

Welcome

Please use the menu on the left to navigate this manual. If you are unsure about terms in this manual, please refer to the AWG GUI Manual. If you come across any difficulties please feel free to contact us. Our contact information can be found in the AWG GUI Manual.

To get a good start on writing your program using our API, please read the Getting Started section to get a general overview of the process. You may then glance over the methods and properties to see which ones would be useful for you. The Helpful Tips section will give you some advice on writing your code. Finally, we have offered several code examples to guide you in writing your own programs. If you are stuck on writing your program, do not hesitate to contact us.

Getting Started -

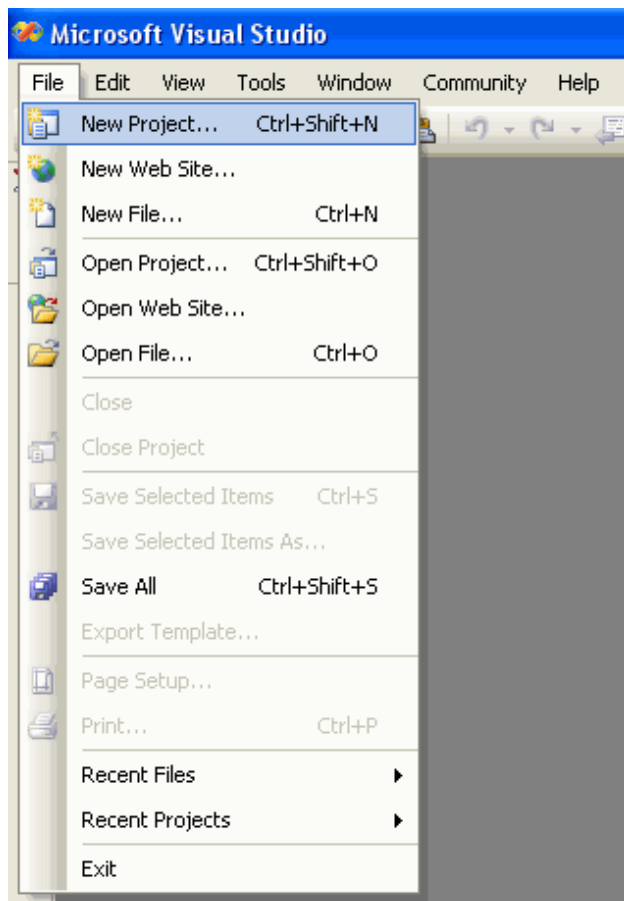
This section will give you a general overview of the program writing process. The API Reference page will show you the necessary steps needed to include the API in your program. The Intermediate Class page gives examples that will help you in writing your own intermediate classes. The Basic Structure page will show you the general order of commands that every waveform should adhere to. Please use the menu on the left to navigate to the desired page.

API Reference -

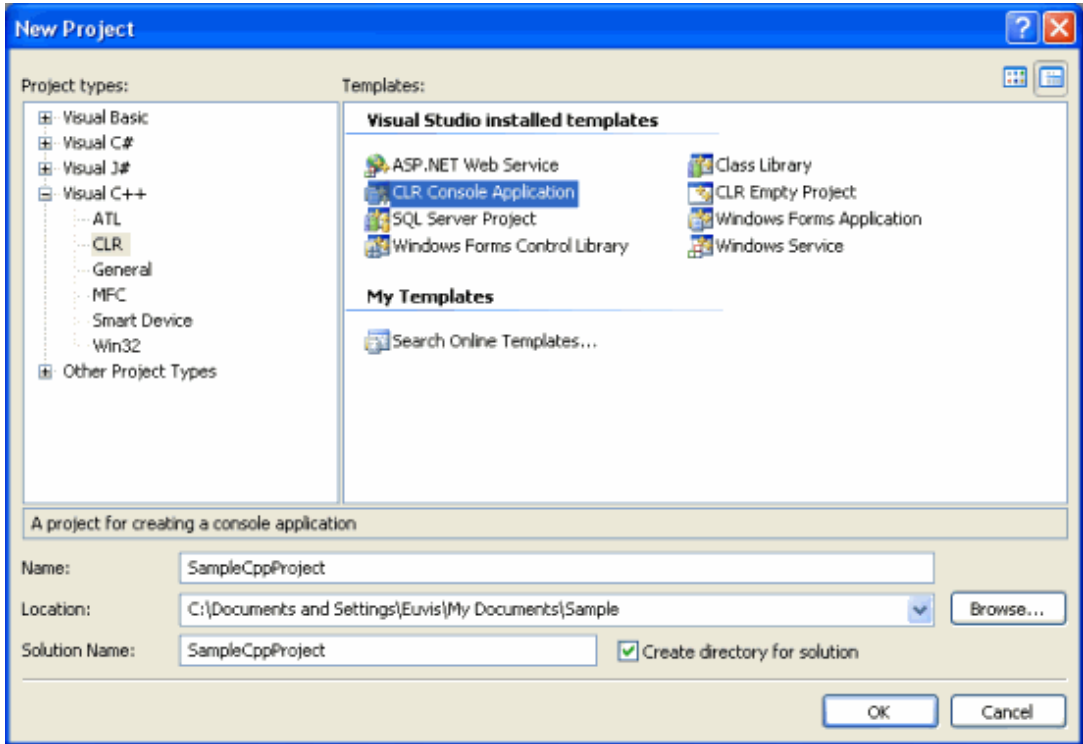
The API file, **Euvis_Module_V1p0.dll** is implemented with the **CLR** (Common Language Runtime) support. Languages targeting the runtime, such as C++/CLI, C#, Visual Basic, Jscript, J# and IL (Intermediate Language) assembler, can be used to reference the AWG API. In this manual, we use C++/CLI (with CLR-support) to demonstrate examples. Other languages can be applied in similar ways.

There is no need for header files or declarations of the API in CLR-supported code. The API can be referenced by adding the `Euvis_Module_V1p0.dll` to References (see below). All the available members, properties, and methods can be identified by using the object browser. In order to use the AWG API, the users' applications need to use the CLR-support option in compiling the codes which use the AWG API. It's not required to recompile all the code, but only the code using the API needs to be compiled with the CLR-support option. To compile the existing code with CLR-support, simply open the **Property** of the concerning source codes and select **Common Language Runtime Support** under **C/C++:General:Compile with Common Language Runtime Support**. If you are just beginning a new project, simply create the project with CLR support. The following is an example to create a new CLR Console C++ project in Visual Studio.

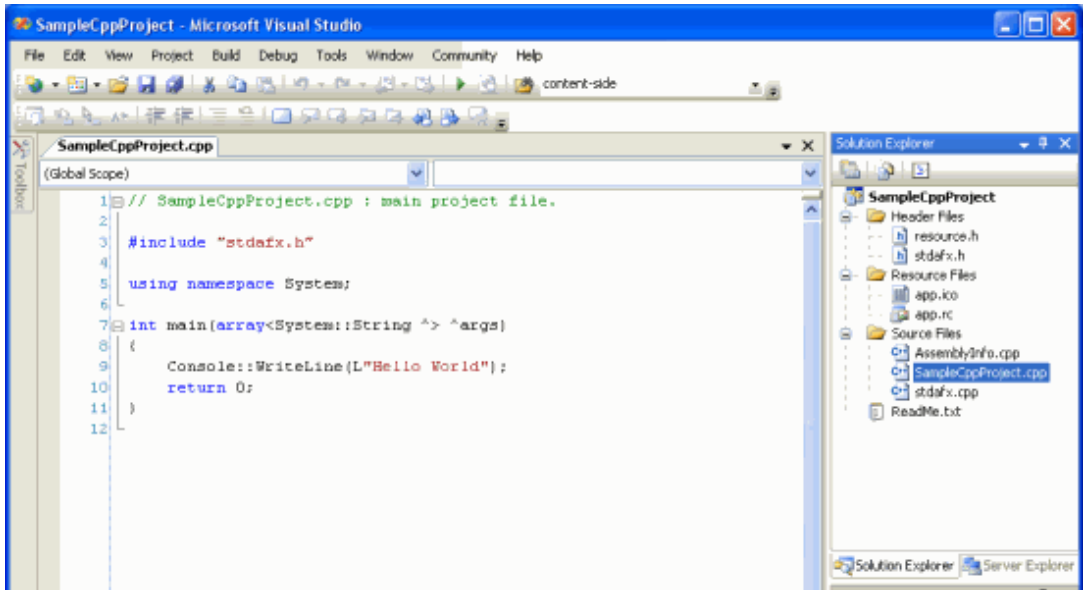
» Go to File --> New Project.



» The New Project window will open. Expand **Visual C++** then click on **CLR**. Select **CLR Console Application** under Templates. Make sure the Location is correct and give the project a name. Once you are done, click on OK.

