE,
//@brief : modulating source
//@remark: {IO:WR}{DATA: 0-Internal,1-External}
RM_MOD_SOURCE,
//@brief : FM frequency difference（unit：Hz）
//@remark: {IO:WR}{DATA: 0~ the 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~the maximum frequency of carrier wave}
RM_MOD_HOP_FREQ,
//@brief : BPSK modulating data source
//@remark: {IO:WR}
//{DATA<double>:
//0:  PN7,
//1:  PN9,
//2:  PN15,
//3:  PN21,
//}
RM_MOD_DATA_SOURCE = 0x8109,
//@brief : BPSK phase、QPSK phase 1,(unit：°)
//@remark: {IO:WR}{DATA:-360~360}}
RM_MOD_PSK_PHASE1,
//@brief : QPSK phase 2(unit：°)
//@remark: {IO:WR}{DATA:-360~360}}
RM_MOD_PSK_PHASE2,
//@brief : QPSK phase 3(unit：°)
//@remark: {IO:WR}{DATA:-360~360}
RM_MOD_PSK_PHASE3,
//@brief : OSK oscillation time (unit：s)
//@remark: {IO:WR}{DATA: 8ns~200s}
RM_MOD_OSC_TIME,
```

```
//@brief : QAM modulating 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 modulating duty ratio difference value
//@remark: {IO:WR}{DATA: 0~PULS DUTY}
RM_MOD_DUTY_DEV,
//}

//{SWEEP
//@brief : sweep frequency mode
//@remark: {IO:WR}{DATA: 0：linear，1：logarithm}
RM_SWEEP_TYPE = 0x8200,
//@brief : trigger source of sweep frequency
//@remark: {IO:WR}{DATA: 0：internal，1：external，2：manual}
RM_SWEEP_SOURCE,
//@brief : sweep frequency time (unit：s)
//@remark: {IO:WR}{DATA: 1ms~500s}
RM_SWEEP_TIME,
//@brief : initial frequency of sweep frequency（unit：Hz）
//@remark: {IO:WR}{DATA: 1uHz~ the maximum carrier frequency}
RM_SWEEP_START_FREQ,
//@brief : stop frequency of sweep frequency（unit：Hz）
//@remark: {IO:WR}{DATA: 1uHz~ the maximum carrier frequency}
RM_SWEEP_STOP_FREQ,
//@brief : sync output trigger frequency（unit：Hz）
//@remark: {IO:WR}{DATA: RM_SWEEP_START_FREQ~RM_SWEEP_STOP_FREQ}
RM_SWEEP_SYNC_FREQ,
//@brief : trigger output of sweep frequency
//@remark: {IO:WR}{DATA: 0-OFF，1-ON}
RM_SWEEP_TIRG_OUT,
//}

//{BURST
//@brief : burst mode
//@remark: {IO:WR}{DATA:
//0:  N cycle,
//1:  infinity,
//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 cycle（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 : cycle number of burst
    //@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;
```