HCC API Documentation

V1.1 (2018-08-28)

www.tripleheinz.com/index.php?page=mtht

HCC API Reference

The HCC API is a series of declarations, conventions and specifications created to promote a common interface to work with MetaTrader history files. The HCC Library is the "first 3rd party" implementation of the HCC API, being the official one considered the private work from MetaQuotes (who created the file format) even if both foundations do not entirely comply between each other. The HCC Library has been conceived as a C DLL using the D programming language v2; therefore functions will be explained using the D syntax which is common to the C family languages.

Programmers are free to create their own implementations of the HCC API matching the described conventions or use the HCC Library in their own programs using this document as a base guide and reference.

Index

HCC Types

The following table describes the types used by the HCC library, their size in memory and what they are used for:

Type	Size	Usage
ReturnCode	Unsigned 32 bits integer	Success or failure of HCC functions. Returned by most HCC functions.
HCCHandle	Unsigned 32 bits integer	Represents an opened HCC file. Used by other HCC functions.
ubyte	Unsigned 8 bits integer	Used for certain byte stream lengths.
ushort	Unsigned 16 bits integer	Used to represent a language ID.
uint	Unsigned 32 bits integer	Type for certain rates members, indexes, ranges, amounts and counts.
double	64 bits floating point	Used for prices.
bool	Unsigned 8 bits integer	Boolean value. Holds “true” or “false”. Used in certain parsing options.
wchar*	Size of a wide char pointer	Null terminated string in UTF-16LE encoding. Used for any string such as file paths, error messages and symbols.
ubyte* ubyte[] ubyte[n]	Size of byte pointer or ‘n’ bytes for static array	Byte stream. It is used internally in all three forms: As a pointer, as a dynamic array and as a static array. Used for parsing separator.

HCC Constants

Type	ID	Value	Description
ReturnCode	HCC_SUCCESS	0	Success of HCC functions. Returned by most HCC functions.

HCC Structures

In the HCC Library, the fields of a structure have no alignment. Members are packed together to match the behaviour in MQL5.

HCCRates		
Primary structure used to define rates in OHLC format plus some extra fields.		
Members		
Type	ID	Description
uint	time	Start date and time of the period. The type is HCC specific and is defined as an unsigned 32 bits arithmetic type that represents the number of seconds elapsed since 1970.01.01 00:00, expressed as UTC time (GMT timezone = 0). MetaTrader uses the same format but stores the timestamp in a 64 bits integral type (datetime) for a broader range or greater accuracy. The HCC type uses 32 bits in memory but can be perfectly promoted to a datetime type.
double	open	Open price.
double	high	Highest price.
double	low	Lowest price.
double	close	Close price.
uint	volume	Tick volume.
uint	spread	Spread.

HCCParsingOptions		
Structure used to pass parsing options to **HCCSetParsingOptions** function.		
Members		
Type	ID	Description
HCC_RATES_SEPARATOR	separator	Specifies the separator type. It can be one of the officially defined ones or a custom one. Defaults to “Standard“.
ubyte[8]	custom_separator	When the separator type is custom then this field is used to specify the byte stream (the separator itself). It is defined as a static array for convenience. It can also be defined by other languages as a dynamic array. Internally it is accessed as a pointer so it can also be defined as such.
ubyte	custom_length	When the separator type is custom then this field specifies the length of the byte stream in *custom_separator*. Valid lengths are in the range >=1 and <=8.
bool	fast_parse	Experimental. Enables optimizations for faster reading speeds in the parsing algorithm. It defaults to “false“ (disabled) because the inconsistent packing in HCC files breaks aligned reads and the algorithm has to fallback to standard routines quite often, resulting some times in equal or worse performance. Try and see which configuration performs better with your files.
bool	halt_corrupt	This option instruct the parsing algorithm (affecting read and write functions) to halt its execution when a corrupt rates structure is detected. It is disabled by default for flexibility but you can enable it if you want to detect suspicious rates in your files or you want a more conservative approach and maintain the file integrity. Keep it disabled to work naturally and freely with files (recommended).
bool	ignore_corrupt	This option is only used when *halt_corrupt* is false. If a corrupt rates structure is detected then you can instruct the parsing algorithm to ignore it and continue processing the next block of data. Enabling this option can affect the rates generated for periods greater than 1 minute and can result in missing rates while working in M1 period. Disabling it will allow you to work with all rates available and will allow you to fix corrupted rates. Keep it disabled to work naturally and freely with files (recommended).

HCC Enumerations

The underlying type for enums is **int** (signed 32 bits integer). A cast to **uint** (unsigned 32 bits integer) may be required by some languages when using these enums with HCC functions.

