ADWGC C library

User's Guide

Byte Paradigm – info@byteparadigm.com

Table of Content

Table of Tables

References

[]

History

Version	Date	Description
1.00	01-May-2006	Initial revision (preliminary)
1.01	16-May-2006	First release
1.02	09-Nov-2006	Update for software version 1.04
1.03	05-Mar-2007	Update with trigger rearm functions
1.04	31-May-2007	Update with edge trigger functions
1.05	13-Nov-2007	Update for GP Series introduction
1.06	16-Feb-2010	Review for release 1.08f.
1.07	09-Aug-2010	Added function prefixes and multi-device support
1.08	20-Jan-2012	Added IO voltage selection
1.09	13-Feb-2015	Minor corrections

1 Introduction

The ADWGC C library is a specialised C library used with the GP Series device in ADWG (Arbitrary Digital Waveform Generator) mode of operation. It provides a serie of 'pure C' functions to configure and control the GP Series device from within a C/C++ compatible environment. As opposed to the GP Series device C++ libraries, this library offers a 'pure C' interface with each function, which is often easier to integrate from any external environment.

This library calls itself other libraries and functions to manage the low level transfer of data between the host PC and the GP Series device. Schematically, any session using the ADWGC library starts by connecting itself to the 'Smart Router' application delivered with the 8PI Control Panel. This application manages the different client connections to the GP Series device and handles priorities between the processes and applications. On the other side, it is responsible for the actual data transfers onto the USB connection.

For advanced information and support, please submit your requests to: support@byteparadigm.com.

2 GP Series device ADWGC C Library

2.1 Functions quick Reference Table

Table 1 gives a list of the functions callable from the ADWG, in the same order as in the header file: ADWGC.h.

Table 1: Quick reference table of ADWG procedures (in order of appearance)

Function prototype	Description
`int adwg_ADWGC(void)`	Initialises the ADWGC library session and creates a first instance of the ADWGC library
`int adwg_CreateInstance(void)`	Creates and additional instance of the library.
`void adwg_SelectInstance(int Handle)`	Selects a library instance.
`int adwg_SelectDevice(char *pSerNum)`	Selects a device using the serial number.
`void adwg_SelectIOVoltage(int IOVoltage)`	Selects the user interface voltage.
`short adwg_Ver(void);`	Returns the software version.
`bool adwg_IsDeviceReady(void);`	Checks if the device is ready to operate.
`bool adwg_IsRunning(void);`	Checks if the device is busy and running.
`void adwg_Terminate(void);`	Closes an instance of the ADWGC library. A call to this function is mandatory for each instance of the library before closing the application.
`void adwg_SetReqClock(int Freq);`	Defines the operating clock frequency (in Hz).
`int adwg_GetReqClock(void);`	Returns the requested operating clock frequency (Hz).
`int adwg_GetSynthClock(void);`	Returns the synthesised operating clock frequency (Hz). It can differ from the requested frequency due to the limited accuracy of the frequency divider.
`int adwg_GetSynthOutClock(void);`	Returns the achieved frequency (Hz) of the output clock. This value depends on the operating clock frequency and the output clock ratio.
`void adwg_SetOutputClock(bool Output);`	Enables or disables the generation of the output clock
`bool adwg_GetOutputClock(void);`	Gets the status of the output clock generation (enabled or disabled).
`void adwg_UseIntClock(void);`	Configures the device to operate with the internal operating clock as reference.
`void adwg_UseExtClock(void);`	Configures the device to operate with an externally supplied operating clock as reference.
`bool adwg_GetClockSource(void);`	Returns the selected clock source.
`void adwg_SetClockContinuity(bool Hole);`	Configures the current continuity mode used to generate the output clock (hole or continuous).
`bool adwg_GetClockContinuity(void);`	Returns the current continuity mode used to generate the output clock (hole or continuous).
`void adwg_SetClockEdge(bool Pos);`	Defines the phase relation between the output clock and the device internal clock.
`bool adwg_GetClockEdge(void);`	Returns the phase relation between the device internal clock and the generated output clock.
`void adwg_SetOutClockRatio(int Ratio);`	Defines the integer ratio between the operating clock frequency and the output clock frequency.
`int adwg_GetOutClockRatio(void);`	Returns the current value of the output clock ratio.
`void adwg_SetClockDisablePLL(bool Disable);`	Forces disabling the PLL or let the device enable it automatically when possible
`bool adwg_GetClockDisablePLL(void);`	Get the operating mode of the PLL (disabled / automatic).

