TCR1000 PC Card
SDK Information

Masterclock®

Table of Contents

Introduction

The TCR Software Development Kit (SDK) is a Windows dynamic-link library (DLL) which defines an application programming interface (API) support files so the DLL can be used. The SDK for the TCR1000 time code reader card driver is compatible with Windows XP, Windows 7, Server 2003, and Server 2008. There DLL comes in two versions, one for x32 systems and one for x64 systems. To operate, the SDK requires an installed and correctly functioning driver for the TCR1000 time code reader card.

The SDK supports SMPTE (all formats) and IRIG (B and B1 formats) time code operation, and automatically determines the correct internal operation. Date support is also provided for SMPTE (Leitch^{TM} format) and IRIG (IEEE 1344). The SDK also provides access to all user groups (SMPTE) and control functions (IRIG) for applications that require access to special information in these fields. The SDK also provides a callback mechanism for TCR1000 interrupts, where applications can define user-level routines that will be executed when the interrupts occur.

The SDK supports two methods of obtaining time code from the TCR1000. The TCR1000_ApiPollTimeCode() function provides a simplistic but less accurate method of obtaining time code. The accuracy of a polled time code is only plus or minus the frame rate (e.g. for SMPTE 30 frames/second time code, ± 33ms). The SDK also provides an asynchronous time notification mechanism whereby a user-supplied callback function is called when significant events occur during time code decoding operation. The time provided with such events is time-stamped with the Windows high-performance timer at the time the event occurs. The timestamp can then be used to derive precise real-time information (+/- 2 millesecond). [See the included VCSample demonstration application for an example of computing high-precision time.]

Abbreviations Used

The following abbreviations are used in this documentation:

API – The application programming interface services provided by the TCR.DLL library
TCR1000 – Refers to the TCR1000 time code reader card
SMPTE – Society of Motion Picture & Television Engineers-defined time code format
IRIG – Range Commander’s Council/Interrange Instrumentation Group-defined time code format
Windows – One of the Microsoft Windows operating systems (XP,Windows 7,Server 2003,Server 2008)
Driver – The TCR1000 time code reader card driver for the Windows operation system.

Recommending reading:

- SMPTE 12M - Time and Control Code Standard
- Range Commander’s Council IRIG Standard 200-95 – Serial Time Code Formats
- IEEE 1344-1995 – Time zone & date encoding for IRIG-B and IRIG-B1 time codes
- Leitch^{TM} Application Note – Time Code Tutorial-Longitudinal SMPTE Time Code for NTSC Color Video Systems

Project Integration Instructions

A Visual Studio application will link to tcr1000_rev_a.dll (for x64, tcr1000_x64_rev_a.dll). The TCR .dll file was created using Visual Studio 2010. It is an unmanaged .dll . For the users application to load the .dll the user may have to load the .dll dynamically. This is especially true if the user application is managed. See http://msdn.microsoft.com/en-us/library/ms810279.aspx for loading a .dll dynamically. Any modules of a Visual C++ application using the API functions should include INTAPI.H.

Use of API Limitation due to Masterclock TCR1000 service

Only one application at a time is allowed to load the tcr1000_rev_a.dll (for x64 system, tcr1000_x64_rev_a.dll). The Masterclock TCR1000 service, TCR1000Sync_REV_A, loads the TCR1000 .dll . If the Masterclock service is running the user **CANNOT** load the TCR1000 .dll . Doing so will cause the Masterclock TCR1000 service (and probably the user application) to fail. If the user wishes to monitor the TCR1000 status, time code status, current time, etc., the user must query the Masterclock service to get this information and NOT use the .dll API calls. A Visual Studio 2010 C++ sample project is included in the folder “Tcr1000 Monitor Sample”. This project shows how to monitor the TCR1000 via the Masterclock service using a Visual Studio construct called pipes.

TCR DLL API Function Reference

TCR1000_ApiGetDLLInitStatus

The **TCR1000_ApiGetDLLInitStatus** returns the DLL internal initialization status.