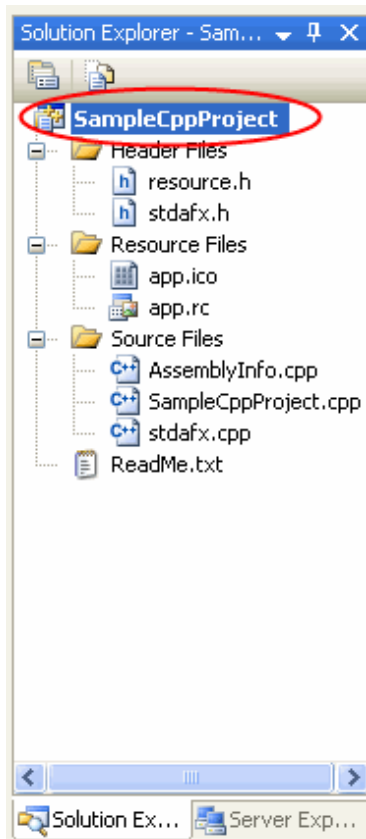


» The code source should open with a sample "Hello World" program.

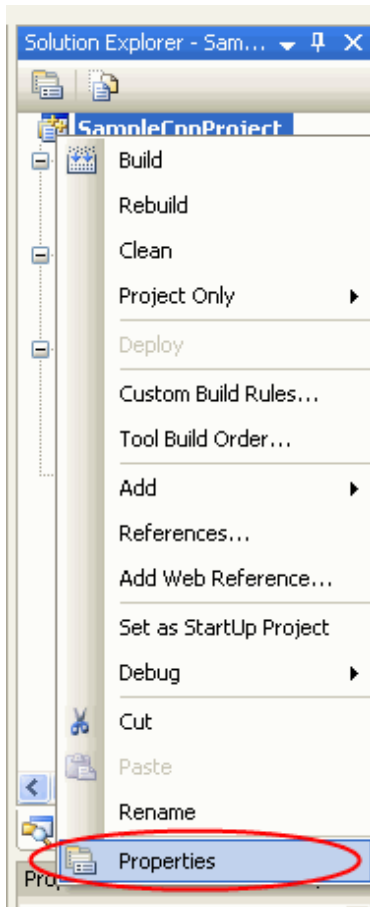


Now we need to add **Euvis_Module_V1p0.dll** to the references of your solution in Visual Studio.

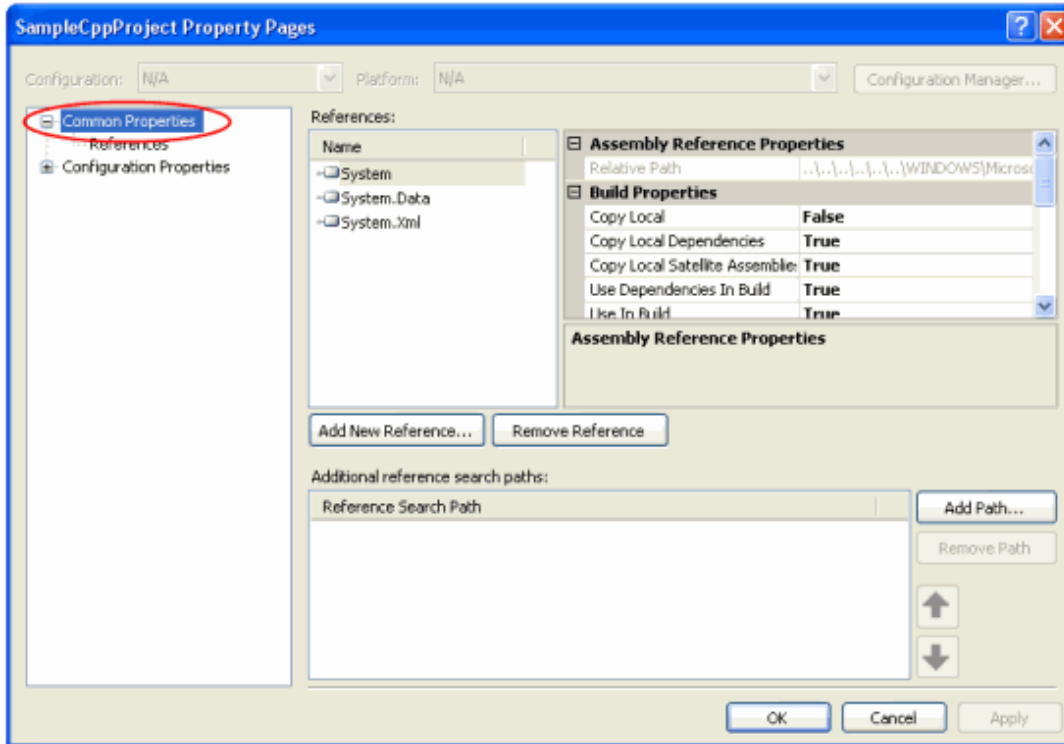
» Right click your solutions file in the Solution Explorer. Your solution will probably be called something different than the one shown.



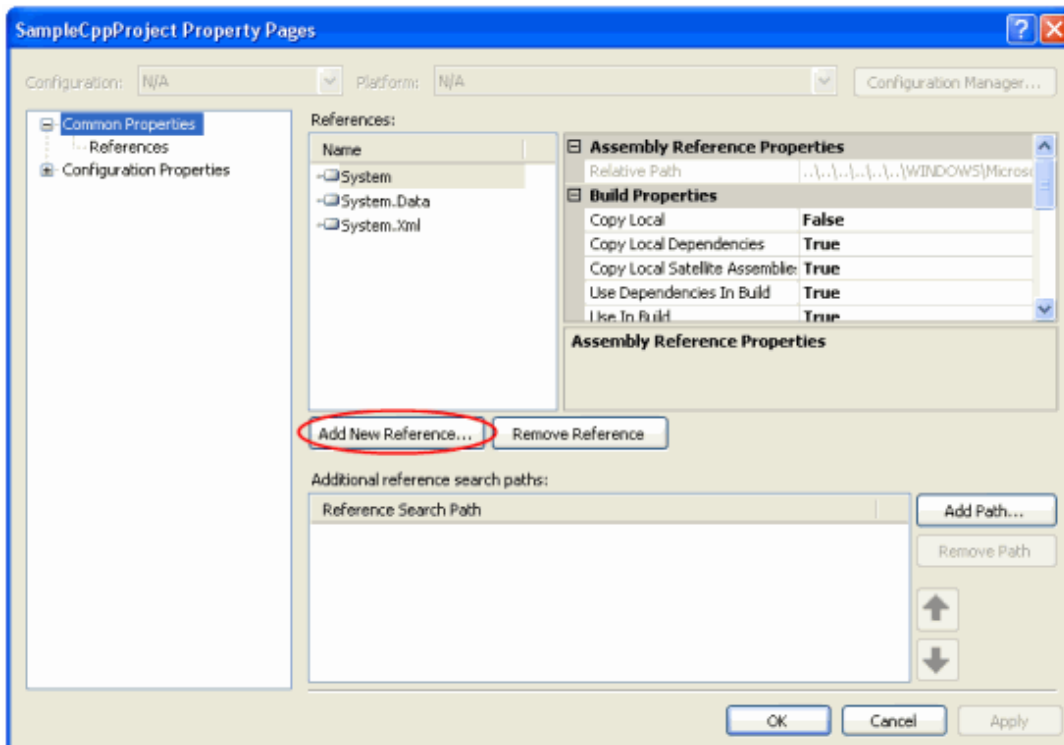
» Click on **Properties** near the bottom of the pop out menu.



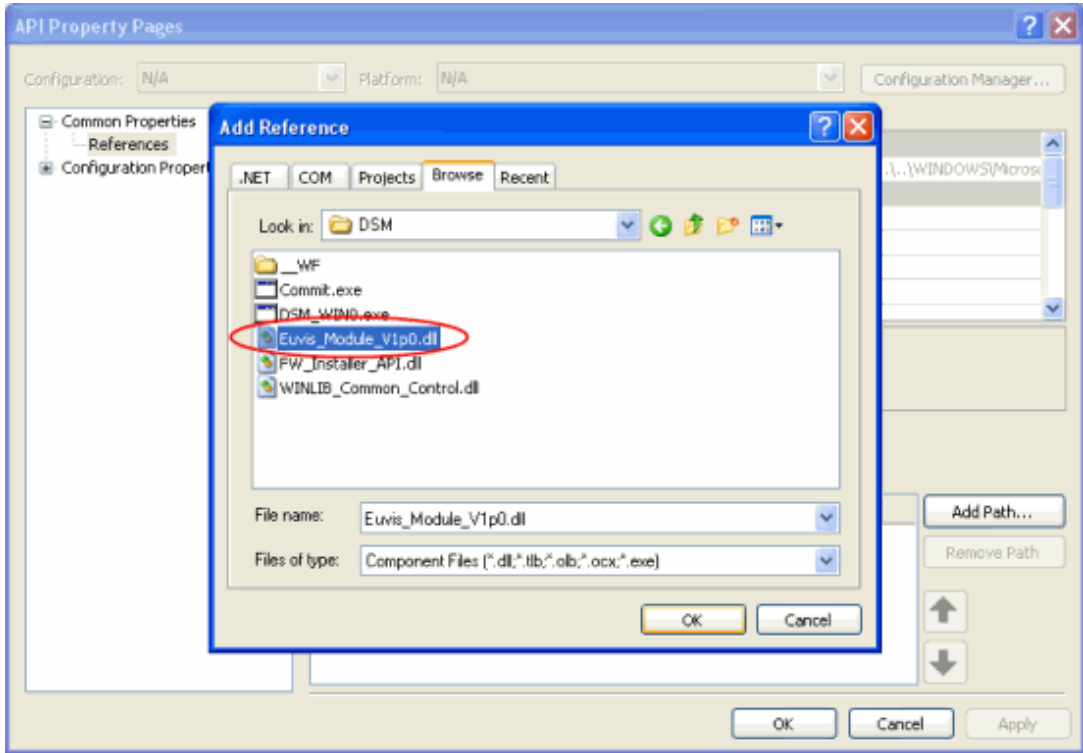
» A new Property window will appear. Click on the "+" next to Common Properties at the top on the left column.