Function prototype	Description
`void adwg_SetInternalTrigger(bool Internal);`	Defines the triggering mode (internal or external).
`bool adwg_GetInternalTrigger(void);`	Returns the current triggering mode.
`void adwg_SetEdgeTrigger(bool Enable)`	Selects the triggering mode (edge or level)
`bool adwg_GetEdgeTrigger(void)`	Returns the triggering mode (edge or level)
`void adwg_SetCtrlTrigMask(short *pMask);`	Defines the triggering mask to select the control lines to use as trigger inputs.
`void adwg_GetCtrlTrigMask(` `  short *pCtrlTrigMask);`	Returns the triggering mask.
`void adwg_SetCtrlTrigPattern(` `  short *pPattern);`	Defines the pattern to detect on the trigger inputs to generate the trigger event.
`void adwg_GetCtrlTrigPattern(` `  short *pTrigPattern);`	Returns the triggering pattern.
`int adwg_SetAutoReArm(bool AutoReArm)`	Enables the trigger auto rearm feature.
`bool adwg_GetAutoReArm(void)`	Returns the trigger auto rearm feature status.
`void adwg_EnCtrlSeq(void);`	Enables the generation of static or periodic sequence on the control lines.
`void adwg_DisCtrlSeq(void);`	Disables the generation of static and periodic sequence on the control lines.
`bool adwg_GetCtrlSeqEn(void);`	Returns the current status for the control sequence generation.
`int  adwg_GetCtrlSeqLength(void);`	Returns the current length of the periodic sequence which will be applied on the control lines.
`void adwg_EnDefaultCtrl(void);`	Enables applying default static levels on the control lines.
`void adwg_DisDefaultCtrl(void);`	Disables applying a default level on the control lines.
`bool adwg_GetDefaultCtrlEn(void);`	Returns the status of the default level feature.
`void adwg_SetCtrlMaskOut(short *pMaskOut);`	The control mask selects the control lines on which the default static levels or the periodic pattern have to be applied.
`void adwg_GetCtrlMaskOut(short *pMaskOut);`	Returns the control mask value.
`void adwg_SetCtrlDefaultVal(` `  short *pDefaultVal);`	Defines the default level to apply on the control lines.
`void adwg_GetCtrlDefaultVal(` `  short *pCtrlDefaultVal);`	Returns the current default level applied on the control lines.
`void adwg_SetDataMaskOut(short *pMaskOut);`	Selects the data lines enabled as outputs.
`void adwg_GetDataMaskOut(short *pMaskOut);`	Returns the data mask value.
`void adwg_SetStaticData(short Data);`	Applies the data pattern to the device system connector pins.
`unsigned short adwg_GetStaticData(void);`	Returns last static data value sent to the device.
`int adwg_GetNrSamples(void);`	Returns the total number of data samples loaded in memory.

`int adwg_AdwgRun(`
`  int  SendSamples,`
`  bool StepStart,`
`  bool Message);`
	Starts transferring data to the GP Series device.

`int adwg_AdwgRunH(`
`  int  SendSamples,`
`  bool StepStart,`
`  bool Message,`
`  int  Handle);`
	Starts transferring data to the GP Series device for a specific library instance.

Function prototype	Description

`int adwg_AdwgRunLoop(`
`  int SendSamples,`
`  bool StepStart,`
`  bool Message);`
	Starts transferring data to the GP Series device. This function must be used with the loop features.

`int adwg_AdwgRunLoopH(`
`  int SendSamples,`
`  bool StepStart,`
`  bool Message,`
`  int  Handle);`
	Starts transferring data to the GP Series device for a specific library instance. This function must be used with the loop features.

`void adwg_Abort(void)`	Aborts the ADWG running in loop / trigger auto rearm modes.
`void adwg_SetInfiniteLoop(bool Enable);`	Enables / disables the infinite loop mode.
`bool adwg_GetInfiniteLoop(void);`	Returns the enabled/disabled status of the infinite loop mode.
`int adwg_MakeConfFileTemplate(` `  char *FileName);`	Creates a configuration file template.

`int adwg_ReadConfAndDataFile(`
`  char *FileName,`
`  bool Message);`
	Reads the device controls, configuration, data header and the data samples to transfer from a file.

