

Working with the SCDateTime Variables and Values

- [Introduction](#)
- [SCDateTime Variables](#)
- [SCDateTimeMS Variables](#)
- [Date Values](#)
- [Time Values](#)
- [Valid Ranges](#)
- [SCDateTime Member Functions](#)
 - [GetAsDouble\(\)](#)
 - [GetDate\(\)](#)
 - [GetDateTimeYMDHMS\(\)](#)
 - [GetDateTimeYMDHMS_MS\(\)](#)
 - [GetDateYMD\(\)](#)
 - [GetDay\(\)](#)
 - [GetDayOfWeek\(\)](#)
 - [GetTimeHMS\(\)](#)
 - [GetHour\(\)](#)
 - [GetMicroSecond\(\)](#)
 - [GetMillisecond\(\)](#)
 - [GetMinute\(\)](#)
 - [GetMonth\(\)](#)
 - [GetSecond\(\)](#)
 - [GetTimeInMilliseconds\(\)](#)
 - [GetTimeInSeconds\(\)](#)
 - [GetTimeAsSCDateTime\(\)](#)
 - [GetTimeInSecondsWithoutMilliseconds\(\)](#)
 - [GetYear\(\)](#)
 - [IsSaturday\(\)](#)
 - [IsSunday\(\)](#)
 - [IsWeekend\(\)](#)
 - [RoundDateTimeDownToMilliSecond\(\)](#)
 - [RoundDateTimeDownToMinute\(\)](#)
 - [RoundDateTimeDownToSecond\(\)](#)
 - [RoundToNearestMilliSecond\(\)](#)
 - [RoundToNearestSecond\(\)](#)
 - [SCDateTime\(\) Constructors](#)
 - [SetDate\(\)](#)
 - [SetDateTime\(\)](#)
 - [SetDateTimeYMDHMS\(\)](#)

- [SetDateTimeYMDHMS_MS\(\)](#)
- [SetDateYMD\(\)](#)
- [SetTime\(\)](#)
- [SetTimeHMS\(\)](#)
- [SetTimeHMS_MS\(\)](#)
- [DAYS\(\)](#)
- [YEARS\(\)](#)
- [HOURS\(\)](#)
- [MINUTES\(\)](#)
- [SECONDS\(\)](#)
- [MILLISECONDS\(\)](#)
- [operator+=\(\)](#)
- [operator-=\(\)](#)
- [Date and Time Functions](#)
 - [DaysInDateSpanNotIncludingWeekends\(\)](#)
- [Working With SCDatetime Arrays](#)
- [DateAt\(\)](#)
- [TimeAt\(\)](#)
- [SetDateAt\(\)](#)
- [SetTimeAt\(\)](#)
- [Using References](#)
- [Date and Time Math](#)

Introduction

The **SCDateTime** data type is used for Date-Time values in the ACSIL ([Advanced Custom Study Interface and Language](#)).

The internal representation of Date-Time values in this data type is exactly the same as the Date-Time values used in Microsoft Excel and Libre Office Calc.

There are various independent global functions and member functions of the **SCDateTime** type which makes it easy to work with these values. This page documents these functions.

SCDateTime Variables

A **SCDateTime** variable contains a Date and Time value as a 64-bit integer. The external interface of **SCDateTime**.

The [ACSIL](#) arrays [sc.BaseDateTimeln\[\]](#) and [sc.DateTimeOut\[\]](#) are of the SCDatetime type.

A SCDatetime variable can be locally defined within an ACSIL study function.

Refer to the examples given below.

Throughout the documentation and examples you will see the variable, **SCDateTimeVariable** being

used. These variables are of the **SCDateTime** type.

The following operators are supported with SCDateTime variables: = , += , -= , == , != , < , <= , > , >= .

A [SCDateTime](#) variable contains a number which represents a Date and Time. The Date is represented by the integer portion of this number and the Time is represented by the fractional portion of the number.