HCC_RATES_PROPERTY		
Used to get members from rates structure using **HCCGetRatesDouble** and **HCCGetRatesInteger** functions.		
Values		
ID	Value	Description
Time	0	
Open	1	
High	2	
Low	3	
Close	4	
Volume	5	
Spread	6	

HCC_TIMEFRAME		
Timeframes expressed as minutes. Used by time computing functions.		
Values		
ID	Value	Description
M1	1	1 Minute.
M2	2	2 Minutes.
M3	3	3 Minutes.
M4	4	4 Minutes.
M5	5	5 Minutes.
M6	6	6 Minutes.
M10	10	10 Minutes.
M12	12	12 Minutes.
M15	15	15 Minutes.
M20	20	20 Minutes.
M30	30	30 Minutes.
H1	60	1 Hour.
H2	120	2 Hours.
H3	180	3 Hours.
H4	240	4 Hours.
H6	360	6 Hours.
H8	480	8 Hours.
H12	720	12 Hours.
D1	1440	1 Day.
W1	10080	1 Week.
MN28	40320	1 Month containing 28 days.
MN29	41760	1 Month containing 29 days.
MN30	43200	1 Month containing 30 days.
MN31	44640	1 Month containing 31 days.
MN	MN30	Standard Month.
TM	MN * 3	1 Trimester.
SM	MN * 6	1 Semester.
Y1	525600	1 Year.
LY1	527040	1 Leap year.

Remarks: You can safely use periods greater than a year in HCC functions but they do not make sense because HCC files store rates in one file per year.

HCC_RATES_SEPARATOR		
Used to specify the type of separator for the rates parsing algorithm. Used by **HCCSetParsingOptions** function.		
Values		
ID	Value	Description
None	0	No separator. This means that rates are stored adjacently using a fixed size structure with a known file length.
Standard	1	Standard separator. (default)
Custom	0x7FFFFFFF	User-defined separator.

Remarks: If you have a known separator that you would like to include in the official enumeration then you can send me the information and I will add it to the documentation if it qualifies. I will ask you for information such as: Broker name, platform, country, year (of the rates data), symbol, the separator stream itself (if known)…anyway any information that can be used to identify and register the separator with a unique entry, oh and the HCC file in question for testing and validation. Usually, an HCC file that can't be parsed correctly is a sign that it may use a non-standard separator.

Return Code Table

HCC functions return a code (ReturnCode type) to notify either success or failure. These codes can be brought to human readable strings to know their meaning using the **HCCGetErrorMessage** function. In current version the default and only language is English but more languages might be added in future releases.

Code	Description
0	Success.
1	Unknown error.
2	Could not retrieve error description. *Please note that this description is naturally not retrievable using the **HCCGetErrorMessage** function.*
10000	The specified file doesn't exist.
10001	Couldn't open the specified file.
10002	The specified file is not readable.
10003	The specified file is not writeable.
10004	The specified file is not seekable.
10005	The specified file doesn't have a valid header.
10006	Read error.
10007	Write error.
10008	Seek error.
10009	The end of file has been reached.
10010	Invalid history file.
10011	Invalid file handle.
10012	Invalid index.
10013	Invalid property.
10014	Invalid separator.
10015	Invalid decimal mark.
10016	Invalid CSV parameters. Separator might have character(s) in common with decimal mark.
10017	History has not been previously loaded.
10018	History cache is empty.
10019	Invalid period.
10020	Corrupt rates detected. Either the time, low and high values are invalid.
10021	Invalid parsing option.
10022	Invalid transaction call. Could not begin or commit a transaction.
10023	Invalid transaction operation.

HCC Functions

The HCC API works by storing the requested rates in an internal buffer. Once the rates are loaded into this buffer then all the other functions can be called to operate on rates. Rates can be reloaded into the buffer with different parameters, discarding the contents of the previously filled buffer.