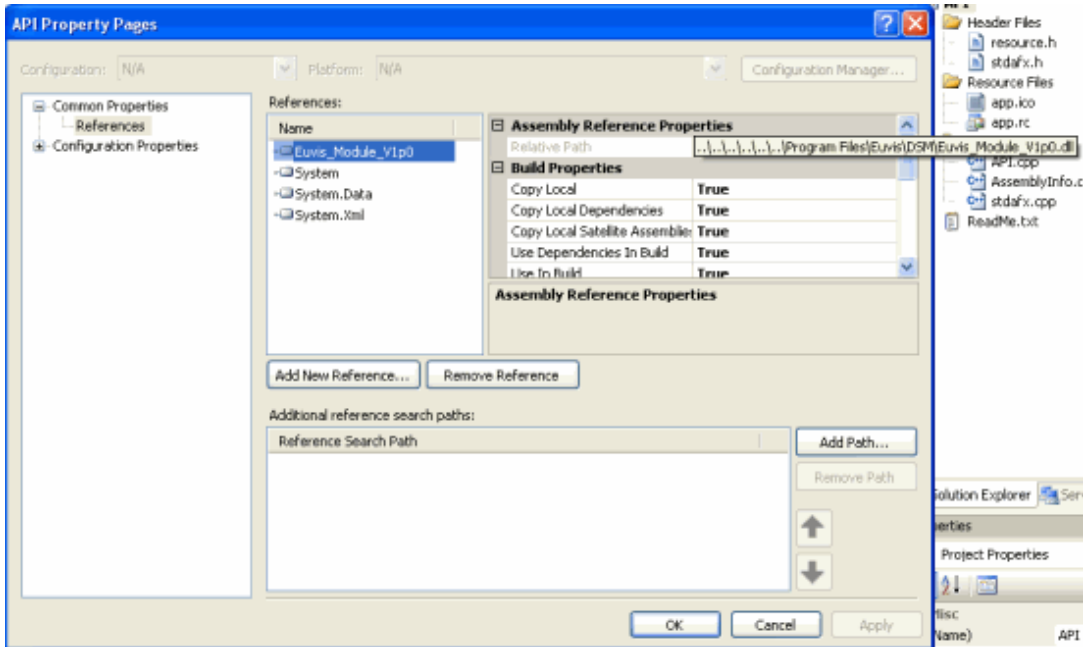
» Click on **References**. Now click on the **Add New Reference** button.



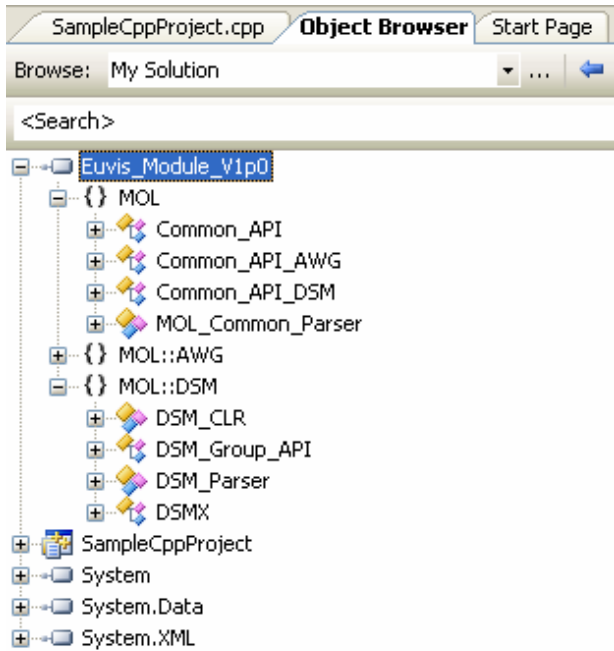
» Click on the **Browse** tab and then browse to the main AWG directory. By default the directory is C:\Program Files\Euvis\AWG. Once you get to the directory, double click on **Euvis_Module_V1p0.dll** then click OK.



» Euvis_Module_V1p0 should now be listed under References along with its path.



The **Euvis_Module_V1p0** file provides you with all of the functions and properties for you to control your waveforms. You can use Visual Studio's Object Browser to view all of the functions and properties that are available to you.



In order to use the functions and properties in the API, you must use the "using" directive to include the AWG and MOL namespaces.

```
using namespace MOL;  
using namespace MOL::AWG;
```

Basic Structure -

For each waveform that is created by the AWG, the controlling program should adhere to the following structure:

1. **Intialization of board(s)**
2. **Configure control settings**
3. **Waveform parameter definitions**
4. **Download data**
5. **Start AWG memory**
6. **Stop memory when complete**

Intialization of board(s)

The initialization of the AWG board only needs to be done once in a program assuming that the AWG is not disconnected. Initialization must be done first before all other steps. Initialization is a multiple step process. First you should find out how many AWG boards are connected by reading the [number](#) property in the *AWG_Group_API* class. Then call the [get_sn](#) member function in the *AWG_Group_API* class to get the board Series Number. The next step is the most important step in the entire AWG program writing process: after you have instantiated a *AWG_API* object, you must initialize it by calling the [ini](#) with the board Series Number as argument. Simply having an *AWG_API* object without calling the [ini](#) method will likely cause your program to fail.

Always call the `ini` function immediately after instantiating a *AWG_API* object!

Configure control settings

Control settings should be redefined everytime a new waveform is implemented. The reason for this is that if you do not change the control settings, the AWG will carry over settings from the previous waveform. Even if two waveforms have the same settings, you should redefine the settings just in case.

First set the mode (either Free Run, Master or Slave) and the related multi-board settings:

Mode Settings	Implementation
Slave Enable/Disable:	slave_mode
Loop Count:	loop_count
Auto-Arm Enable/Disable	auto_armed
SYNCO Enable/Disable	synco_enable
Internal SYNCI Enable/Disable	synci_enable

Then set the general control settings that apply to all modes:

Control Settings	Implementation
-------------------------	-----------------------

Oversampling Factor:	<u>over_sampling_rate</u>
Marker 1 Enable/Disable:	<u>marker1_enabled</u>
Marker 1 Polarity	<u>marker1_polarity</u>
Marker 2 Enable/Disable:	<u>marker2_enabled</u>
Marker 2 Polarity	<u>marker2_polarity</u>
Marker 3 Enable/Disable:	<u>marker3_enabled</u>
Marker 3 Polarity	<u>marker3_polarity</u>
Data Length Enable/Disable:	<u>data_length_enable</u>

Waveform parameter definitions

There are three ways you can set up your waveforms:

Built-In Waveform Parameters	Implementation
Waveform code:	<u>code</u>
Delay	<u>delay</u>

The second way to define a waveform is to use the [user_define_bulk](#) function.

The final way to define a waveform is to use the [user_define_file](#) function.

There are two mandatory parameters that you must set for ALL waveforms.

Mandatory Parameters	Implementation
Data Length	<u>data_length</u>
Memory Depth	<u>memory_depth</u>

Finally, there are optional parameters that may be useful for your waveform.

Optional Parameters	Implementation
Marker 1 Start	<u>marker1_start</u>
Marker 1 Width	<u>marker1_width</u>
Marker 2 Start	<u>marker2_start</u>
Marker 2 Width	<u>marker2_width</u>
Marker 3 Start	<u>marker3_start</u>
Marker 3 Width	<u>marker3_width</u>
Phase Reset Time 1	<u>RESET_T1</u>
Phase Reset Time 2	<u>RESET_T2</u>
Phase Reset Time 3	<u>RESET_T3</u>

Download data

The next step is to download all of the information stored in the computer memory to the memory on the AWG. To do this for the built-in waveform you should use the [download](#) method.

For the User-Defined functions, whenever the functions are called the data is automatically downloaded to the AWG memory.

Start memory

When in Free Run mode, start the memory by using the [restart](#) method.

For Master mode, use the [arm](#) method to arm the board. When armed, the Master board will wait for the trigger signal then start the memory.

For Slave mode, use the [slave mode](#) property to arm the board. When armed, the Master board will wait for the trigger signal then start the memory.

Stop memory

With the absence of a stop command, the AWG will continue to run until the power is shut off. If you want to stop the AWG memory use the [stop](#) method.

Methods -

All methods with the exception of **get_sn** are part of the *Common_API* class or the *AWG_API* class. You must instantiate an object of the *Common_API* class in order to use any of the methods in that class, you must instantiate an object of the *AWG_API* class in order to use any of the methods in that class and you must instantiate an object of the *AWG_Group_API* class to use the **get_sn** method. You must use the "." operator to access any of the methods. In the examples, we have used an example **com** object of the *Common_API* class, an example **awg** object of the *AWG_API* class and an example **ug** object of the *AWG_Group_API* class.