Example Code

```
SCDateTime MySCDateTime; //Locally defined SCDateTime variable.  
  
//MySCDateTime will contain, with this function call, the specified date time.  
MySCDateTime.SetDateTimeYMDHMS(2007, 1, 30, 16, 10, 0);  
  
int Hour, Minute, Second;  
sc.BaseDateTimeIn[.].GetTimeHMS(Hour, Minute, Second);  
  
sc.DateTimeOut[DestinationIndex] = sc.BaseDateTimeIn[SourceIndex];
```

SCDateTimeMS Variables

A **SCDateTimeMS** variable contains a Date and Time value as a 64-bit integer. It is identical to and supports the same member functions as [SCDateTime Variables](#).

SCDateTimeMS variables can be assigned to persistent variables by using the [sc.GetPersistentSCDateTime](#) and [sc.SetPersistentSCDateTime](#) functions.

As of version 2196, the **SCDateTime** and **SCDateTimeMS** Date-Time classes used within ACSIL and Sierra Chart are now exactly the same. SCDateTime now functions exactly like SCDateTimeMS where when comparisons are done, they are done to the microsecond rather than to the second. And there is no longer any internal rounding to the nearest second in **SCDateTime** for functions which would previously do this like **SCDateTime::GetTime**.

Instead the **GetTime** function will simply remove any microseconds component and just return number of seconds since midnight. Whereas previously **SCDateTime** would round to the nearest second.

Date Values

Date values are integer (**int**) values representing the number of days since December 30, 1899.

You can get a Date Value from a SCDateTime variable by using the [GetDate\(\)](#) member function.

You can set the date part of a SCDateTime variable by using the [SetDate\(\)](#) member function. Or by constructing a SCDateTime and specifying the date value for the first parameter and 0 for the second parameter (time value). Example: **SCDateTime DateVariable(DateValue, 0);**

You can construct a Date Value from Year, Month, Day components by using the [SetDateYMD\(\)](#) function on a SCDateTime variable, and deconstruct a Date Value using the [GetDateYMD\(\)](#) function on a SCDateTime variable.

Time Values

Time Values are integer (**int**) values representing the number of seconds since midnight (00:00).

You can get a Time Value from a `SCDateTime` variable by using the [GetTimeInSeconds\(\)](#) member function.

You can set the time part of a `SCDateTime` variable from an integer time value by using the [SetTime\(\)](#) member function. Or by constructing a `SCDateTime` and specifying 0 for the first parameter (date value) and the time value for the second parameter. Example: **SCDateTime DateVariable(0, 720);**. 720 means 720 seconds from midnight.

You can construct a Time Value from Hour, Minute, Second, Millisecond components by using the [SCDateTime::SetTimeHMS_MS\(\)](#) function, and get the individual Time Value components by using the [SCDateTime::GetTimeHMS\(\)](#) function.

To compare the time components of two different `SCDateTime` variables with precision to the second, get the time values from them by using the [GetTimeInSeconds\(\)](#) function and compare those time values.

The internal time value of an `SCDateTime` can represent milliseconds/microseconds. There are corresponding member functions for these.

Valid Ranges

The valid ranges for the Date and Time components in a `SCDateTime` variable are:

- **Year:** Four digit year. Example: 2012.
- **Month:** 1 through 12.
- **Day:** 1 through [days in month].
- **Hour:** 0 through 23.
- **Minute:** 0 through 59.
- **Second:** 0 through 59.
- **Millisecond:** 0 through 999. (Currently used only as a counter for trades within the same second. Does not represent actual milliseconds.)
- **Microsecond:** 0 through 999. (Not currently supported)

SCDateTime Member Functions

GetAsDouble()

Type: `SCDateTime` member function.

```
double GetAsDouble();
```

GetAsDouble() returns the internal Date-Time value as a double precision floating point value. This can be used to store the value externally.