DWORD TCR1000_ApiGetDLLInitStatus(void)

Parameters

This function has no parameters.

Return Values

The return value is one of DLL_INIT_xxx defined in INTAPI.H:

DLL_INIT_OK	The DLL initialized successfully.
DLL_INIT_CFGERR	Driver configuration information is corrupt or missing from the registry. The TCR driver set should be re-installed.
DLL_INIT_TCPROBLEM	Unable to detect time code type.
DLL_INIT_NOKMD	The TCR driver is not loaded. See troubleshooting section of TCR manual for assistance.
DLL_INIT_UNEXPECTED	Unexpected initialization failure. Contact technical support.

Remarks

The DLL initializes an instance of itself on behalf of each calling process. A process should always call this function to determine the DLL initialization status before using any other API functions.

TCR1000_ApiGetTimeCodeStatus

The **TCR1000_ApiGetTimeCodeStatus** returns the status of time code decoding on the TCR.

DWORD TCR1000_ApiGetTimeCodeStatus(void)

Parameters

This function has no parameters.

Return Values

The return value is one of DLL_TC_STATUS_xxx defined in INTAPI.H, currently:

DLL_TC_STATUS_NORMAL	Time code is available.
DLL_TC_STATUS_INVALID	Time code data is out of range.
DLL_TC_STATUS_LOST	Time code is not available.
DLL_TC_STATUS_DRVFAIL	Time code not available because of failed communications with the TCR driver.
DLL_TC_STATUS_CALIB	The TCR1000 card is calibrating the time code signal level.

Remarks

The time code status is determined by the TCR driver software and provided to the DLL for reporting purposes.

TCR1000_ApiGetDeviceStatus

The **TCR1000_ApiGetDeviceStatus** function returns the status of communication with the TCR.

DWORD TCR1000_ApiGetDeviceStatus(void)

Parameters

This function has no parameters.

Return Values

The return value is one of HW_STATUS_xxx defined in INTAPI.H, currently:

HW_STATUS_NORMAL	Driver is interacting normally with the TCR.
HW_STATUS_CARDCOMMFAIL	Driver is failing I/O port communications with the TCR.

Remarks

The last asynchronous or polled time code operation sets this status.

TCR1000_ApiGetGpsStatus

The **TCR1000_ApiGetGpsStatus** function returns information relevant to operation of the Masterclock, Inc. GPS device if SMPTE is being decoded and the master clock is configured to return GPS status in the SMPTE time code frame.

DWORD TCR1000_ApiGetGpsStatus(void)

Parameters

This function has no parameters.

Return Values

The return value is one of GPS_STATUS_xxx defined in INTAPI.H, currently:

GPS_STATUS_FREEWHEEL	Masterclock GPS device is generating time code internally, rather than from a GPS signal.

Remarks

The return value of this function for time code sources other than a Masterclock, Inc. GPS master clock with Masterclock proprietary status bits enabled in the SMPTE time code is undefined.

TCR1000_ApiSetFrameRate

This function is deprecated for the TCR1000. Since the TCR1000 detects the frame rate, this is no longer function necessary. It still exists, but the frame rate is set from the card data, not by this function.

The **TCR1000_ApiSetFrameRate** function sets the frame rate for time code types that support sub-second frame rates.

BOOL TCR1000_ApiSetFrameRate(
    WORD *rate* // the sub-second frame rate of the time code in use
);

Parameters

rate

The value of this parameter is used by the API's asynchronous and polling time functions to derive the wMilliseconds member of the TCRTIME structure. Currently, this feature is available only for SMPTE-type
time codes with valid frame rates being 30, 25, or 24.

Return Values

TRUE if the function succeeded in registering the frame rate. FALSE if the specified frame rate was invalid for the time code type being decoded, or if the time code does not support sub-second frame rates.

Remarks

The API will not set the wMilliseconds member in the TCRTIME data structure returned by asynchronous and polled time functions unless the time code frame rate is specified by the user application. Some asynchronous functions, such as ASYNC_TIMESET and ASYNC_PERSEC will set wMilliseconds to 0 anyway since the event always falls on a one-second boundary.