`int adwg_ReadDataFile(`
`  char *FileName,`
`  bool Message);`
	Reads the data header configuration and a set of data to be sent from the specified file.

`int adwg_ReadConfFileADWG(`
`  char *FileName ,`
`  bool Message);`
	Reads the device controls and configuration from a file.

`int adwg_WriteConfAndDataFile(` `  char *FileName);`	Writes the device configuration, programmed control sequence and data samples to a formatted file.
`int adwg_WriteConfFile(char *FileName);`	Writes the device configuration and the programmed control sequence to a formatted file.
`void adwg_ClearBuffer(void);`	Clears the data buffer loaded in memory.
`int adwg_WriteSingleSample(short Data);`	Appends the memory data buffer with the data sample specified as argument.

`int adwg_WriteMultiSample(`
`  void *pData,`
`  int Length);`
	Appends the memory data buffer with multiple samples.

`int adwg_GetLastErr(void);`	Returns the last error detected during the data transfer.
`void adwg_ResetCfg(void);`	Reset the device to its default configuration.
`void adwg_Unsupervised(bool Enable);`	Enables or disables the unsupervised operating mode.
`void adwg_SetAdwgContMode(bool Continuous);`	Enables or disables the continuous operating mode.
`bool adwg_GetAdwgContMode(void);`	Returns the status of the continuous operating mode.
`void adwg_UpdateAdwgContMode(void);`	Downloads the mode configuration to the GP Series device.

2.2 Functions details

This section gives a detailed description of each function available to control the GP Series device ADWG operating mode with the ADWGC library. The functions are listed in alphabetical order.

`int adwg_ADWGC (void)`

parameters: none
returns: A handle to the initialised library instance.
description: Initialises an ADWG session and creates a first instance of the ADWGC library. This function must be called at the start of any session

using the ADWGC library. It enables the control of the device by 'connecting' the C session as a client to the 'Smart Router'.

int adwg_AdwgRun (int SendSamples, bool StepStart, bool Message)

parameters: SendSamples: integer value that specifies the number of samples to transfer for the device.
StepStart: must be set to 0. Used to support advanced functions – please contact support@byteparadigm.com for more information.
Message: boolean enabling or disabling the message pop-ups during run.

returns: An error code integer. ≥0 if successful.

description: Starts transferring data to the GP Series device. The specified number of samples is applied onto the device system connector. Only the pins enabled and configured as output are driven; the other ones remain in high impedance state.

conditions: If the device is not configured to operate in ADWG file mode, this function automatically configures it with the current defined settings.

int adwg_AdwgRunH (int SendSamples, bool StepStart, bool Message, int Handle)

parameters: SendSamples: integer value that specifies the number of samples to transfer for the device.
StepStart: must be set to 0. Used to support advanced functions – please contact support@byteparadigm.com for more information.
Message: boolean enabling or disabling the message pop-ups during run.
Handle: handle of a library instance

returns: An error code integer. ≥0 if successful.

description: Starts transferring data to the GP Series device using the device linked to the specified handle. The specified number of samples is applied onto the device system connector. Only the pins enabled and configured as output are driven; the other ones remain in high impedance state.

conditions: If the device is not configured to operate in ADWG file mode, this function automatically configures it with the current defined settings.

int adwg_AdwgRunLoop (int SendSamples, bool StepStart, bool Message)

parameters: SendSamples: integer value that specifies the number of samples to transfer for the device.
StepStart: must be set to 0. Used to support advanced functions – please contact support@byteparadigm.com for more information.
Message: boolean enabling or disabling the message pop-ups during run.

returns: An error code integer. ≥0 if successful.

description: Starts transferring data to the GP Series device. The specified number of samples is applied onto the device system connector. Only the pins enabled and configured as output are driven; the other ones remain in high impedance state. This function must be used to start runs with the infinite loop and the trigger auto rearm features.

conditions:

int adwg_AdwgRunLoopH (int SendSamples, bool StepStart, bool Message, int Handle)

parameters: SendSamples: integer value that specifies the number of samples to transfer for the device.