Methods Quick Reference -

Property	Data Type
arm	Void arm()
download	Void download()
flush	Void flush()
get_sn	UInt16 get_sn(UInt16 indexNumber)
ini	Void ini(UInt32 seriesNumber)
memory_clock_toggle	Void memory_clock_toggle()
memory_data	UInt64 memory_data(UInt32 memoryAddress)
restart	Void restart()
stop	Void stop()
USB_Reconnect	Void USB_Reconnect()
user_define_bulk	Void user_define_bulk(UInt32* freqArray, UInt32 arraySize)
	Void user_define_bulk(UInt32* freqArray, UInt32* controlArray, UInt32 arraySize)
user_define_file	Void user_define_file(String fileName, Double inputClock)

AWG_Group_API -

[get_sn](#)

get_sn

Description

Retrieves the Series Number from the physical device.

Syntax

```
C++  
UInt16 get_sn(UInt16 deviceIndexNum)
```

Arguments

deviceIndexNum	index number of the AWG device
----------------	--------------------------------

Return Value

seriesNumber	Series Number of AWG device
--------------	-----------------------------

Example

```
int seriesNumber;  
seriesNumber = ug.get_sn(0);
```

Notes

This method is part of the *AWG_Group_API* class. The Series Number is a unique number that identifies each AWG board. You should always use this method to get the Series Number in order to initialize the AWG board.

If this method returns with "0" it means that there was no AWG board found. If you try to initialize an *AWG_API* object (using the [ini](#) method) with argument "0", your program will fail. You should have a conditional test to test whether the `get_sn` function returns with "0" and if it does, to stop further execution and to display an error message.

Common_API -

[arm](#)
[download](#)
[flush](#)
[ini](#)
[memory_clock_toggle](#)
[restart](#)
[stop](#)
[USB_Reconnect](#)
[user_define_bulk](#)
[user_define_file](#)

arm

Description

Arms AWG board in Master mode.

Syntax

```
C++  
void arm()
```

Arguments

None

Return Value

None

Example

```
awg.arm();
```

Notes

This function only arms the board in Master mode. It will not arm the board in Slave mode. To arm the board in Slave mode, use the [slave_mode](#) property.

[return to top](#)

download

Description

Downloads the built-in waveform parameters to the AWG board.

Syntax

```
C++  
void download()
```

Arguments

None

Return Value

None

Example

```
awg.download();
```

Notes

This function is only needed to download the built-in waveform. If you use the [user_define_bulk](#) or the [user_define_file](#) functions then you should NOT use this function.

[return to top](#)

flush

Description

Resets memory to starting memory address.

Syntax

```
C++  
void flush()
```

Arguments

None

Return Value

None

Example

```
awg.flush();
```

Notes

It is a good idea to reset the memory after you stop it. This member method along with the memory stop method, [stop](#), should usually be used together.

[return to top](#)

ini

Description

Initializes the AWG board.

Syntax

```
C++  
void ini(UInt32 series_number)
```

Arguments

<code>series_number</code>	Series Number of the AWG board
----------------------------	--------------------------------

Return Value

None

Example

```
int seriesNumber = ug.get_sn(0);  
awg.ini(seriesNumber);
```

Notes

Please use the [get_sn](#) method to get the Series Number.

When you call the `ini` function, some of the member properties will be initialized to their default values. To see these default values, please see the [Properties](#) section main page.

[return to top](#)

memory_clock_toggle

Description

Toggles memory clock high or low.

Syntax

```
C++  
void memory_clock_toggle()
```

Arguments

None

Return Value

None

Example

```
awg.memory_clock_toggle();
```

Notes

When memory clock is high, this function will set it low; if memory clock is low, this function will set it high.

[return to top](#)

restart

Description

Starts memory.

Syntax

```
C++  
void restart()
```

Arguments

None

Return Value

None

Example

```
awg.restart();
```

Notes

When in Master or Slave mode, do not use this method to start the memory. Instead use the [arm](#) method to arm the Master mode AWG or the [slave_mode](#) property to arm the Slave mode AWG. After arming, the boards will wait for the trigger signal before starting the memory.

[return to top](#)

stop

Description

Stops the AWG memory.

Syntax

```
C++  
void stop();
```

Arguments

None

Return Value

None

Example

```
awg.stop();
```

Notes

It is a good idea to reset the memory after you stop it. This member method along with the memory reset method, [flush](#), should usually be used together.

[return to top](#)

USB_Reconnect

Description

Reconnects to USB after a disconnect between the computer and the AWG

Syntax

```
C++  
void USB_Reconnect()
```

Arguments

None

Return Value

None

Example

```
awg.USB_Reconnect();
```

Notes

You can use this method to reconnect to the AWG board if you accidentally disconnect the USB cable or if AWG was accidentally turned off.

[return to top](#)

user_define_bulk

Description

Chirping with bulk memory.

Syntax

```
C++  
void user_define_bulk(UInt32* bulkMemory, UInt32 numOfPoints)  
void user_define_bulk(UInt32* bulkMemory, UInt32* controlMemory, UInt32  
numOfPoints)
```

Arguments

<code>bulkMemory</code>	Bulk memory that will be used for the waveform
<code>controlMemory</code>	Control bit words for each value
<code>numOfPoints</code>	Number of values in the waveform

Return Value

None

Example

```
int i;
unsigned *u = new unsigned[16];
unsigned *c = new unsigned[16];
const unsigned FC_START = 0x800000;
const unsigned FC_STEP = 0x1000000;

for( i=1, u[0]=FC_START; i<16; i++ )
{
    if ( (i-1) < 4 )
    {
        c[i-1] = 3;
        u[i-1] = 0;
        continue;
    }

    u[i]=u[i-1]+FC_STEP;
}

awg.user_define_bulk( u, c, 16 );

delete u;
delete c;
```

Control Bit Words

The control bit words are basically three control bits that represent the markers. Bits 0,1,2 (LSB) control whether the markers are high or low. For each marker bit, "0" turns marker low while "1" turns marker high.

Notes

This method will essentially output the *data words* that are stored in an array. Note that there are two versions of this method. The first method with three arguments will output with the data words in the bulk memory array. The overloaded method with three arguments does the same but you also have the option of setting the markers on a point-by-point basis. This gives you more control than setting the marker properties ([marker1_start](#), [marker1_width](#), [marker2_start](#), [marker2_width](#), [marker3_start](#), and [marker3_width](#)) that only allow you to have a range of points.

For example if you used the `user_define_bulk` method with three arguments and you wanted a waveform with 10,000 points then you can only set each marker high for one data range, say points 2000 through 3000 for marker 1. But if you use the overloaded method with three arguments, then you have the option of setting each marker high more than once, such as for points 2000 through 3000 and then points 7446 through 8564 for marker 1. The same goes for each marker.

If you decide to use `user_define_bulk` with the overloaded three arguments, the default setting if you do not specify a control bit word in the control array is with each marker low.

If you are using the `user_define_bulk` method with three arguments then you must set the `marker1_start`, `marker1_width`, `marker2_start`, `marker2_width`, `marker3_start`, and `marker3_width` properties.

If you are using the overloaded `user_define_bulk` method with three arguments, the `marker1_start`, `marker1_width`, `marker2_start`, `marker2_width`, `marker3_start`, and

`marker3_width` properties will be ignored and you must specify markers using the control array.

When you call the `user_define_bulk` method, the program will automatically download the data to the AWG. There is no need use the [download](#) method.

[return to top](#)

user_define_file

Description

Loads custom user waveform files

Syntax

```
C++
UInt32 user_define_file(String filename, Double clock)
```

Arguments

<code>filename</code>	file name of custom user file
<code>clock</code>	frequency of the input clock

Return Value

<code>numOfPoints</code>	number of frequencies in user waveform file
--------------------------	---

Example

```
unsigned numOfPoints;
Double clock = 2e9;
numOfPoints = awg.user_define_file("user_defined_file.uda", clock);
```

Notes

The `user_define_file` will use the contents of an external `.uda` file, provided that the file is formatted the correct way. The file gives you complete control of the waveform data. The waveforms are not limited to linear waveforms that are available in the built-in waveforms.

If you are using `user_define_file` and specify either #Type "1" or "2" then you must set the [marker2_start](#) and [marker2_width](#) properties in order to get the correct markers. If you are using `user_define_file` and specify either #Type "5" or "6", the `marker2_start` and `marker2_width` properties will be ignored and you must specify markers with the control bit words within the `.uda` file.

If you are using `user_define_file` and specify either #Type "1" or "2" then you must set the [RESET_T1](#), [RESET_T2](#), and [RESET_T3](#) properties in order to get the correct phase reset by memory options. If you are using `user_define_file` and specify either #Type "5" or "6", the `RESET_T1`,

RESET_T2, and **RESET_T3** properties will be ignored and you must specify phase reset with the control bit words within the .uda file.

When you call the **user_define_bulk** method, the program will automatically download the data to the AWG. There is no need use the [download](#) method.

For more information regarding the .uda file and how to format it, please see the AWG Manual.

[return to top](#)

AWG_API -

[ini](#)
[memory_data](#)

ini

Description

Initializes the AWG module.