A typical program workflow would be like:

1. Open an HCC file to obtain an HCC handle with a call to **HCCOpenFile**.
2. At this point you may optionally call HCC handle specific functions such as **HCCGetSymbol** and **HCCGetPrecision**.
3. Load history into the internal buffer with a call to **HCCLoadHistory**.
4. Now that history is loaded into memory you may optionally call history specific functions to retrieve general information about the stored data such as **HCCGetHistoryLength** and **HCCGetHistoryPeriod**.
5. Now, here is what you want: Working with rates. To do that you can directly get each rates structure with a call to **HCCGetRates**. Alternatively you can get individual properties with a call to **HCCGetRatesDouble** and **HCCGetRatesInteger**.
6. If you are in need of exporting your memory loaded history to a CSV file then you can do that with a call to **HCCExportHistoryCSV**.
7. Close the HCC handle with a call to **HCCCloseFile**.

void **HCCInitialize**(void* gc)			
Perform library initialization routines.			
Parameters			
Type	ID	Direction	Description
void*	gc	IN	Handle of void pointer type to the garbage collector. This parameter can be safely set to NULL if you don't know what to do with it. Setting this parameter to NULL will instruct the HCC library to use its own integrated garbage collector.
Remarks			
This function is available only for the D programming language and is totally optional. If you wish to set the garbage collector then make sure to always call this function before calling any other function of the HCC API. The handle to the garbage collector is absolutely optional but the idea of specifying it is for system resources optimization purposes: It is better to have only one garbage collector running in the background than two of them performing the same operations.			

void **HCCTerminate**()
Perform library termination routines (module destructors, garbage collection, etc).
Remarks
This function is available only for the D programming language and is totally optional. If you previously called the function **HCCInitialize** then make sure to always call this function before your program ends.

ReturnCode **HCCGetErrorMessage**(ReturnCode code, ushort language, wchar* message)			
Gets verbose description for the specified code.			
Parameters			
Type	ID	Direction	Description
ReturnCode	code	IN	Code of the message you want to get.
ushort	language	IN	Language ID of the message. Pass 0 for the default language.
wchar*	message	OUT	Message associated for the code. Memory for the string must be previously allocated and the space must be long enough to hold the returned string including the terminating null character.
Remarks			
There is no retrievable message for error code #2; in that case it will return the same code. This is one of the very few functions that cannot take advantage of the automatic memory management and require preallocated memory due to the work with vector data. It has been implemented this way to maintain compatibility with languages that do not have direct access to pointers.			

ReturnCode **HCCOpenFile**(const(wchar)* path, out HCCHandle file)			
Open history file for further operation.			
Parameters			
Type	ID	Direction	Description
const(wchar)*	path	IN	Path to the history file. It can be an absolute path (C:\data\myfile.hcc) or a path relative to the current working directory (..\data\myfile.hcc).
HCCHandle	file	OUT	Handle to the opened file. You should keep a reference to this handle because it is used by most HCC functions.

ReturnCode **HCCCloseFile**(HCCHandle file)			
Close history file pointed by the specified handle.			
Parameters			
Type	ID	Direction	Description
HCCHandle	file	IN	File handle to close. Using this handle with any HCC function after it is closed will return error code #10011. The handle must be a valid one created by the **HCCOpenFile** function.
Remarks			
This function does not reset the value of the handle to any other value. The handle becomes invalid after calling this function, even if the value held is non zero.			

ReturnCode **HCCGetSymbol**(HCCHandle file, wchar* symbol)			
Get the symbol associated with the specified history file.			
Parameters			
Type	ID	Direction	Description
HCCHandle	file	IN	File handle whose symbol you want to get.
wchar*	symbol	OUT	The symbol of the history file. Memory for the string must be previously allocated and the space must be long enough to hold the returned string including the terminating null character.
Remarks			
This is one of the very few functions that cannot take advantage of the automatic memory management and require preallocated memory due to the work with vector data. It has been implemented this way to maintain compatibility with languages that do not have direct access to pointers.			

ReturnCode **HCCGetPrecision**(HCCHandle file, out uint precision)

Get symbol decimal precision (number of digits after the decimal mark).

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	File handle.
uint	precision	OUT	Precision value.

Remarks

The precision value is most often used for text formatting purposes.

ReturnCode **HCCSetParsingOptions**(HCCHandle file, HCCParsingOptions options)

Sets all the options for the rates parsing algorithm. This affects how rates are detected by the HCC functions.

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	File handle.
HCCParsingOptions	options	IN	Parsing options.

Remarks

The options defined are persistent between function calls until the next call to **HCCSetParsingOptions** which overrides the options to the ones of the latest call.
Read and write functions are affected by the options set by this function.

The default options are set for a flexible and natural work with history files. You can alternatively enable a strict mode to maintain file integrity by setting the option *halt_corrupt* to true.

ReturnCode **HCCLoadHistory**(HCCHandle file, uint period, uint start, uint how_many, out uint load_count)

Load rates from the specified file handle to an internal buffer in system memory using specific parameters.

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	Source history file.
uint	period	IN	Timeframe of history. Can be any value >0 or one of the HCC predefined ones (HCC_TIMEFRAME). This parameter is still experimental, you should pass 1.
uint	start	IN	Zero based index of the first element.
uint	how_many	IN	Specify how many rates to load. Zero means all available.
uint	load_count	OUT	Return the number of rates loaded.