This double precision floating point value can be passed to the `SCDateTime` constructor to create a new `SCDateTime` from it.

For a complete explanation of the Date component of this value, refer to [Date Value](#). The date value is the integer portion of the double. The fractional portion is the Time value which is represented as a fraction of one day where $1/86400000$ is 1 millisecond. 86400000 is the number of milliseconds in a day.

Example Code

```
const double DateTimeDouble = SCDateTimeVariable.GetAsDouble();
```

GetDate()

Type: `SCDateTime` member function.

```
int GetDate();
```

GetDate() returns the date part of the `SCDateTime` variable. The value returned is a [Date Value](#).

Example Code

```
int Date = SCDateTimeVariable.GetDate();
```

GetDateTimeYMDHMS()

Type: `SCDateTime` member function

```
void GetDateTimeYMDHMS(int& Year, int& Month, int& Day, int& Hour, int& Minute, int& Second);
```

GetDateTimeYMDHMS() gets the **Year**, **Month**, **Day**, **Hour**, **Minute**, and **Second** components of the `SCDateTime` variable. The function will set the **Year**, **Month**, **Day**, **Hour**, **Minute**, and **Second** variables provided as parameters.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Year, Month, Day, Hour, Minute, Second;  
  
SCDateTimeVariable.GetDateTimeYMDHMS(Year, Month, Day, Hour, Minute, Second);
```

GetDateTimeYMDHMS_MS()

Type: SCDatetime member function

```
void GetDateTimeYMDHMS_MS(int& Year, int& Month, int& Day, int& Hour, int& Minute, int& Second, int& MilliSecond);
```

GetDateTimeYMDHMS_MS() gets the year, month, day, hour, minute, second, and millisecond components of the SCDatetime variable. The function will set the **Year**, **Month**, **Day**, **Hour**, **Minute**, **Second**, and **MilliSecond** variables provided as parameters.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Year, Month, Day, Hour, Minute, Second, MilliSecond;  
SCDateTimeVariable.GetDateTimeYMDHMS_MS(Year, Month, Day, Hour, Minute, Second, MilliSecond);
```

GetDateYMD()

Type: SCDatetime member function

```
void GetDateYMD(int& Year, int& Month, int& Day);
```

GetDateYMD() gets the year, month, and day components of the SCDatetime variable.

The function will set the **Year**, **Month**, and **Day** variables provided as parameters.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Year = 0;  
int Month = 0;  
int Day = 0;  
SCDateTimeVariable.GetDateYMD(Year, Month, Day);
```

GetDay()

Type: SCDatetime member function

```
int GetDay();
```

GetDay() returns an integer value representing the day of the SCDatetime variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Day = sc.BaseDateTimeIn[sc.Index].GetDay();
```

GetDayOfWeek()

Type: SCDatetime member function

```
int GetDayOfWeek();
```

The **GetDayOfWeek()** function returns the day of the week for the SCDatetime variable.

The return value will be one of the following:

- SUNDAY
- MONDAY
- TUESDAY
- WEDNESDAY
- THURSDAY
- FRIDAY
- SATURDAY

Example Code

```
if (SCDateTimeVariable.GetDayOfWeek() == MONDAY)
{
    // SCDateTimeVariable is a Monday
}
```

GetTimeHMS()

Type: SCDatetime member function

```
int GetTimeHMS(int& Hour, int& Minute, int& Second);
```

sc.GetTimeHMS() gets the **Hour**, **Minute**, and **Second** components of the internal [Time Value](#) of the SCDatetime variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Hour = 0;
int Minute = 0;
int Second = 0;

sc.GetTimeHMS(Hour, Minute, Second);
```

GetHour()

Type: SCDatetime member function

```
int GetHour();
```

GetHour() returns an integer value representing the hour of the SCDatetime variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Hour = sc.BaseDateTimeIn[sc.Index].GetHour();
```

GetMicroSecond()