Syntax

```
C++  
void ini(UInt32 series_number)
```

Arguments

<code>series_number</code>	Series Number of the AWG module
----------------------------	---------------------------------

Return Value

None

Example

```
int seriesNumber = ug.get_sn(0);  
awg.ini(seriesNumber);
```

Notes

Please use the [get_sn](#) method to get the Series Number.

When you call the `ini` function, some of the member properties will be initialized to their default values. To see these default values, please see the [Properties](#) section main page.

[return to top](#)

memory_data

Description

Retrieves the waveform data word from memory

Syntax

```
C++  
UInt64 memory_data(UInt32 memAddress)
```

Arguments

memAddress	memory address
------------	----------------

Return Value

dataWord	data word
----------	-----------

Example

```
unsigned dataWord;  
unsigned memAddress = 0x0000A;  
dataWord = awg.memory_data(memAddress);
```

Notes

This method is a helpful debugging tool should you ever need it.

[return to top](#)

Properties -

All properties are part of the *Common_API* class or the *AWG_API* class except **number** which is part of the *AWG_Group_API* class. You must create an object of the *Common_API* class in order to use any of the properties in that class, you must create an object of the *AWG_API* class in order to use any of the properties in that class and you must create an object of the *AWG_Group_API* class in order to use any of the properties in that class. In the examples, we have used an example **com** object of the *Common_API* class, an example **awg** object of the *AWG_API* class and an example **ug** object of the *AWG_Group_API* class.

Default Property Values

The following is a list of default values that are set when you initialize the AWG with the [ini](#) method. If you do not change these values in your own program, they will remain at these default values so be sure to redefine these properties if you want to change waveform parameters.

Waveform	
marker1_start	0x0
marker1_width	0x8
marker2_start	0x0
marker2_width	0x8
marker3_start	0x0
marker3_width	0x8
code	0
data_length	0x10
data_length_enable	false
data_length_offset	0x192
delay	0
memory_depth	0x100000
Inverse Sinc Filter	
isinc_freq_scaling_factor	1.335
isinc_on	true
isinc_scaling_factor	0.6
wv_full_scale	0x1000 for AWG252 0x1000 for AWG452 0x800 for AWG801
wv_scaling_factor	0.975
Mode	
loop_count	0
Hardware Settings	
over_sampling_rate	1
memory_dll	true

marker1_enable	true
marker2_enable	true
marker2_polarity	1
marker3_enable	true
marker3_polarity	1
auto_armed	true
synco_enable	true
synco_enable	true
slave_mode	false

Properties Quick Reference -

Property	Data Type	Permissions
ate	Boolean	read / write
auto_armed	Boolean	read / write
Clock	Double	read / write
code	Int32	read / write
data_length	UInt64	read / write
data_length_enable	Boolean	read / write
data_length_offset	UInt64	read / write
delay	UInt64	read / write
device_aux	UInt32	read-only
device_cat	UInt32	read-only
device_name	String	read-only
device_sn	UInt32	read-only
device_subcat	UInt32	read-only
Dump_Directory	String	read / write
File_Directory	String	read / write
firmware_aux	UInt32	read-only
firmware_cat	UInt32	read-only
firmware_name	String	read-only
firmware_subversion	UInt32	read-only
firmware_version	UInt32	read-only
isinc_freq_scaling_factor	Double	read / write
isinc_on	Boolean	read / write
isinc_scaling_factor	Double	read / write
loop_count	UInt32	read / write
marker1_enabled	Boolean	read / write
marker1_polarity	Boolean	read / write
marker1_start	UInt32	read / write
marker1_width	UInt32	read / write
marker2_enabled	Boolean	read / write
marker2_polarity	Boolean	read / write
marker2_start	UInt32	read / write
marker2_width	UInt32	read / write
marker3_enabled	Boolean	read / write
marker3_polarity	Boolean	read / write
marker3_start	UInt32	read / write
marker3_width	UInt32	read / write
memory_clock	Int16	write-only
memory_clock_advance	UInt32	write-only
memory_depth	UInt64	read / write
memory_dll	Boolean	read / write
module_id	UInt32	read / write

Euvis AWG API Manual

module_name	SByte	read / write
number	UInt16	read / write
over_sampling_rate	Int16	read / write
slave_mode	Boolean	read / write
status	UInt32	read-only
sync_T1	UInt32	read / write
sync_T2	UInt32	read / write
synci_enable	Boolean	read / write
synco_enable	Boolean	read / write
trigger	Boolean	write-only
wv_full_scale	Int16	read / write
wv_scaling_factor	Double	read / write

AWG_Group_API -

[number](#)

number

Description

Specifies how many boards are connected.

Syntax

```
UInt16 number
```

Read

Gets how many boards are connected.

Example

```
int numOfBoards;  
numOfBoards = ug.number;
```

Write

Read-Only

Notes

This is part of the *AWG_Group_API* class.

Common_API -

[ate](#)
[auto_armed](#)
[Clock](#)
[code](#)
[data_length](#)
[data_length_enable](#)
[data_length_offset](#)
[delay](#)
[device_aux](#)
[device_cat](#)
[device_name](#)
[device_sn](#)
[device_subcat](#)
[Dump_Directory](#)
[File_Directory](#)
[firmware_aux](#)
[firmware_cat](#)
[firmware_name](#)
[firmware_subversion](#)
[firmware_version](#)
[loop_count](#)
[marker1_enabled](#)
[marker1_polarity](#)
[marker1_start](#)
[marker1_width](#)
[marker2_enabled](#)
[marker2_polarity](#)
[marker2_start](#)
[marker2_width](#)
[marker3_enabled](#)
[marker3_polarity](#)
[marker3_start](#)
[marker3_width](#)
[memory_clock](#)
[memory_clock_advance](#)
[memory_depth](#)
[memory_dll](#)
[module_id](#)
[module_name](#)
[over_sampling_rate](#)
[slave_mode](#)
[status](#)
[sync_T1](#)
[sync_T2](#)
[synci_enable](#)
[synco_enable](#)
[trigger](#)

ate

Description

Specifies ATE setting.

Syntax

```
Boolean ate
```

Read

Gets current status of ATE.

Example

```
bool ateStatus;  
ateStatus = awg.ate;
```

Write

Sets new ATE status;

Example

```
awg.ate = false;
```

Notes

Please refer to the AWG manual for a more detailed description of ATE.

[return to top](#)

auto_armed

Description

Specifies Auto-Arm status.

Syntax

```
Boolean auto_armed
```

Read

Gets status of Auto-Arm.

Example

```
bool autoArmStatus;  
autoArmStatus = awg.auto_armed;
```

Write

Sets Auto-Arm status.

Example

```
awg.auto_armed = false;
```

Notes

Usually you should only use Auto-Arm if you are in Master mode. For a more detailed explanation of Auto-Arm please see the AWG manual.

[return to top](#)

Clock

Description

Specifies the input clock into the AWG

Syntax

Double Clock

Read

Gets the current input clock frequency

Example

```
double inputClock;  
inputClock = awg.Clock;
```

Write

Sets new input clock frequency

Example

```
awg.Clock = 2000;
```

Notes

Units of Clock is in ???.

[return to top](#)

code

Description

Specifies waveform code.

Syntax

```
Int32 code
```

Read

Gets the current waveform code.

Example

```
int wfCode;  
wfCode = awg.code;
```

Write

Sets new waveform code.

Example

```
awg.code = 1;
```

Configurations

Code	Description
0	Frequency ramps up from start frequency to stop frequency then goes back to start frequency. Frequency vs. Time graph resembles sawteeth.
1	Frequency ramps up from start frequency to stop frequency then ramps down. Frequency vs. Time graph resembles triangles.
2	Frequency ramps down from a higher frequency to a lower frequency then goes back to the original higher frequency. In other words, the reverse of wave code "0".

Notes

???

[return to top](#)

data_length

Description

Specifies the Data Length which is the amount of memory addresses to make available for chirping. The larger the Data Length, the more frequencies that can be output.

Syntax

```
UInt64 data_length
```

Read

Gets the current Data Length.

Example

```
unsigned wFDL;  
wFDL = awg.data_length;
```

Write

Sets the new Data Length.

Example

```
awg.data_length = 0x40;
```

Notes

The Data Length should always be either shorter or equal to the Memory Depth. If you want the AWG to go back to the starting value before reaching the Memory Depth, you should turn on the Data Length using the [data_length_enable](#) property. You would use the Data Length if the number of data values in your waveform is not one of the Memory Depth settings. Please note that even if the Data Length is disabled, you should still specify the correct Data Length. For example, if the number of data values in your desired waveform is 65,536 (which is one of the Memory Depth options) then you would set the Memory Depth to 65536 and disable the Data Length but you still would need to set Data Length to the hexadecimal equivalent of 65536 (4000).

The Data Length is useful for users who want a duty cycle waveform. To do this, set the Data Length but set the [data_length_enable](#) to "false". The AWG will output the waveform data up to the Data Length and then will output the starting amplitude until it reaches the Memory Depth. To see the different patterns of waveforms available, please see the AWG Manual.

Users who want a duty cycle waveform but who have a total number of data values not equal to one of the Memory Depth settings can use the [user_define_bulk](#) or the [user_define_file](#) functions.