StepStart: must be set to 0. Used to support advanced functions – please contact support@byteparadigm.com for more information.
Message: boolean enabling or disabling the message pop-ups during run.
Handle: handle of a library instance

returns: An error code integer. ≥0 if successful.

description: Starts transferring data to the GP Series device using the device linked to the specified handle. The specified number of samples is applied onto the device system connector. Only the pins enabled and configured as output are driven; the other ones remain in high impedance state. This function must be used to start runs with the infinite loop and the trigger auto rearm features.

conditions:

`void adwg_Abort (void)`

parameters: none.

returns:

description: Aborts the ADWG running in loop / trigger auto rearm modes.

conditions:

`void adwg_ClearBuffer (void)`

parameters: none.

returns:

description: Clears the data buffer loaded in memory. Before being sent to the device, the data samples are stored in a memory area. This function allows to completely clear the area of the memory where the samples are buffered.

see also: WriteSingleSample, WriteMultiSample.

`int adwg_CreateInstance(void)`

parameters: none

returns: A handle to the created library instance.

description: Creates an additional instance of the library. This in needed when using more than one device at the same time. Each instance of the library can be linked to a different device with the adwg_SelectDevice function.

`void adwg_DisCtrlSeq (void)`

parameters: none.

returns:

description: Disables the generation of the static or periodic sequences on the control lines.

see also: EnCtrlSeq, EnDefaultCtrl, DisDefaultCtrl.

`void adwg_DisDefaultCtrl (void)`

parameters: none.

returns:

description: Disables the application of a forced default level on the control lines.

see also: EnDefaultCtrl, EnCtrlSeq, DisCtrlSeq.

`void adwg_EnCtrlSeq (void)`

parameters: none.

returns:
description: Enables the generation of the static or periodic sequences on the control lines.
see also: DisCtrlSeq, EnDefaultCtrl, DisDefaultCtrl.

void adwg_EnDefaultCtrl (void)
parameters: none.
returns:
description: Enables the application of default static levels on the control lines. When this feature is enabled, static levels can be set in the selected control lines. Valid for File or Static modes.
conditions: To enable the default level feature for the control lines, the control sequence feature has to be enabled.
see also: DisDefaultCtrl, EnCtrlSeq, DisCtrlSeq.

bool adwg_GetAdwgContMode (void)
parameters: none.
returns: 0 if the continuous mode is programmed.
1 if the continuous mode is programmed.
description: Returns the status of the 'continuous' operating mode.
conditions: The continuous mode is only defined for the File mode.
see also: SetAdwgContMode, UpdateAdwgContMode, SetAdwgMode.

bool adwg_GetAutoReArm (void)
parameters: none.
returns: 0 if the trigger auto rearm feature is disabled.
1 if the trigger auto rearm feature is enabled.
description: Returns the status of the 'trigger auto rearm' feature.
conditions: Trigger auto rearm only works with an external trigger.
see also: SetAutoReArm, AdwgRunLoop.

bool adwg_GetClockContinuity (void)
parameters: none.
returns: 0 if the output clock is continuous.
1 if the output clock is generated with 'holes'.
description: Returns the 'continuity mode' used to generate the output clock.
see also: SetClockContinuity.

bool adwg_GetClockDisablePLL (void)
parameters: none.
returns: 1 when the device internal PLL is disabled.
0 when the PLL automatic control mode is enabled.
description: Returns the control mode of the device internal PLL.
see also: SetClockDisablePLL.

bool adwg_GetClockEdge (void)
parameters: none.
returns: 0 Output data lines are generated in phase with the falling edge of the output clock.
1 Output data lines are generated in phase with the rising edge of the output clock.
description: Returns the phase relation between the device internal clock and the generated output clock.

see also: SetClockEdge.

bool adwg_GetClockSource (void)
parameters: none.
returns: 1 if the internal reference clock is used.
0 if an externally supplied reference clock is used.
description: Returns the reference clock source used to generate the device operating clock.
see also: UseIntClock , UseExtClock.

void adwg_GetCtrlDefaultVal (short *pDefaultVal)
parameters: *pDefaultVal: pointer to a short value. This value represents a 6 bits value and ranges from 0 to 63.
returns:
description: Returns the current default level pattern defined for the 6 control lines. The value is returned through the *pDefaultVal pointer.
see also: SetCtrlDefaultVal

void adwg_GetCtrlMaskOut (short *pMaskOut)
parameters: *pMaskOut: pointer to a short value. This value represents a 6 bits mask and ranges from 0 to 63.
returns:
description: Returns a mask value representing the control lines enabled as output. The value is returned through the *pMaskOut pointer.
see also: SetCtrlMaskOut

bool adwg_GetCtrlSeqEn (void)
parameters: none.
returns: 1 when the control sequence feature is enabled.
0 when the control sequence feature is disabled.
description: Returns the status of the 'enable control sequence' feature.
see also: DisCtrlSeq, EnCtrlSeq.