Type: SCDatetime member function

```
int GetMicroSecond();
```

GetMicroSecond() returns an integer value representing the microsecond of the SCDatetime variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int MicroSecond = sc.BaseDateTimeIn[sc.Index].GetMicroSecond();
```

GetMillisecond()

Type: SCDatetime member function

```
int GetMillisecond();
```

GetMillisecond() returns the milliseconds portion of the Date-Time variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int MilliSecond = sc.BaseDateTimeIn[sc.Index].GetMillisecond();
```

GetMinute()

Type: SCDatetime member function

```
int GetMinute();
```

GetMinute() returns an integer value representing the minute of the SCDatetime variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Minute = sc.BaseDateTimeIn[sc.Index].GetMinute();
```

GetMonth()

Type: SCDatetime member function

```
int GetMonth();
```

GetMonth() returns an integer value representing the month of the SCDatetime variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Month = sc.BaseDateTimeIn[sc.Index].GetMonth();
```

GetSecond()

Type: SCDatetime member function

```
int GetSecond();
```

GetSecond() returns an integer value representing the second of the SCDatetime variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Second = sc.BaseDateTimeIn[sc.Index].GetSecond();
```

GetTimeInMilliseconds()

Type: SCDatetime member function

```
int GetTimeInMilliseconds();
```

GetTimeInMilliseconds() returns the time part of the SCDatetime variable in milliseconds since midnight.

The internal time value is rounded to the nearest millisecond if it contains microseconds.

Example Code

```
int TimeInMilliseconds = SCDatetimeVariable.GetTimeInMilliseconds();
```

GetTimeInSeconds()

Type: SCDatetime member function

```
int GetTimeInSeconds();
```

GetTimeInSeconds() returns the time part of the SCDatetime variable in seconds. The value returned is a [Time Value](#).

Example Code

```
int TimeInSeconds = SCDatetimeVariable.GetTimeInSeconds();
```

GetTimeAsSCDateTime()

Type: SCDatetime member function

```
SCDateTime GetTimeAsSCDateTime();
```

GetTimeAsSCDateTime() returns the time part of the SCDatetime variable. The value returned is a [SCDateTime variable](#).

Example Code

```
SCDateTime TimeOnly = SCDatetimeVariable.GetTimeAsSCDateTime();
```

GetTimeInSecondsWithoutMilliseconds()

Type: SCDatetime member function

```
int GetTimeInSecondsWithoutMilliseconds();
```

GetTimeInSecondsWithoutMilliseconds() returns the time part of the SCDatetime variable without the milliseconds part. The value returned is a [Time Value](#).

The millisecond/microsecond component is discarded so that the value is truncated to the containing second.

Example Code

```
int Time = SCDatetimeVariable.GetTimeInSecondsWithoutMilliseconds();
```

GetYear()

Type: SCDatetime member function

```
int GetYear();
```

GetYear() returns an integer value representing the year of the SCDatetime variable.

Refer to the [Valid Ranges](#) section for the range of values returned.

Example Code

```
int Year = sc.BaseDateTimeIn[sc.Index].GetYear();
```

IsSaturday()

Type: SCDatetime member function

```
int IsSaturday();
```

IsSaturday() returns an boolean value (true or false) depending on whether the day of the week for the SCDatetime variable is **SATURDAY**.

IsSunday()

Type: SCDatetime member function

```
int IsSunday();
```

IsSunday() returns an boolean value (true or false) depending on whether the day of the week for the SCDatetime variable is **SUNDAY**.

IsWeekend()

Type: SCDatetime member function

```
int IsWeekend();
```

IsWeekend() returns an boolean value (true or false) depending on whether the day of the week for the SCDatetime variable is a weekend day (**SATURDAY** or **SUNDAY**).

RoundDateTimeDownToMilliSecond()

Type: SCDatetime member function