Data Length will not work for waveforms that have data shorter than 1000 values even if Data Length is enabled.

[return to top](#)

data_length_enable

Description

Specifies whether Data Length is enabled or disabled.

Syntax

```
Boolean data_length_enable
```

Read

Sees if Data Length is enabled or disabled.

Example

```
bool dataLengthEnable;  
dataLengthEnable = awg.data_length_enable;
```

Write

Enables or disables Data Length.

Example

```
awg.data_length_enable = true;
```

Notes

Data Length will not be available to you for waveforms that have data shorter than 1000 values even if Data Length is enabled.

[return to top](#)

data_length_offset

Description

Specifies the Data Length Offset. At 2.5 Ghz, it is best to set this to 0x192.

Syntax

```
UInt64 data_length_offset
```

Read

Gets current Data Length Offset.

Example

```
unsigned dataLengthOffset;  
dataLengthOffset = awg.data_length_offset;
```

Write

Sets new Data Length Offset.

Example

```
awg.data_length_offset = 0x192;
```

[return to top](#)

delay

Description

Specifies the delay which is the number of memory addresses to keep at the start value before the waveform starts.

Syntax

```
UInt64 delay
```

Read

Gets current delay.

Example

```
unsigned wfDelay;  
wfDelay = awg.delay;
```

Write

Sets new delay.

Example

```
awg.delay = 0xFF;
```

Notes

Please note that the delay length is included in the Data Length and Memory Depth. This means that if you had 100 data values in the waveform and you had Memory Depth (or Data Length) set to 100 points and delay set to 10 points, then the AWG will output the start value for the first 10 points and then will only proceed through the first 90 values (including the start value) that you specified before then starting over again.

[return to top](#)

device_aux

Description

Specifies the Auxiliary code of the device.

Syntax

```
UInt32 device_aux
```

Read

Gets the Auxiliary code of the device.

Example

```
int deviceAuxCode;  
deviceAuxCode = awg.device_aux;
```

Write

Read-Only.

[return to top](#)

device_cat

Description

Specifies the Category number of the device.

Syntax

```
UInt32 device_cat
```

Read

Gets the Category number of the device.

Example

```
int deviceCatNum;  
deviceCatNum = awg.device_cat;
```

Write

Read-Only.

[return to top](#)

device_name

Description

Specifies the Name of the device.

Syntax

```
String device_name
```

Read

Gets the Name of the device.

Example

```
String^ deviceName;  
deviceName = awg.device_name;
```

Write

Read-Only.

[return to top](#)

device_sn

Description

Specifies the Series Number of the device.

Syntax

```
UInt32 device_sn
```

Read

Gets the Series Number of the device.

Example

```
int seriesNumber;  
seriesNumber = awg.device_sn;
```

Write

Read-Only.

[return to top](#)

device_subcat

Description

Specifies the Subcategory of the device.

Syntax

```
UInt32 device_subcat
```

Read

Gets the Subcategory of the device.

Example

```
int deviceSubcategory;  
deviceSubcategory = awg.device_subcat;
```

Write

Read-Only.

[return to top](#)

Dump_Directory

Description

Specifies where to save memory dump files.

Syntax

```
String Dump_Directory
```

Read

Gets the current location of memory dump files.

Example

```
String^ dumpDirectory;  
dumpDirectory = awg.Dump_Directory;
```

Write

Sets new location for memory dump files. Specified as a string so must have double quotes.

Example

```
awg.File_Directory = ".\MD";
```

[return to top](#)

File_Directory

Description

Specifies where to save and load waveform files.

Syntax

```
String File_Directory
```

Read

Gets the current location of waveform files.

Example

```
String^ fileDirectory;  
fileDirectory = awg.File_Directory;
```

Write

Sets new location for waveform files. Specified as a string so must have double quotes.

Example

```
awg.File_Directory = ".\__WF";
```

[return to top](#)

firmware_aux

Description

Specifies the Auxillary code of the firmware.

Syntax

```
UInt32 firmware_aux
```

Read

Gets the Auxillary code of the firmware.

Example

```
int fwAuxCode;  
fwAuxCode = awg.firmware_aux;
```

Write

Read-Only.

[return to top](#)

firmware_cat

Description

Specifies the Category of the firmware.

Syntax

```
UInt32 firmware_cat
```

Read

Gets the Category of the firmware.

Example

```
int fwCatNum;  
fwCatNum = awg.firmware_cat;
```

Write

Read-Only.

[return to top](#)

firmware_name

Description

Specifies the Name of the firmware.

Syntax

```
String firmware_name
```

Read

Gets the Name of the firmware.

Example

```
String^ fwName;  
fwName = awg.firmware_name;
```

Write

Read-Only.

[return to top](#)

firmware_subversion

Description

Specifies the Subversion of the firmware.

Syntax

```
UInt32 firmware_subversion
```

Read

Gets the Subversion of the firmware.

Example

```
int fwSubversion;  
fwSubversion = awg.firmware_subversion;
```

Write

Read-Only.

[return to top](#)

firmware_version

Description

Specifies the Version of the firmware.

Syntax

```
UInt32 firmware_version
```

Read

Gets the Version of the firmware.

Example

```
int fwVersion;  
fwVersion = awg.firmware;
```

Write

Read-Only.

[return to top](#)

loop_count

Description

Specifies the loop count of the chirp.

Syntax

```
UInt32 loop_count
```

Read

Gets the current loop count.

Example

```
unsigned loopCount;  
loopCount = awg.loop_count;
```

Write

Sets the new loop count.

Example

```
awg.loop_count = 0xA;
```

Notes

When this property is set to "0", the loop count is set to infinite so the AWG will be in Free Run mode. Be sure to set this property at the start of every chirp especially if you want the AWG to be operated in Free Run mode. Failure to set **loop_count** might result in unpredictable behavior by the AWG.

[return to top](#)

marker1_enabled

Description

Specifies marker signal status.

Syntax

```
Boolean marker1_enabled
```

Read

Gets Marker 1 status.

Example

```
bool markerStatus;  
marker1Status = awg.marker1_enabled;
```

Write

Read-Only

Notes

There are three sets of markers listed in the Object Browser. Marker 1 is always enabled and its polarity cannot be changed.

[return to top](#)

marker1_polarity

Description

Selects the polarity of the output marker.

Syntax

```
Boolean marker1_polarity
```

Read

Gets polarity of output marker.

Example

```
bool markerPolarity;  
markerPolarity = awg.marker1_polarity;
```

Write

Read-Only

Notes

There are three sets of markers listed in the Object Browser. Marker 1 is always enabled and its polarity cannot be changed.

[return to top](#)

marker1_start

Description

Specifies the starting point of the output marker.

Syntax

```
UInt32 marker1_start
```

Read

Gets the starting point of the marker.

Example

```
unsigned markerStart;  
markerStart = awg.marker1_start;
```

Write

Sets the starting point of the marker.

Example

```
awg.marker1_start = 10;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker1_width

Description

Specifies how many data points the marker is active for.

Syntax

```
UInt32 marker1_width
```

Read

Gets the width of the marker.

Example

```
unsigned markerWidth;  
markerWidth = awg.marker1_width;
```

Write

Sets the width of the marker.

Example

```
avg.marker1_width = 100;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker2_enabled

Description

Specifies marker signal status.

Syntax

```
Boolean marker2_enabled
```

Read

Gets markers status.

Example

```
bool markerStatus;  
markerStatus = avg.marker2_enabled;
```

Write

Sets new markers status.

Example

```
avg.marker2_enabled = true;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker2_polarity

Description

Selects the polarity of the output marker.

Syntax

```
Boolean marker2_polarity
```

Read

Gets polarity of output marker.

Example

```
bool markerPolarity;  
markerPolarity = awg.marker2_polarity;
```

Write

Sets polarity of output marker.

Example

```
awg.marker2_polarity = false;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker2_start

Description

Specifies the starting point of the output marker.

Syntax

```
UInt32 marker2_start
```

Read

Gets the starting point of the marker.

Example

```
unsigned markerStart;
```

```
markerStart = awg.marker2_start;
```

Write

Sets the starting point of the marker.

Example

```
awg.marker2_start = 10;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker2_width

Description

Specifies how many data points the marker is active for.

Syntax

```
UInt32 marker2_width
```

Read

Gets the width of the marker.

Example

```
unsigned markerWidth;  
markerWidth = awg.marker2_width;
```

Write

Sets the width of the marker.

Example

```
awg.marker2_width = 100;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker3_enabled

Description

Specifies marker signal status.

Syntax

```
Boolean marker3_enabled
```

Read

Gets markers status.

Example

```
bool markerStatus;  
markerStatus = awg.marker3_enabled;
```

Write

Sets new markers status.

Example

```
awg.marker3_enabled = true;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker3_polarity

Description

Selects the polarity of the output marker.

Syntax

```
Boolean marker3_polarity
```

Read

Gets polarity of output marker.

Example

```
bool markerPolarity;  
markerPolarity = awg.marker3_polarity;
```

Write

Sets polarity of output marker.

Example

```
awg.marker3_polarity = false;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker3_start

Description

Specifies the starting point of the output marker.