Remarks

This function can be called as many times as you want. With each call the internal buffer is cleared and the new rates are stored.
To be memory efficient, instead of loading the whole history (*start* = 0 and *how_many* = 0) you can load history in small chunks by making sequential calls and increasing the *start* value to the number of rates loaded already. Variable *load_count* is very helpful for this purpose. The HCC Library is very quick and efficient for this kind of sequential load.
In case of reaching the end of file, the function will return **HCC_SUCCESS** and *load_count* will return 0.

Rates are created and loaded based on the specified period. A file may contain thousand of rates but this doesn't mean there are as much as needed to create rates in the specified timeframe. This could result in history having less rates than the source file or having no rates at all.

ReturnCode **HCCGetHistoryLength**(HCCHandle file, out uint length)			
Get the number of rates in the internal history buffer.			
Parameters			
Type	ID	Direction	Description
HCCHandle	file	IN	File handle associated with the history buffer.
uint	length	OUT	Number of rates in previously loaded history.
Remarks			
This function is useful in case you did not hold a reference to the value of *load_count* returned by the **HCCLoadHistory** function, as it returns the same value. History must have been loaded first using **HCCLoadHistory** or else error code #10017 will be returned. History length may be 0 if the history file has no rates available. If that is the case then the function will return **HCC_SUCCESS** and *load_count* will return 0.			

ReturnCode **HCCGetHistoryPeriod**(HCCHandle file, out uint period)			
Get the timeframe of the currently loaded history.			
Parameters			
Type	ID	Direction	Description
HCCHandle	file	IN	File handle associated with the history buffer.
uint	period	OUT	Current timeframe.
Remarks			
History must have been loaded first using **HCCLoadHistory** or else error code #10017 will be returned. If history was previously loaded but there were no rates available then this function will return **HCC_SUCCESS** and *period* will return the timeframe used when history was loaded.			

ReturnCode **HCCGetRates**(HCCHandle file, uint index, out HCCRates rates)			
Get the selected rates structure from history buffer to do further work with its values.			
Parameters			
Type	ID	Direction	Description
HCCHandle	file	IN	File handle associated with the history buffer.
uint	index	IN	Zero based index of the rates structure to load. This index points to an element in history buffer and not to an element in history file.
HCCRates	rates	OUT	The selected rates structure.
Remarks			
There must be rates in history previously loaded by **HCCLoadHistory** and the index must be valid one.			

ReturnCode **HCCGetRatesDouble**(HCCHandle file, uint index, HCC_RATES_PROPERTY prop, out double value)			
Alternative function to get a member of double type from a rates structure.			
Parameters			
Type	ID	Direction	Description
HCCHandle	file	IN	File handle associated with the history buffer.
uint	index	IN	Zero based index of the rates structure in history buffer.
HCC_RATES_PROPERTY	prop	IN	Member/property to load.
double	value	OUT	Return the desired value.

ReturnCode **HCCGetRatesInteger**(HCCHandle file, uint index, HCC_RATES_PROPERTY prop, out uint value)

Alternative function to get a member of uint type from a rates structure.

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	File handle associated with the history buffer.
uint	index	IN	Zero based index of the rates structure in history buffer.
HCC_RATES_PROPERTY	prop	IN	Member/property to load.
uint	value	OUT	Return the desired value.

ReturnCode **HCCBeginTransaction**(HCCHandle file)

Begin a write transaction for the specified file. A write transaction is a special mode that allows queuing and executing write calls to the HCC file. Instead of allowing writes in immediate mode, a transaction enables a smart caching system along with arbitrary optimizations in a deferred writing mode. This system will convert random writes into sequential writes and will minimize the number of I/O flushes, resulting in better write performance.

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	File handle.

Remarks

Only one transaction can be active at a time, calling this function while a transaction is active will return error code #10022. To commit and subsequently end a transaction, you must call the function **HCCCommitTransaction**. Some HCC functions can only be called while a transaction is active.

ReturnCode **HCCCommitTransaction**(HCCHandle file, out uint updated_count)

Commit all write operations cached in an active transaction.

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	File handle.
uint	updated_count	OUT	Number of rates affected by an update operation.

Remarks

A transaction must be active to call this function; otherwise it will return error code #10022. This function will subsequently end the transaction, clearing its queued write operations. This function performs device I/O operations; it may take a considerable amount of time depending on the complexity and number of write operations. Do not call it under critical and latency-sensitive sections of your programs.