TCR1000_ApiPollTimeCode

The **TCR1000_ApiPollTimeCode** returns the current time code and sets the time code and device status.

DWORD TCR1000_ApiPollTimeCode(
    PTCRTIME *tcrt* // pointer to TCRTIME structure that receives polled time data
);

Parameters

tcrt

Pointer to a TCRTIME structure that is to receive polled time data.

Return Values

The return value is one of DLL_TC_STATUS_xxx defined in INTAPI.H:

DLL_TC_STATUS_NORMAL	Time code is available.
DLL_TC_STATUS_INVALID	Time code data is out of range.
DLL_TC_STATUS_LOST	Time code is not available.
DLL_TC_STATUS_DRVFAIL	Time code not available because of failed communications with the TCR driver.
DLL_TC_STATUS_CALIB	The TCR1000 card is calibrating to the time code signal level.

Remarks

The API will attempt to fill the date members of the TCRTIME structure relative to the type of time code being decoded. If the time code source is not actually encoding date, the value of the date members is undefined. It is the responsibility of the application programmer to determine if date will or will not be encoded by the time code source.

TCR1000_ApiEnableCallback

The **TCR1000_ApiEnableCallback** function enables the asynchronous callback of a user-supplied callback function, or adds hooks to an already-established callback function, on behalf of the calling thread.

```
DWORD TCR1000_ApiEnableCallback(
        DWORD cbToAdd, // mask of callback hooks
        PTCRASYNC_CALLBACK_ROUTINE cbFunction // pointer to user-supplied callback function
);
```

Parameters

cbToAdd

Any combination of callback hooks ASYNC_xxx (defined in INTAPI.H) to be enabled or added, from:

ASYNC_TIMESET	Once/minute interrupt (second = 30, frame = 0)
ASYNC_PERSEC	Once/second interrupt (second = 0)
ASYNC_ENDTC	Once/frame interrupt (after time code data, before sync word)
ASYNC_STARTTC	Once/frame interrupt (before time code data, after sync word)
ASYNC_TIMELOSS	Time code lost
ASYNC_TIMEGAIN	Time code regained from lost state

cbFunction

A user-supplied callback function of type PTCRASYNC_CALLBACK_ROUTINE. A skeleton example of the callback function would be:

```
DWORD _stdcall TCR1000_ApiAsyncCallback(DWORD dwStatus, PVOID data, DWORD dataLen)
{
        // code here to handle asynchronous notifications
}
```

Return Values

On success, the new callback hooks mask (always non-zero). On failure, 0.

Remarks

Each thread in a process may have an active callback function but no more than one. Additional calls to **TCR1000_ApiEnableCallback** in the thread will use the function passed by the initial call. All calls to **TCR1000_ApiEnableCallback** are on behalf of the calling thread, not the calling process. **TCR1000_ApiEnableCallback** must be called once for every event that the callback function should be run for. Enabling a callback, in some circumstances, can affect the callback hooks of other threads or other processes using the API. See the section entitled *Relationship of TCR Interrupts to API Asynchronous Notifications* for additional description of issues regarding the use of this function.

TCR1000_ApiDisableCallback

The **TCR1000_ApiDisableCallback** function disables one or more of the callback hooks currently active for the
callback function of the calling thread.

DWORD TCR1000_ApiDisableCallback(
DWORD *cbToRemove* // mask of callback hooks to remove
);

Parameters
cbToRemove

Any combination of callback hooks ASYNC_xxx (defined in IntApiDefs.H) to be disabled for the callback
function in use by the calling thread:

ASYNC_TIMESET	Once/minute interrupt (second = 30, frame = 0)
ASYNC_PERSEC	Once/second interrupt (second = 0)
ASYNC_ENDTC	Once/frame interrupt (after time code data, before sync word)
ASYNC_STARTTC	Once/frame interrupt (before time code data, after sync word)
ASYNC_TIMELOSS	Time code lost
ASYNC_TIMEGAIN	Time code regained from lost state

Return Values
On success, the new callback hooks mask
On failure, ASYNC_MASK_ERROR

Remarks
A thread must disable its callback function before terminating.