Syntax

```
UInt32 marker3_start
```

Read

Gets the starting point of the marker.

Example

```
unsigned markerStart;  
markerStart = awg.marker3_start;
```

Write

Sets the starting point of the marker.

Example

```
awg.marker3_start = 10;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

marker3_width

Description

Specifies how many data points the marker is active for.

Syntax

```
UInt32 marker3_width
```

Read

Gets the width of the marker.

Example

```
unsigned markerWidth;  
markerWidth = awg.marker3_width;
```

Write

Sets the width of the marker.

Example

```
awg.marker3_width = 100;
```

Notes

There are three sets of markers listed in the Object Browser.

[return to top](#)

memory_clock

Description

Toggles memory clock either low or high;

Syntax

```
Int16 memory_clock
```

Read

Write-Only

Write

Toggles memory clock for "1" and "0" for low.

Example

```
awg.memory_clock = 0;
```

Notes

You should only toggle the memory clock when the memory is stopped.

[return to top](#)

memory_clock_advance

Description

Specifies how many memory addresses to skip.

Syntax

```
UInt32 memory_clock_advance
```

Read

Write-Only

Write

Advances memory by amount user specifies.

Example

```
awg.memory_clock_advance = 6;
```

Notes

You should only advance the memory address when the memory is stopped.

[return to top](#)

memory_depth

Description

Specifies the Memory Depth which is the maximum number of memory addresses to make available in the device.

Syntax

```
UInt64 memory_depth
```

Read

Gets the current Memory Depth.

Example

```
unsigned memoryDepth;  
memoryDepth = awg.memory_depth;
```

Write

Sets a new Memory Depth.

Example

```
awg.memory_depth = 0x100000;
```

Notes

Memory Depth is a hardware limited setting.

You should always set the memory depth to 0x100000 to be able to use the full memory depth. If the number of data in your waveform is not equal to the memory depth, you must set the Data Length using the [data_length](#) property.

[return to top](#)

memory_dll

Description

Gets memory DLL status.

Syntax

```
Boolean memory_dll
```

Read

Sees if memory DLL is on or off.

Example

```
bool memDLLStatus;  
memDLLStatus = awg.memory_dll;
```

Write

Sets new memory DLL status.

Example

```
awg.memory_dll = true;
```

Notes

For a more detailed explanation of the memory DLL please see the AWG manual.

[return to top](#)

module_id

Description

Specifies the identification number of the board

Syntax

```
UInt32 module_id
```

Read

Reads the identification number of board.

Example

```
unsigned moduleID;  
moduleID = awg.module_id;
```

Write

Sets new identification number of board.

Example

```
avg.module_id = 28;
```

Notes

???

[return to top](#)

module_name

Description

Specifies the name of the board

Syntax

```
SByte module_name
```

Read

Gets the current name of the board

Example

???

Write

Sets the new name of the board

Example

???

Notes

???

[return to top](#)

over_sampling_rate

Description

Specifies the oversampling factor.

Syntax

```
Int16 over_sampling_rate
```

Read

Gets the current oversampling factor.

Example

```
int oversamplingFactor;  
oversamplingFactor = awg.over_sampling_rate;
```

Write

Sets new oversampling factor.

Example

```
awg.over_sample_rate = 2;
```

Notes

When this property is set to "1", the oversampling factor is 1. When set to "2", the oversampling factor is 2. When set to "3", the oversampling factor is 4. For more information regarding the oversampling factor, please see the AWG manual.

[return to top](#)

slave_mode

Description

Specifies the Slave mode status.

Syntax

```
Boolean slave_mode
```

Read

Gets current Slave mode status.

Example

```
bool slaveModeStatus;
```

```
slaveModeStatus = awg.slave_mode;
```

Write

Sets new Slave mode status.

Example

```
awg.slave_mode = false;
```

Notes

If you intend to be in Master or Slave mode, please remember to set the loop count with the [loop_count](#) property.

This property is also used to arm the Slave mode board. To arm the board, simply set the property to "true":

```
awg.slave_mode = true;
```

[return to top](#)

status

Description

Specifies updated status of device.

Syntax

```
UInt32 status
```

Read

Gets the status of the device. Refer to table below for description of status configurations.

Example

```
unsigned deviceStatus;  
deviceStatus = awg.status;
```

Write

Read-Only

Configurations

--	--	--	--

Decimal	Hex Code	Binary Code	Status
1	0x1	00000001	Armed
2	0x2	00000010	Triggered
4	0x4	00000100	In Loop
8	0x8	00001000	Auto-Armed
16	0x10	00010000	Slave
32	0x20	00100000	Slave Wait
64	0x40	01000000	Infinite Loop
128	0x80	10000000	Data Length Enabled

Note that the device can be simultaneously be in multiple states. In binary code, each "1" bit represents a state. If there is more than one state, then the binary code will have multiple "1" bits. For example if the device was Armed, Triggered, In Loop and Auto-Armed, then the binary code would be "00001111". Status can be output in any format so it is best to output it in binary code.

[return to top](#)

sync_T1

Description

Specifies the T_{SYNC1} time.

Syntax

```
UInt32 sync_T1
```

Read

Gets the current T_{SYNC1} time.

Example

```
unsigned tSync1;
tSync1 = awg.sync_T1;
```

Write

Sets a new T_{SYNC1} time. Minimum value of "0x0" and maximum value of "0xFF" (255).

Example

```
awg.sync_T1 = 0xAA;
```

Notes

This property is only available if SYNCO is enabled with the [synco_enable](#) property. For a more detailed discussion of the T_{SYNC1} time, please see the AWG Manual.

[return to top](#)

sync_T2

Description

Specifies the T_{SYNC2} time.

Syntax

```
UInt32 sync_T2
```

Read

Gets the current T_{SYNC2} time.

Example

```
unsigned tSync2;  
tSync2 = awg.sync_T2;
```

Write

Sets a new T_{SYNC2} time. Specified. Minimum value of "0x0" and maximum value of "0xFF" (255).

Example

```
awg.sync_T2 = 0xBB;
```

Notes

This property is only available if SYNCO is enabled with the [synco_enable](#) property. For a more detailed discussion of the T_{SYNC1} time, please see the AWG Manual.

[return to top](#)

synci_enable

Description

Specifies internal SYNCI status.

Syntax

```
Boolean slave_mode
```

Read

Gets status of internal SYNCI.

Example

```
bool synciStatus;  
synciStatus = awg.synci_enable;
```

Write

Sets new internal SYNCI status.

Example

```
awg.synci_enable = false;
```

Notes

For more information on the Internal SYNCI signal please see the AWG manual.

[return to top](#)

synco_enable

Description

Specifies SYNCO signal status.

Syntax

```
Boolean synco_enable
```

Read

Gets current status of SYNCO signal.

Example

```
bool syncoStatus;  
syncoStatus = awg.synco_enable;
```

Write

Sets new SYNCO signal status.

Example

```
awg.synco_enable = true;
```

Notes

The SYNCO signal should normally only be used to synchronize a Slave board. The SYNCO signal is always off when in Slave mode regardless of the **synco_enable** setting.

[return to top](#)

trigger

Description

Pulses the software trigger

Syntax

```
Boolean trigger
```

Read

Write-only

Write

Activates the software trigger

Example

```
awg.trigger = 1;
```

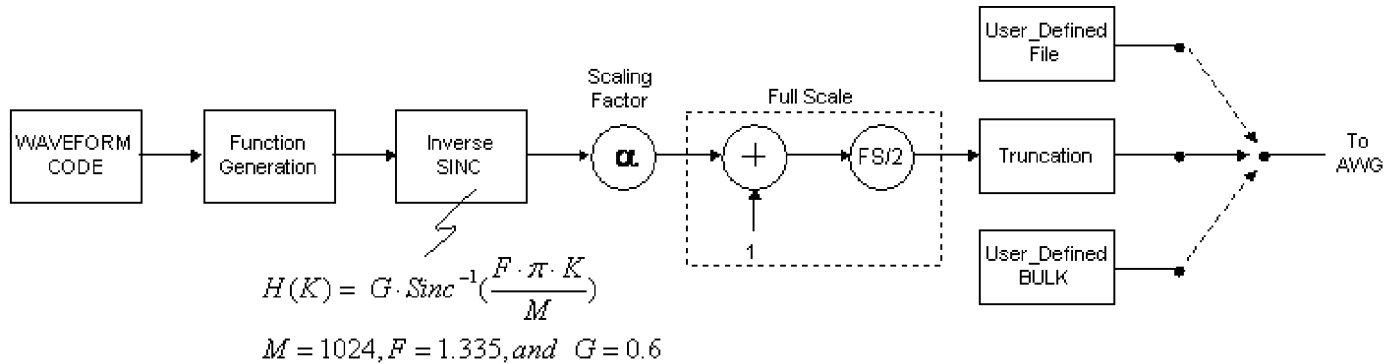
Notes

To activate the trigger, simply set the property to "1". There is no need to reset it to "0" after activating it.

[return to top](#)

AWG_API -

Due to the windowing effects of using time-limited waveform data, the waveform spectrum will appear to have had a sinc filter applied. The API includes an inverse sinc filter designed to minimize these effects when using a one of the prestored waveform styles. After the inverse sinc filter is a scaling factor to reduce the amplitude of the waveform data, which is followed by a positive offset and scaling to bring the waveform output to non-negative values. With the following properties you can adjust these filters.



- [isinc_freq_scaling_factor](#) (F)
- [isinc_on](#)
- [isinc_scaling_factor](#) (G)
- [wv_full_scale](#) (Full Scale)
- [wv_scaling_factor](#) (a)

isinc_freq_scaling_factor

Description

This property scales the frequencies to which the inverse sinc filter will be applied.

Syntax