ReturnCode **HCCUpdateRates**(HCCHandle file, uint index, HCCRates rates)

Update rates data at the specified index in file with the data from the specified rates structure. This function operates in M1 period (native) and updates are applied directly to the file. It does not make changes to the loaded history. History must be reloaded to reflect changes.

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	File handle.
uint	index	IN	Zero based index of the rates structure to update. This index points to an element in history file and not to an element in history buffer.
HCCRates	rates	IN	Input rates data to be written. Data must be composed in M1 period.

Remarks

This function can only be called while there is an active write transaction, thus it is not executed immediately, instead, it gets cached and executed when the transaction is committed.
To help preserve file integrity, some validations are applied immediately upon calling this function while others are deferred and applied when the transaction is committed.
When committing a transaction, if multiple update calls were issued and one of them contains invalid rates data then that update operation is ignored and the transaction process is continued with the next update.
Multiple update calls for the same index get overwritten and only the last update call is executed.

ReturnCode **HCCExportHistoryCSV**(HCCHandle file, const(wchar)* csv_path, const(wchar)* sep, const(wchar)* decim, ushort language, uint start, uint how_many, out uint export_count)

Export already memory mapped rates to CSV file.

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	File handle associated with the history buffer.
const(wchar)*	csv_path	IN	Path to the CSV file. If the file does not exist then it will be created. If the file exists already then its content will be replaced.
const(wchar)*	sep	IN	CSV list separator. Pass *null* to use the system defined value.
const(wchar)*	decim	IN	Decimal mark. Pass *null* to use the system defined value.
ushort	language	IN	Language ID of the CSV header. Pass 0 for the default language.
uint	start	IN	Zero based index of the first element.
uint	how_many	IN	Specify how many rates to export. Zero means all.
uint	export_count	OUT	Return the number of rates exported.

Remarks

This function export rates previously loaded by **HCCLoadHistory** only. If history is empty because it wasn't loaded before or file did not have rates available then this function will return error code #10017 or #10018 accordingly.

In Windows, you can change the default system parameters at the regional settings in the control panel.

ReturnCode **HCCExportCSV**(HCCHandle file, const(wchar)* csv_path, const(wchar)* sep, const(wchar)* decim, ushort language, uint period, uint start, uint how_many, out uint export_count)

Directly export rates within the specified range to a CSV file.

Parameters

Type	ID	Direction	Description
HCCHandle	file	IN	File handle to export.
const(wchar)*	csv_path	IN	Path to the CSV file. If the file does not exist then it will be created. If the file exists already then its content will be replaced.
const(wchar)*	sep	IN	CSV list separator. Pass *null* to use the system defined value.
const(wchar)*	decim	IN	Decimal mark. Pass *null* to use the system defined value.
ushort	language	IN	Language ID of the CSV header. Pass 0 for the default language.
uint	period	IN	Timeframe of history. Can be any value >0 or one of the HCC predefined ones (HCC_TIMEFRAME). This parameter is still experimental, you should pass 1.
uint	start	IN	Zero based index of the first element.
uint	how_many	IN	Specify how many rates to export. Zero means all.
uint	export_count	OUT	Return the number of rates exported.

Remarks

This function directly exports rates without the need to load history before.

In Windows, you can change the default system parameters at the regional settings in the control panel.

ReturnCode **HCC2CSV**(const(wchar)* hcc_path, const(wchar)* csv_path, const(wchar)* sep, const(wchar)* decim, ushort language, uint period, uint start, uint how_many, out uint export_count)

Convenient function to directly and efficiently export an HCC file to a CSV file.

Parameters

Type	ID	Direction	Description
const(wchar)*	hcc_path	IN	Path to the source history file.
const(wchar)*	csv_path	IN	Path to the CSV file. If the file does not exist then it will be created. If the file exists already then its content will be replaced.
const(wchar)*	sep	IN	CSV list separator. Pass *null* to use the system defined value.
const(wchar)*	decim	IN	Decimal mark. Pass *null* to use the system defined value.
ushort	language	IN	Language ID of the CSV header. Pass 0 for the default language.
uint	period	IN	Timeframe of history. Can be any value >0 or one of the HCC predefined ones (HCC_TIMEFRAME). This parameter is still experimental, you should pass 1.
uint	start	IN	Zero based index of the first element.
uint	how_many	IN	Specify how many rates to export. Zero means all.
uint	export_count	OUT	Return the number of rates exported.

Remarks

This export function is the most efficient one because it does not need to create a file handle and it does not go through a couple of overheads.

In Windows, you can change the default system parameters at the regional settings in the control panel.