int adwg_GetCtrlSeqLength (void)
parameters: none.
returns: An integer value.
description: Returns the current length of the defined control sequence.
see also: ReadConfAndDataFile.

void adwg_GetCtrlTrigMask (short *pCtrlTrigMask)
parameters: *pCtrlTrigMask: pointer to a short value. This value represents a 6 bits mask and ranges from 0 to 63.
returns:
description: Returns the mask applied on the control lines to detect the external trigger. The value is returned through the *pCtrlTrigMask pointer.
see also: SetCtrlTrigMask

void adwg_GetCtrlTrigPattern (short *pCtrlTrigPattern)
parameters: *pCtrlTrigPattern: pointer to a short value. This value represents a 6 bits pattern and ranges from 0 to 63.

returns:
description: Returns the pattern applied on the control lines to generate a trigger event and start applying data samples on the GP Series device system connector. The value is returned through the *pCtrlTrigPattern pointer.
see also: SetCtrlTrigPattern.

void adwg_GetDataMaskOut (short *pMaskOut)
parameters: *MaskOut: pointer to a short value. This value represents a 16 bits mask and ranges from 0 to 65535.
returns:
description: Returns a msk value representing the data lines enabld as output. The value is returned through the *pCtrlTrigPattern pointer.
see also: SetDataMaskOut.

bool adwg_GetDefaultCtrlEn (void)
parameters: none.
returns: 1 when the default level feature is enabled.
0 when the default level feature is disabled.
description: Returns the status of the 'deafult level' feature for the control lines.
see also: EnDefaultCtrl, EnCtrlSeq.

bool adwg_GetEdgeTrigger (void)
parameters: none.
returns:
description: Returns the current triggering mode: 1 for 'edge triggering mode'; 0 for 'level triggering mode'.

see also:

bool adwg_GetInfiniteLoop (void)
parameters: none.
returns: 1 when the 'infinite loop mode' is enabled.
0 when the 'infinite loop mode' is disabled.
description: Returns the enable/disable status of the 'infinite loop mode'.
see also: SetInfiniteLoop.

bool adwg_GetInternalTrigger (void)
parameters: none.
returns: 1 when the internal triggering mode is enabled.
0 when the external triggering mode is enabled.
description: Returns the triggering mode in use.
see also: SetInternalTrigger.

int adwg_GetLastErr (void)
parameters: none.
returns: An integer value.
description: Returns the last error code returned by the device.

see also: AdwgRun.

```
int adwg_GetNrSamples (void)
```
parameters: none.
returns: An integer value.
description: Returns the total number of data samples loaded in memory.
see also: ReadConfAndDataFile.

```
int adwg_GetOutClockRatio (void)
```
parameters: none.
returns: An integer value between 1 and 65535.
description: Returns the integer ratio between the output clock frequency and the operating clock frequency.
see also: SetOutClockRatio.

```
int adwg_GetOutputClock (void)
```
parameters: none.
returns: 1 when the output clock is enabled.
0 when the output clock is disabled.
description: Returns the status of the output clock generation.
see also: SetOutputClock.

```
int adwg_GetReqClock (void)
```
parameters: none.
returns: An integer value representing a frequency in Hz.
description: Returns the frequency requested with the function 'SetReqClock', for the device operating clock frequency.
see also: void SetReqClock(int Freq), int GetSynthClock(void).

```
unsigned short adwg_GetStaticData (void)
```
parameters: none.
returns: An unsigned short value ranging from 0 to 65535.
description: Returns the last static data transferred to the device.
see also: SetStaticData.

```
int adwg_GetSynthClock (void)
```
parameters: none.
returns: An integer value representing a frequency in Hz.
description: Returns the achieved frequency for the system operating clock. Since the operating clock is generated using an integer ratio to divide a reference clock, not all frequencies can be generated. This function returns the achieved frequency value. This value can differ from the requested clock frequency. If the requested frequency can not be achieved, the device uses the first achievable frequency smaller than the requested one.
see also: void SetReqClock(int Freq), int GetSynthClock(void).

```
int adwg_GetSynthOutClock (void)
```
parameters: none.
returns: An integer value representing a frequency in Hz.
description: Returns the achieved frequency of the output clock.
see also: void SetOutClockRatio(int Ratio)

```
bool adwg_IsDeviceReady (void)
```
parameters: none

returns: 1 if device is connected
description: Checks if the device is properly connected to the host PC.