Each thread in a process may have an active callback function but no more than one. All calls to **TCR1000_ApiDisableCallback** are on behalf of the calling thread, not the calling process. Disabling a callback hook does not affect the callback hooks of other threads or other processes. Call **TCR1000_ApiDisableCallback** with *cbToRemove* set to ASYNC_ALL to disable the callback function for the current thread. When all callback hooks have been disabled for a callback function the function is no longer called and is removed from the API's internal callback tables. See the section entitled *Relationship of TCR Interrupts to TCR DLL Asynchronous Notification* for additional issues regarding the use of this function.

TCR1000_ApiShutdown

For the TCR1000, this function is now deprecated and does nothing. The DLL can now handle applications closing without disabling their callback routines.

The **TCR1000_ApiShutdown** function allows the DLL to gracefully shut down threads and resources allocated on
behalf of the calling process.

void TCR1000_ApiShutdown(void)

Parameters

This function has no parameters.

Return Values

This function returns no values.

Remarks

This function should be called before a process using the API terminates.

TCR1000_ApiGetTimeCodeType

The **TCR1000_ApiGetTimeCodeType** function returns the time code type currently in use.

DWORD TCR1000_ApiGetTimeCodeType(void)

Parameters

This function has no parameters

Return Values

On of TC_TYPE_xxx defined in INTAPI.H:

TC_TYPE_SMPTE	SMPTE time code type
TC_TYPE_IRIG	IRIG-B time code type
TC_TYPE_UNDEFINED	Undefined time code type

TCR1000_ApiClearEnhanced

The **TCR1000_ApiClearEnhanced** function forces off all enabled enhanced interrupt features on the TCR1000
time code reader card.

BOOL TCR1000_ApiClearEnhanced(void)

Parameters
This function has no parameters.

Return Values
Returns TRUE if all enhanced features cleared successfully. Returns FALSE if unable to clear enhanced features.

Remarks
Use of this function defies the multithread-safe capability of this API and is not recommended for use in an environment where multiple applications will use the asynchronous notifications.

TCR1000_ApiRequestRecalibration

The **TCR1000_ApiRequestRecalibration** function forces the TCR1000 card to recalibrate the automatic gain control to the incoming time code's signal level.

BOOL TCR1000_ApiRequestRecalibration (void)

Parameters
This function has no parameters.

Return Values
Returns TRUE if recalibration started successfully.

Remarks
Once the TCR1000 card has detected the time code format and calibrated its gain level to the time code's signal level it will expect that level to remain constant. If the time code format changes or the level fluctuates, the card may cease to decode time code properly. In a situation where a level change is anticipated, such as a switch to a different time code feed, this API instructs the TCR1000 to recalibrate the gain.

This function can be used to switch between SMPTE and IRIG-B time codes.

TCR1000_ApiGetDLLVersion

The **TCR1000_ApiGetDLLVersion** function returns the software version of the TCR DLL API library.

```
void TCR1000_ApiGetDLLVersion(
        PUINT verMaj, // receives major version number
        PUINT verMin, // receives minor version number
        PUINT verRev // receives revision level
);
```

Parameters

verMaj

A pointer to a variable that will receive the major version of the API library.

verMin

A pointer to a variable that will receive the minor version of the API library.

verRev

A pointer to a variable that will receive the revision level of the API library.

Return Values

This function returns no values.

TCR1000_ApiGetDriverVersion

The **TCR1000_ApiGetDriverVersion** function returns the software version of the TCR driver.

BOOL TCR1000_ApiGetDriverVersion(
    PUINT verMaj, // receives major version number
    PUINT verMin, // receives minor version number
    PUINT verRev // receives revision level
);

Parameters

verMaj

A pointer to a variable that will receive the major version of the TCR driver.

verMin

A pointer to a variable that will receive the minor version of the TCR driver.

verRev

A pointer to a variable that will receive the revision level of the TCR driver.