```
void RoundDateTimeDownToMilliSecond();
```

The **RoundDateTimeDownToMilliSecond()** function rounds down the Date-Time value contained within the SCDatetime variable to the millisecond and removes the microseconds. Therefore, the microseconds will be zero.

Example Code

```
SCDateTime BarDateTime = sc.BaseDateTimeIn[sc.Index];  
BarDateTime.RoundDateTimeDownToMilliSecond();
```

RoundDateTimeDownToMinute()

Type: SCDatetime member function

```
void RoundDateTimeDownToMinute();
```

The **RoundDateTimeDownToMinute()** function rounds down the Date-Time value contained within the SCDatetime variable to the minute and removes the seconds. Therefore, the seconds will be zero.

Example Code

```
SCDateTime BarDateTime = sc.BaseDateTimeIn[sc.Index];  
BarDateTime.RoundDateTimeDownToMinute();
```

RoundDateTimeDownToSecond()

Type: SCDatetime member function

```
void RoundDateTimeDownToSecond();
```

The **RoundDateTimeDownToSecond()** function rounds down the Date-Time value contained within the SCDatetime variable to the second and removes the milliseconds. Therefore, the milliseconds will be zero.

Example Code

```
SCDateTime BarDateTime = sc.BaseDateTimeIn[sc.Index];  
BarDateTime.RoundDateTimeDownToSecond();
```

RoundToNearestMilliSecond()

Type: SCDatetime member function

```
void RoundToNearestMilliSecond();
```

The **RoundToNearestMilliSecond()** function rounds the Date-Time value contained within the SCDatetime variable to the nearest millisecond and removes the microseconds. Therefore, the microseconds will be zero.

Example Code

```
SCDateTime BarDateTime = sc.BaseDateTimeIn[sc.Index];  
BarDateTime.RoundToNearestMilliSecond();
```

RoundToNearestSecond()

Type: SCDatetime member function

```
void RoundToNearestSecond();
```

The **RoundToNearestSecond()** function rounds the Date-Time value contained within the SCDatetime variable to the nearest second and removes the milliseconds. Therefore, the milliseconds will be zero.

Example Code

```
SCDateTime BarDateTime = sc.BaseDateTimeIn[sc.Index];  
BarDateTime.RoundToNearestSecond();
```

SCDateTime() Constructors

Type: SCDatetime constructor.

```
SCDateTime();
```

```
SCDateTime(double DateTime );
```

```
SCDateTime(const SCDateTime & DateTime);
```

```
SCDateTime(int DateValue, int TimeValue);
```

```
SCDateTime(int Hour, int Minute, int Second, int Millisecond);
```

```
SCDateTime(int Year, int Month, int Day, int Hour, int Minute, int Second);
```

SCDateTime() is the constructor function that constructs and initializes a SCDatetime variable. Using the different constructors, it is supported to assign the Year, Month, Day, Hour, Minute,

Second, Millisecond values to the SCDatetime variable. Or a [Date Value](#) or [Time Value](#).

Refer to the [Valid Ranges](#) section for valid values that can be used for the parameters.

Example Code

```
SCDateTime DateTime(2017, 1, 30, 0, 0, 0);  
SCDateTime DateTime(2012, 1, 1, 12, 0, 0);
```

SetDate()

Type: SCDatetime member function

```
SCDateTime& SetDate(int Date);
```

SetDate() sets the date part of the SCDatetime variable with the given **Date**. **Date** must be given as a [Date Value](#).

The existing time portion of the SCDatetime variable is preserved when using the **SetDate()**function.

Example Code

```
SCDateTime SCDatetimeVariable;  
SCDateTimeVariable.SetDate(sc.BaseDateTimeIn[sc.Index].GetDate());
```

SetDateTime()

Type: SCDatetime member function

```
int SetDateTime(int Date, int Time);
```

SetDateTime() sets the SCDatetime variable with the given **Date** and **Time** components. **Date** must be given as a [Date Value](#), and **Time** must be given as a [Time Value](#).

Example Code