bool adwg_IsRunning (void)
parameters: none
returns: 1 if device is running
description: Checks if the device is busy transmitting data.

int adwg_MakeConfFileTemplate (char *FileName)
parameters: FileName: specifies the function output file name (including path).
returns: 0 when file generation is complete.
description: Creates a standard template to be used as configuration file with the device.

int adwg_ReadConfAndDataFile (char *FileName, bool Message)
parameters: FileName: specifies the function input file name (including path).
Message: when set to 1, enables the pop-up windows for information and error messages.
returns: 0 if read successful; another value if read failed.
description: Reads the device configuration, control sequence configuration, data header and a set of data to be sent from the specified file.
see also: ReadConfFileADWG, ReadDataFile.

int adwg_ReadConfFileADWG (char *FileName, bool Message)
parameters: FileName: specifies the function input file name (including path).
Message: when set to 1, enables the pop-up windows for information and error messages.
returns: 0 if read successful; another value if read failed.
description: Reads the device configuration and the control sequence configuration from the specified file. Skips the data header and any set of data defined in the input file.
see also: ReadConfAndDataFile, ReadDataFile.

int adwg_ReadDataFile (char *FileName, bool Message)
parameters: FileName: specifies the function input file name (including path).
Message: when set to 1, enables the pop-up windows for information and error messages.
returns: 0 if read successful; another value if read failed.
description: Reads the data header configuration and a set of data to be sent from the specified file. Skips the device and the control sequence configurations.
see also: ReadConfAndDataFile, ReadConfFileADWG

void adwg_ResetCfg (void)
parameters: none.
returns:
description: Resets the device to a default state.

int adwg_SelectDevice(char *pSerNum)
parameters: pSerNum: ascii encoded string containing an 11 character serial number
returns: -1 when the selection fails, a positive of zero value is returned on success

description: Selects the device based on its serial number. The selected device is associated with the currently selected library instance.

`void adwg_SelectIOVoltage(int *IOVoltage)`

parameters: IOVoltage: integer value representing the IO voltage, the IO voltage can be internally generated of user applied. The voltage level is defined in millivolts.
returns:
description: This function only takes the following predefined values: 3300, 2500, 1800, 1500 and 1200. The nearest value must be selected when the user applies a different external voltage level. For example, set IOVoltage to 2500 when 2.7V is applied.
The default value is 3300.

`void adwg_SelectInstance(int Handle)`

parameters: Handle: Handle to a instance of a previously create library instance.
returns:
description: Selects the library instance corresponding to the supplied handle. The selected instance must first be created with adwg_CreateInstance.

`void adwg_SetAdwgContMode (bool Continuous)`

parameters: Continuous: boolean value (0 or 1).
returns:
description: Enables (Continuous = 1) or disables (Continuous = 0) the continuous operating mode. In continuous mode, the amount of data specified with the AdwgRun function parameters should be transferred as a single continuous run. If the device fails to do so, an underrun condition is reported. In non-continuous mode, pause times are allowed inside the data run to provide enough data to the device.
conditions: The continuous mode is only defined for the File mode. SetAdwgContMode prepares the 'continuous mode setting'. To definitely set the GP Series device in the mode programmed with this function, the function 'UpdateAdwgContMode' must be called.
see also: UpdateAdwgContMode, GetAdwgContMode.