```
Double isinc_freq_scaling_factor
```

Read

Gets the current inverse sinc filter frequency scaling factor

Example

```
double F;
F = awg.isinc_freq_scaling_factor;
```

Write

Sets the inverse sinc filter frequency scaling factor

Example

```
awg.isinc_freq_scaling_factor = 1.335;
```

Notes

This is the factor F in the diagram at the top of this page.

[return to top](#)

isinc_on

Description

Specifies whether or not to use the inverse sinc filter.

Syntax

```
Boolean isinc_on
```

Read

Sees if isinc filter is on

Example

```
bool isincStatus;  
isincStatus = awg.isinc_on;
```

Write

Enables or disables isinc filter

Example

```
awg.isinc_on = true;
```

Notes

If you see that your spectrum appears to have been attenuated by a sinc function, you should try to use this inverse sinc filter.

[return to top](#)

isinc_scaling_factor

Description

This property scales the amplitude of the inverse sinc filter to be applied.

Syntax

```
Double isinc_scaling_factor
```

Read

Gets the current inverse sinc filter amplitude scaling factor

Example

```
double G;  
G = awg.isinc_scaling_factor;
```

Write

Sets the inverse sinc filter amplitude scaling factor

Example

```
awg.isinc_scaling_factor = 0.6;
```

Notes

This is the factor G in the diagram at the top of this page.

[return to top](#)

wv_full_scale

Description

This property determines the number of amplitude resolution points in the full scale output waveform.

Syntax

```
Int16 wv_full_scale
```

Read

Gets the current value of the full scale amplitude.

Example

```
unsigned FullScale;  
FullScale = awg.wv_full_scale;
```

Write

Sets the full scale amplitude of the output waveform

Example

```
awg.wv_full_scale = 0x1000;
```

Notes

This is the number of amplitude points to use for the full scale and is indicated by FS in the diagram at the top of this page. This value must be a natural number (positive integer).

Please note that the maximum full scale is 4096 points (0x1000) for the AWG252 and the AWG452, while the maximum full scale is 2048 points (0x800) for the AWG801.

[return to top](#)

wv_scaling_factor

Description

This property scales the waveform proportionally.

Syntax

```
Double wv_scaling_factor
```

Read

Gets the current value of the waveform scaling property.

Example

```
double alpha;  
alpha = awg.wv_scaling_factor;
```

Write

Sets the waveform scaling factor.

Example

```
avg.wv_scaling_factor = 0.975;
```

Notes

Please use a positive number less than 1. If truncation of the waveform occurs, particularly with two-tone or multi-tone waveforms, when the component tone waveforms add in-phase beyond the maximum amplitude, please use this scaling factor to reduce the total power of your waveform.

[return to top](#)

Helpful Tips -

This section will hopefully help you solve some of the more common problems that we have observed when making a program with the API.

General -

Be sure to have the general order of commands as listed on the [Basic Structure](#) page in the Getting Started section.

It is recommended that you always stop the memory at the beginning of a new waveform definition.

When downloading data into the AWG, create a delay between the time you start downloading and the time you start the memory. This will ensure that all of the data that you have stored on the computer will be downloaded correctly to the AWG. For longer Memory Depth waveforms, this is especially important. The delay will be dependent on your system resources. You can experiment with the delay time by using the AWG GUI to download waveforms of various Memory Depths.

As already stated on the [data length](#) properties page, when you want to perform a duty cycle waveform and the total number of values is equal to the Memory Depth setting, you should turn off Data Length Enable and set the Data Length to the number of data points. If your total number of points is not equal the Memory Depth setting, then you must either use the [user_define_bulk](#) or the [user_define_file](#) functions.

Multi-Board -

Listed below is the recommended sequence of commands to put AWG into Master mode. These commands can be combined in a intermediate class member function.

```
awg.slave_mode = false;  
awg.auto_armed = true;  
awg.synci_enable = true;  
awg.synco_enable = true;
```

When you want to arm the Master AWG board to wait for the trigger signal, do not use the [restart](#) method (as you do in Free Run Mode). Instead use the [arm](#) method. The **arm** command will only arm the Master board, not the Slave board.

Listed below is the recommended sequence of commands to put AWG into Slave mode.

```
awg.slave_mode = true;  
awg.auto_armed = false;  
awg.synci_enable = false;
```

To arm the Slave board to wait for the SYNCI signal, use the [slave_mode](#) property, not the **arm** or **restart** methods.

For either Master or Slave mode, be sure to specify the loop count to a non-zero, non-negative integer. If you fail to do this then the board will use the last specified loop count setting (default is 0) in which case your board might be in Free Run mode.

We recommend that you stop the memory before arming either the Master or Slave boards. This has the effect of stabilizing the hardware.

Debugging -

You can have the GUI open when you are running your own program. The GUI can be very helpful when you need to debug your program. Make sure that you use the USB Reconnect command before using the GUI.

You can use the GUI to check the status of the AWG. You can also stop the AWG memory with it and check the data in memory.

All parser commands are sent to the AWG251.log file in your program's .exe folder.

The easiest way to see whether Data Length is enabled is to set the Data Length to half of the Memory Depth and the marker widths to half of the Data Length. If the marker has a half duty cycle, then Data Length is on (marker runs to half of the Data Length). But if the marker has a quarter duty cycle, then the AWG is not running up to the Data Length (marker is a quarter of Memory Depth).

Examples -

The Solution files for the example are zipped and included below.

Please note that you must re-reference the Euvis_Module_V1p0.dll API file for the examples. To do this, follow the instructions as detailed on the [API Reference](#) page.

[example.zip](#)

Basic Sine Wave -

This example will take you through some of the methods and properties of the API. If you are unsure of any of the methods or properties please refer to them in the detailed descriptions in the [Methods](#) and [Properties](#) sections.

For convenience here is the source code:

[main.cpp](#)

Object Instantiation

```
AWG_Group_API    awg_group;  
AWG_API          awg;
```

The above code creates an object of the *AWG_Group_API* class called "awg_group" and an object of the *AWG_API* class called "awg". When you want to access any of the methods or properties in the *AWG_Group_API* you must use the "." operator and the "awg_group" object name. For example if you wanted to read the AWG Series Number you would write the following:

```
int AWG_SN;  
AWG_SN = awg_group.number;
```

When you want to access any of the methods and properties in the *Common_API* class or the *AWG_API* class you must use the "." operator and the "awg" object name. For example if you wanted to set the waveform code you would write the following:

```
awg.code = 0;
```

Board Initialization

```
int SeriesNumber = awg_group.get_sn(0);
```

Gets the Series Number of the AWG board and stores it into the integer variable "SeriesNumber". The "0" argument is the AWG's index number. The example assumes that there is only one AWG board connected. When there is only one AWG board connected its index number will always be 0.

```
awg.ini( SeriesNumber );
```

Initializes the AWG board. This step is very important and must be done before sending any commands to the AWG. Note that the argument being passed into the function is the Series Number of the board and not the index number.

```
if( awg.module_id != MODULE_ID::AWG_452 )
{
    ...
}
```

The above code tests to see if the right board is connected. The code for the AWG can be found in the enumerated list located above the main function.

Board Setup

```
awg.signature_ini();
awg.Clock_Frequency = 2e9;
```

The **signature_ini()** function sets the board signature file while the **Clock_Frequency** sets the input clock frequency. The **Clock_Frequency** property of course should match with your real input frequency clock.

Waveform Setup

```
awg.memory_depth = 0x100000;
awg.loop_count = 0;
awg.code = WAVEFORM_CODE::WAVEFORM_SIN_AB;
awg.data_length = 0x2000;
awg.waveform_parameter_u0 = 1;
awg.waveform_parameter_u1 = 0x10;
awg.marker1_start = 0x10;
awg.marker1_width = 0x100;
awg.marker2_start = 0x200;
awg.marker2_width = 0x100;
awg.marker3_start = 0x100;
awg.marker3_width = 0x100;
```

All of the above code defines a simple sine wave that is 1/16th of the input clock. Please refer to the [Properties](#) section for a detailed description of each.

Download and Start Waveform

```
awg.download();
```

Downloads the waveform you specified to the hardware.

```
awg.stop();
awg.flush();
awg.restart();
```

All three lines above should be used everytime you want to run a downloaded waveform. The **stop** function stops all activity on the AWG board. The **flush** function resets the memory address to 0. The **restart** function sends the signal to run the AWG memory and start the waveform.

Appendix -

Jan 16, 2009

New API version, Euvis_module_V1p0.dll:

All Euvis modules share the same API, Euvis_module_V1p0.dll. To access the module-specific API for AWG, please use the namespace AWG_API. All the available methods and functions can be identified via Object Browser in the Visual Studio.