```
r_StopDateTime.SetDateTime(sc.BaseDateTimeIn[sc.Index].GetDate(), Input_StopTime.GetTime());
```

SetDateTimeYMDHMS()

Type: SCDatetime member function

```
SCDateTime& SetDateTimeYMDHMS(int Year, int Month, int Day, int Hour, int Minute, int
```

Second);

SetDateTimeYMDHMS() sets the SCDatetime variable with the given **Year**, **Month**, **Day**, **Hour**, **Minute**, and **Second** components.

Refer to the [Valid Ranges](#) section for valid values that can be used.

Example Code

```
SCDateTimeVariable.SetDateTimeYMDHMS(2007, 1, 30, 16, 10, 0);
```

SetDateTimeYMDHMS_MS()

Type: SCDatetime member function

SCDateTime& **SetDateTimeYMDHMS_MS**(int **Year**, int **Month**, int **Day**, int **Hour**, int **Minute**, int **Second**, int **MilliSecond**);

SetDateTimeYMDHMS_MS() sets the SCDatetime variable with the given **Year**, **Month**, **Day**, **Hour**, **Minute**, **Second**, and **MilliSecond** components.

Refer to the [Valid Ranges](#) section for valid values that can be used.

Example Code

```
SCDateTimeVariable.SetDateTimeYMDHMS_MS(2007, 1, 30, 16, 10, 0, 0);
```

SetDateYMD()

Type: SCDatetime member function

SCDateTime& **SetDateYMD**(int **Year**, int **Month**, int **Day**);

SetDateYMD() sets the date part of the SCDatetime variable with the given **Year**, **Month**, and **Day** components.

Refer to the [Valid Ranges](#) section for valid values that can be used.

Example Code

```
SCDateTimeVariable.SetDateYMD(2007, 1, 30);
```

SetTime()

Type: SCDatetime member function

```
SCDateTime& SetTime(int Time);
```

The **SetTime()** function sets the time part, in seconds, of the SCDateTime variable with the given **Time**. **Time** must be given as a [Time Value](#).

The existing date portion of the SCDateTime variable is preserved when using the **SetTime()** function.

Example Code

```
SCDateTime TimeValue(16, 14, 59, 0); //Construct a time value  
SCDateTimeVariable.SetTime(TimeValue);
```

SetTimeHMS()

Type: SCDateTime member function

```
SCDateTime& SetTimeHMS(int Hour, int Minute, int Second);
```

SetTimeHMS() sets the time part of the SCDateTime variable with the given **Hour**, **Minute**, and **Second** components. The existing date in the variable is not changed and left as is.

Refer to the [Valid Ranges](#) section for valid values that can be used.

Example Code

```
SCDateTimeVariable.SetTimeHMS(16, 10, 0);
```

SetTimeHMS_MS()

Type: SCDateTime member function

```
SCDateTime& SetTimeHMS_MS(int Hour, int Minute, int Second, int MilliSecond);
```

SetTimeHMS_MS() sets the time part of the SCDateTime variable with the given **Hour**, **Minute**, **Second**, and **MilliSecond** components. The existing date in the variable is not changed and left as is.

Refer to the [Valid Ranges](#) section for valid values that can be used.

Example Code

```
SCDateTimeVariable.SetTimeHMS_MS(16, 10, 0, 0);
```


DAYS()

Type: SCDatetime static member function

```
static SCDatetime& DAYS(int Days);
```

The **DAYS()** static member function constructs and returns an SCDatetime variable containing the number of days according to the specified number of days.

Example Code

```
SCDatetime DateTimeVariable;  
DateTimeVariable += SCDatetime::DAYS(5);
```

YEARS()

Type: SCDatetime static member function

```
static SCDatetime& YEARS(int Years);
```

The **YEARS()** static member function constructs and returns an SCDatetime variable containing the number of years according to the specified number of years.

Example Code