Return Values

Returns TRUE with the TCR driver version number in parameters. Returns FALSE if driver version could not be determined.

TCR1000_ApiGetFirmwareVersion

The **TCR1000_ApiGetFirmwareVersion** function returns the software version of the TCR driver.

BOOL TCR1000_ApiGetFirmwareVersion(
        PUINT verMaj, // receives major version number
        PUINT verMin, // receives minor version number
        PUINT verRev // receives revision level
);

Parameters

verMaj

A pointer to a variable that will receive the major version of the TCR firmware.

verMin

A pointer to a variable that will receive the minor version of the TCR firmware.

verRev

A pointer to a variable that will receive the revision level of the TCR firmware.

Return Values

Returns TRUE with the TCR firmware version number in parameters. Returns FALSE if the firmware version could not be determined.

TCR1000_ApiGetFirmwareChecksum

The **TCR1000_ApiGetFirmwareChecksum** function returns the checksum of the TCR firmware.

BOOL TCR1000_ApiGetFirmwareChecksum(
        PDWORD pChecksum // receives checksum of the TCR firmware
);

Parameters

verMaj

A pointer to a variable that will receive the TCR firmware checksum.

Return Values

Returns TRUE with the TCR firmware checksum in parameters. Returns FALSE if the firmware checksum could not be determined.

The Asynchronous Callback Function

When enabled, the API will call a user-supplied callback function reporting the activation of interrupts on the TCR.

An example of the callback function in C/C++:

```
DWORD _stdcall TCR1000_ApiAsyncCallback (DWORD dwStatus, PVOID data, DWORD dataLen)
{
        PTCRTIME time;

        switch(dwStatus)
        {
                case ASYNC_PERSEC:
                        // data is a pointer to a TCRTIME data structure and dataLen = sizeof(TCRTIME)
                        time = (PTCRTIME)data;
                        break;

                case ASYNC_TIMESET:
                        // data is a pointer to a TCRTIME data structure and dataLen = sizeof(TCRTIME)
                        time = (PTCRTIME)data;
                        break;

                case ASYNC_STARTTC:
                        // data is a pointer to a TCRTIME data structure and dataLen = sizeof(TCRTIME)
                        time = (PTCRTIME)data;
                        break;

                case ASYNC_ENDTC:
                        // data is a pointer to a TCRTIME data structure and dataLen = sizeof(TCRTIME)
                        time = (PTCRTIME)data;
                        break;

                case ASYNC_TIMELOSS:
                        // data is NULL and dataLen = 0
                        break;

                case ASYNC_TIMEGAIN:
                        // data is NULL and dataLen = 0
                        break;
        }

        return(0);
}
```

The callback function may have any name that the user application requires, but must have the exact calling convention and parameter declaration as shown above to avoid corruption of the stack when the function is called by the API.

When the callback function provides a TCRTIME data structure as the data passed, the date members of the TCRTIME structure will be set relative to the type of time code being decoded. If the time code source is not actually encoding date, the value of the date members is undefined. It is the responsibility of the application programmer to determine if date will or will not be encoded by the time code source.