`int adwg_SetAutoReArm (bool AutoReArm)`

parameters: AutoReArm: boolean value (0 or 1).
returns: An integer error code ≥ 0 when successful.
description: Enables (AutoReArm = 1) or disables (AutoReArm = 0) the trigger 'auto rearm' feature. When an external trigger is selected, the 'auto rearm' feature allows to automatically send the same ADWG run, loaded into the GP Series device embedded memory, upon the arrival of a new external trigger.
Two operating modes exist:

- GP Series device driven: the pattern fits un the 16Mbytes memory of the GP-20050, as such it is locally stored in the device and independent of the USB bandwidth
- Software driver: the pattern is too large to fit in the GP Series device memory, in this case it is streamed from the host computer and the throughput is dependant of the USB bandwidth

Not that the selection between both operating modes is done automatically based on the size of the pattern.

conditions: The trigger 'auto rearm' feature is only available when an external trigger is used.
see also: GetAutoReArm, AdwgRunLoop.

void adwg_SetClockContinuity (bool Hole)
parameters: Hole: boolean value (0 or 1).
returns:
description: Sets the 'continuity mode' of the output clock. When the parameter 'Hole' is set to 0, then the output clock is continuous. When the parameter 'Hole' is set to 1, the output clock is generated only when output data are available.
see also: GetClockContinuity

void adwg_SetClockDisablePLL (bool Disable)
parameters: Disable: boolean value (0 or 1).
returns:
description: Forces the disabling of the device internal PLL used for the connector clock generation. When the Disable parameter is set to 0, the PLL will be automatically enabled, if possible. When Disable is set to 1, the PLL is not used.
see also: GetClockDisablePLL

void adwg_SetClockEdge (bool Pos)
parameters: Pos: boolean value (0 or 1).
returns:
description: Defines the phase relation between the output clock and the device internal clock. When the clock edge 'Pos' parameter is set to 1, the output clock is in phase with the device internal clock; when this parameter is set to 0, there is a 180° phase shift between the clocks.
see also: GetClockContinuity

void adwg_SetCtrlDefaultVal (short *pDefaultVal)
parameters: *pDefaultVal: pointer to a short value. This value represents a 6 bit value and ranges from 0 to 63.
returns:
description: The static default value is applied on the control lines with the first data sent out to the device. The levels remain constant until a new control pattern is defined. If a new default value is defined, it will only be applied on the control lines with the next outgoing data sample. The default value is given as a pointer to a short value equivalent to the binary value of the default value (example: default value = (Binary)011001 ➔ *pDefaultVal points to a short = 25).
see also: GetCtrlMaskOut

void adwg_SetCtrlMaskOut (short *pMaskOut)
parameters: *pMaskOut: pointer to a short value. This value represents a 6 bit mask and ranges from 0 to 63.
returns:
description: The control mask selects the device control lines on which the default static levels or the periodic patterns have to be applied. When a mask bit is set to 0, the corresponding control line is masked. The mask is given as a pointer to a short value equivalent to the binary value of the mask

(example: mask = (Binary)011001 ➔ *pMaskOut points to a short = 25).

see also: GetCtrlMaskOut

void adwg_SetCtrlTrigMask (short *pMask)

parameters: *pMask: pointer to a short value. This value represents a 6 bit mask and ranges from 0 to 63.

returns:

description: When the external triggering mode is used, the trigger mask selects the control lines to be used as trigger inputs. When a mask bit is set to 0, the corresponding control line is masked for triggering. The mask is given as a pointer to a short value equivalent to the binary value of the mask (example: mask = (Binary)011001 ➔ *pMask points to a short = 25).

see also: GetCtrlTrigMask

void adwg_SetCtrlTrigPattern(short *pPattern)

parameters: *pPattern: pointer to a short value. This value represents a 6 bit pattern and ranges from 0 to 63.

returns:

description: Defines the pattern to detect on the trigger inputs to generate the trigger event. The pattern is given as a pointer to a short value equivalent to the binary value of the pattern (example: pattern = (Binary)011001 ➔ *pPattern points to a short = 25).

see also: GetCtrlTrigPattern

void adwg_SetDataMaskOut (short *pMaskOut)

parameters: *pMaskOut: pointer to a short value. This value represents a 16 bit mask and ranges from 0 to 65535.

returns:

description: Applies a mask onto the data output lines to enable / disable them. The mask is given as a pointer to a short value equivalent to the binary value of the mask (example: mask = (Binary)0110111110111101 ➔ *pMaskOut points to a short = 28605).

see also: GetDataMaskOut

void adwg_SetEdgeTrigger (bool Enable)

parameters: Enable: Valid values: 0 or 1 – 0 for 'level trigger', 1 for 'edge trigger'

returns:

description: Selects the triggering mode: in edge triggering mode (Enable = 1), the ADWG run will be triggered upon detection of a transition to the programmed trigger pattern. In level triggering mode (Enable = 0), the ADWG run is triggered upon detection of the programmed trigger pattern.