```
SCDatetime DateTimeVariable;  
DateTimeVariable += SCDatetime::YEARS(5);
```

HOURS()

Type: SCDatetime static member function

```
static SCDatetime& HOURS(int Hours);
```

The **HOURS()** static member function constructs and returns an SCDatetime variable containing the number of hours according to the specified number of hours.

Example Code

```
SCDatetime DateTimeVariable;  
DateTimeVariable += SCDatetime::HOURS(5);
```

MINUTES()

Type: SCDatetime static member function

```
static SCDatetime& MINUTES(int Minutes);
```

The **MINUTES()** static member function constructs and returns an SCDatetime variable containing the number of minutes according to the specified number of minutes.

Example Code

```
SCDatetime DateTimeVariable;  
DateTimeVariable += SCDatetime::MINUTES(5);
```

SECONDS()

Type: SCDatetime static member function

```
static SCDatetime& SECONDS(int Seconds);
```

The **SECONDS()** static member function constructs and returns an SCDatetime variable containing the number of seconds according to the specified number of seconds.

Example Code

```
SCDatetime DateTimeVariable;  
DateTimeVariable += SCDatetime::SECONDS(5);
```

MILLISECONDS()

Type: SCDatetime static member function

```
static SCDatetime& MILLISECONDS(int Milliseconds);
```

The **MILLISECONDS()** static member function constructs and returns an SCDatetime variable containing the number of milliseconds according to the specified number of milliseconds.

Example Code

```
SCDatetime DateTimeVariable;  
DateTimeVariable += SCDatetime::MILLISECONDS(5);
```

operator+=()

Type: SCDatetime member operator

```
const SCDatetime& +=(const SCDatetime& DateTime);
```

The **+=** member operator adds the given **DateTime** parameter to the existing Date-Time value in

the `SCDateTime` object it is called in relation to.

The given **DateTime** parameter usually will represent a certain amount of time, like for example 1 hour, and not represent a particular absolute Date-Time.

Example Code

```
SCDateTime DateTimeVariable;  
DateTimeVariable += SCDateTime::YEARS(5);
```

operator-=(())

Type: `SCDateTime` member operator

```
const SCDateTime& -=(const SCDateTime& DateTime);
```

The **+=** member operator subtracts the given **DateTime** parameter to the existing Date-Time value in the `SCDateTime` object it is called in relation to.

The given **DateTime** parameter usually will represent a certain amount of time, like for example 1 hour, and not represent a particular absolute Date-Time.

Example Code

```
SCDateTime Time(1, 10, 0, 0); //1 Hour and 10 Minutes.  
SCDateTime DateTimeVariable;  
DateTimeVariable -= Time;
```

Date and Time Functions

DaysInDateSpanNotIncludingWeekends()

Type: Function

```
int DaysInDateSpanNotIncludingWeekends(int FirstDate, int LastDate)
```

The **DaysInDateSpanNotIncludingWeekends()** function calculates the number of days within the date span specified by the **FirstDate** and the **LastDate** parameters, not including Saturday and Sunday.

The calculation includes both of these dates as well. Both of these parameters are [Date Values](#).

Example Code

```
int NumberOfDays = DaysInDateSpanNotIncludingWeekends(sc.BaseDateTimeln[sc.Index - 100].GetDate()
```

Working With SCDatetime Arrays

When working with the [sc.BaseDateTimeln\[\]](#) array, you can use different methods for getting the date and/or time parts from elements in the array. These methods are given below.

Using the GetDate() and GetTimelnSeconds() member functions of a SCDatetime variable

```
// Get the date of the bar at the current index
int CurrentBarDate = sc.BaseDateTimeln[sc.Index].GetDate();

// Get the time of the bar at the current index
int CurrentBarTime = sc.BaseDateTimeln[sc.Index].GetTimelnSeconds();
```

Using the DateAt() and TimeAt() member functions of a SCDatetime array