see also:

void adwg_SetInfiniteLoop (bool Enable)

parameters: Enable: boolean value (0 or 1).

returns:

description: Enables / disables the infinite loop mode. In this mode, the device infinitely loops over the set of loaded data.

conditions: The amount of data to transmit must not exceed the device embedded memory.

void adwg_SetInternalTrigger (bool Internal)

parameters: Internal: boolean value (0 or 1).
returns:
description: Defines the triggering mode. When Internal is set to 1, the internal triggering mode is selected; when Internal is set to 0, the external triggering mode is selected. When internal mode is selected, the device does not wait for any external event to start the data run.
conditions:

void adwg_SetOutClockRatio (int Ratio)

parameters: Ratio: integer value representing the ratio between the output clock frequency and the operating clock frequency. Valid range: 1 to 65535.
returns:
description: Defines the output clock ratio with respects to the operating clock: Frequency(Output Clock) = Frequency(Operating Clock)/Ratio.

void adwg_SetOutputClock (bool Output)

parameters: Output: boolean value (0 or 1).
returns:
description: Enables or disables the generation of an output clock. When Output is set to 1, the output clock is enabled; when Output is set to 0, the output clock is disabled.

void adwg_SetReqClock (int Freq)

parameters: Freq: integer value representing the requested frequency in Hz. Range: from 800 (800 Hz) to 100000000 (100MHz) depending on the used device's specifications.
returns:
description: Defines the requested operating clock frequency. The device will actually set the operating clock frequency to the closest frequency available.
see also: int GetReqClock(void), int GetSynthClock(void).

void adwg_SetStaticData (short Data)

parameters: Data: short value equivalent to a 16 bits data. Range: 0 to 63535.
returns:
description: Applies the data pattern specified with the parameter Data to the device system connector pins.

void adwg_Terminate (void)

parameters: none
returns:
description: Closes an instance of the ADWGC library and closes the communication with the device. A call to this function is mandatory for each instance of the library before closing the application.

void adwg_Unsupervised (bool Enable)

parameters: Enable: boolean – valid values: 0 or 1.

returns:
description: Enables or disables the unsupervised operating mode. In this mode, the traffic part reserved to the control between the host and the device is reduced to a minimum. As a consequence, more bandwidth is available for the data on the USB connection.

`void adwg_UpdateAdwgContMode (void)`
parameters: none
returns:
description: Actually downloads the 'continuous mode' setting prepared with function SetContMode to the GP Series device.
see also: SetContMode.

`void adwg_UseExtClock (void)`
parameters: none
returns:
description: Configures the device to operate with an externally supplied operating clock as reference.

`void adwg_UseIntClock (void)`
parameters: none
returns:
description: Configures the device to operate with the internal operating clock as reference.

`short adwg_Ver (void)`
parameters: none
returns: A short integer value representing the software version.
description: Returns the software version as a decimal value. Example: software version 1.04 ➔ returns 0x104 converted as a decimal value: 260.

`int adwg_WriteConfAndDataFile (char *FileName)`
parameters: *FileName: specifies the function output file name (including path).
returns: 0 if write successful; another value if write failed.
description: Writes the device configuration, the programmed control sequence and the data samples to a formatted file.

`int adwg_WriteConfFile (char *FileName)`
parameters: *FileName: specifies the function output file name (including path).
returns: 0 if write successful; another value if write failed.
description: Writes the device configuration to a file.

`int adwg_WriteMultiSample (void *pData, int Length)`
parameters: *pData: pointer to a memory area where a chained set of sample is stored.
returns: Length: specifies the length in bytes of the chained list pData points to
description: Appends the memory data buffer with the data samples found in the location pointed to by *pData. The amount of data is specified by the Length parameter.
see also: WriteSingleSample, ClearBuffer.

int adwg_WriteSingleSample (short Data)

parameters:	Data: specifies the sample to be added to the memory data buffer
returns:	0 if write successful; another value if write failed.
description:	Appends the memory data buffer with the data sample specified as argument. With multiple calls to this function, it is possible to create a set of samples to be transmitted with the device without having to use a data file, potentially offering a fast way to load data in memory.
see also:	WriteMultiSample, ClearBuffer.