```
// Get the date of the bar at the current index
int CurrentBarDate = sc.BaseDateTimeln.DateAt(sc.Index);

// Get the time of the bar at the current index
int CurrentBarTime = sc.BaseDateTimeln.TimeAt(sc.Index);
```

Using the [] array operator on the sc.BaseDateTimeln SCDatetime array to get the Date and Time

```
if (sc.BaseDateTimeln[sc.Index] - sc.BaseDateTimeln[sc.Index - 1] > SCDatetime::MINUTES(1))
{
    // The current bar being processed has a Date-Time which is more than one minute later than the prior bar Date-Time
}
```

DateAt()

Type: SCDatetimeArray member function

```
int DateAt(int Index);
```

DateAt() returns the date part of the SCDatetime variable at the given **Index** in the SCDatetime array. The value returned is a [Date Value](#).

Example Code

```
// Get the date at the current index  
int Date = sc.BaseDateTimeIn.DateAt(sc.Index);
```

TimeAt()

Type: SCDatetimeArray member function

```
int TimeAt(int Index);
```

TimeAt() returns the time part of the SCDatetime variable at the given **Index** in the SCDatetime array. The value returned is a [Time Value](#).

Example Code

```
// Get the time at the current index  
int Time = sc.BaseDateTimeIn.TimeAt(sc.Index);
```

SetDateAt()

Type: SCDatetimeArray member function

```
int SetDateAt(int Index, int Date);
```

SetDateAt() sets the date part of the DateTime variable at the given **Index** in the SCDatetime array. **Date** must be given as a [Date Value](#). The value returned is the same value that is passed in for **Date**. This function must not be used on the [sc.BaseDateTimeIn\[\]](#) array. The only array you'll probably ever use this on is the [sc.DateTimeOut\[\]](#) array.

Example Code

```
// Set the date for a custom chart bar  
int Date = sc.DateTimeOut.SetDateAt(CustomIndex,sc.BaseDateTimeIn[sc.Index].GetDate());
```

SetTimeAt()

Type: SCDatetimeArray member function

```
int SetTimeAt(int Index, int Time);
```

SetTimeAt() sets the time part of the DateTime variable at the given **Index** in the SCDatetime array. **Time** must be given as a [Time Value](#). The value returned is the same value that is passed in for **Time**. This function must not be used on the [sc.BaseDateTimeIn\[\]](#) array. The only array you will need to use this on is the [sc.DateTimeOut\[\]](#) array.

Example Code

```
// Set the time for a custom chart bar  
int Time = sc.DateTimeOut.SetTimeAt(CustomIndex,sc.BaseDateTimeIn[sc.Index].GetTimeInSeconds());
```

Using References

The `SCDateTime` reference type, **`SCDateTimeArrayRef`**, can be used to set a reference to a **`SCDateTimeArray`** array to simplify writing code. See the example below.

Example Code

```
SCDateTimeArrayRef DateTimes = sc.BaseDateTimeIn;  
int Time;  
Time = DateTimes[sc.Index].GetTimeInSeconds();
```

Date and Time Math

[SCDateTime Variable Documentation](#)

Adding 30 seconds to a `SCDateTime` variable

```
SCDateTimeVariable.AddSeconds(30);
```

Subtracting 5 minutes from a `SCDateTime` variable

```
SCDateTimeVariable.SubtractMinutes(5);
```

Adding 1 hour to a `SCDateTime` variable

```
SCDateTimeVariable.AddHours(1);
```

Adding 1 day to a `SCDateTime` variable

```
SCDateTimeVariable.AddDays(1);
```

Adding 1 week to a SCDatetime variable

```
SCDateTimeVariable.AddDays(DAYS_PER_WEEK);
```

Adding 1 year to a SCDatetime variable

```
SCDateTimeVariable.AddYears(1);
```

*Last modified Wednesday, 23rd August, 2023.