Structure and Constants Reference

```
// A structure defining a SMPTE time code frame (the union ‘rawTc’ member of TCRTIME will contain
// this data structure if operating with SMPTE time code.

typedef struct _SMPTE
{
UCHAR fu; // frame units
UCHAR u1; // user group 1
UCHAR ft; // frame tens
UCHAR u2; // user group 2
UCHAR su; // second units
UCHAR u3; // user group 3
UCHAR st; // second tens
UCHAR u4; // user group 4
UCHAR mu; // minute units
UCHAR u5; // user group 5
UCHAR mt; // minute tens
UCHAR u6; // user group 6
UCHAR hu; // hour units
UCHAR u7; // user group 7
UCHAR ht; // hour tens
UCHAR u8; // user group 8
} SMPTE, *PSMPTE;
```

```
// A structure defining an IRIG-B or IRIG-B1 time code frame (the union ‘rawTc’ member of TCRTIME
will contain
// this data structure if operating with IRIG-B or IRIG-B1 time code.

typedef struct _IRIG
{
UCHAR su; // second units
UCHAR st; // second tens
UCHAR mu; // minute units
UCHAR mt; // minute tens
UCHAR hu; // hour units
UCHAR ht; // hour tens
UCHAR du; // day units
UCHAR dt; // day tens
UCHAR dh; // day hundreds
UCHAR cf1; // control functions 1 (see TCR-500 manual for more information)
UCHAR cf2; // control functions 2 (see TCR-500 manual for more information)
UCHAR cf3; // control functions 3 (see TCR-500 manual for more information)
UCHAR cf4; // control functions 4 (see TCR-500 manual for more information)
UCHAR cf5; // control functions 5 (see TCR-500 manual for more information)
UCHAR cf6; // control functions 6 (see TCR-500 manual for more information)
UCHAR cf7; // control functions 7 (see TCR-500 manual for more information)
} IRIG, *PIRIG;

// TCRTIME data structure provided by polled and asynchronous time responses

typedef struct _TCRTIME
{
        // What type of timecode are we decoding

        DWORD dwTimeCodeType;

        // raw time code values

        union
        {
                IRIG            irig;
                SMPTE           smpte;
        } rawTc;

        // if a time given asynchronously (by an interrupt on TCR), contains high-performance counter
        // reading at time of interrupt, for polled time codes the value of this member is undefined

        LARGE_INTEGER intPerfCount;

        // decoded time, remember that wMilliseconds is not provided unless frame rate is set
        // via TCRSetFrameRate() call.

        WORD                    wHour;
        WORD                    wMinute;
        WORD                    wSecond;
        WORD                    wMilliseconds;
        WORD                    wMonth;
        WORD                    wDay;
        WORD                    wYear;

        // validity flag (does checksum pass?  TCR-500/500PCI only)

        BYTE            valid;

} TCRTIME, *PTCRTIME;
```

// Communication statuses between TCR device driver and TCR1000

HW_STATUS_NORMAL	0x00 // normal
HW_STATUS_CARDCOMMFAIL	0x01 // I/O communication failure

// Time code types

TC_TYPE_SMPTE	0x00000001L	
TC_TYPE_IRIG	0x00000002L	// deprecated
TC_TYPE_IRIG_B1	0x00000002L	// originally, IRIG was only B1
TC_TYPE_IRIG_B	0x00000003L	
TC_TYPE_UNDEFINED	0x00008000L	

// DLL time code statuses

DLL_TC_STATUS_NORMAL	0x00	// time code read was successfully
DLL_TC_STATUS_INVALID	0x01	// time code data is invalid
DLL_TC_STATUS_LOST	0x02	// time code not available
DLL_TC_STATUS_DRVFAIL	0x03	// TCR driver communication failure
DLL_TC_STATUS_CALIB	0x04	// time code signal calibration in progress

// DLL initialization statuses

DLL_INIT_OK	0	// initialization was successful
DLL_INIT_CFGERR	3	// TCRSync configuration is invalid/missing
DLL_INIT_NOKMD	4	// unable to find the TCR driver
DLL_INIT_TCPROBLEM	5	// unable to detect time code type
DLL_INIT_INV_DLL	6	// problem loading DLL or its functions
DLL_INIT_UNEXPECTED	255	// unexpected failure

// Asynchronous notifications callback hooks (maps to TCR interrupts)

ASYNC_TIMESET	0x00000001	// once/minute (sec=30, frame=0)
ASYNC_TIMEGAIN	0x00000002	// time code regained from lost state
ASYNC_TIMELOSS	0x00000004	// time code lost interrupt
ASYNC_ENDTC	0x00000008	// 1/frame (after time code/before sync word)
ASYNC_STARTTC	0x00000010	// 1/frame (before time code, after sync word)
ASYNC_PERSEC	0x00000020	// once/second interrupt (frame=0)
ASYNC_ALL	0x7FFFFFFF	
ASYNC_MASK_ERROR	0x80000000	

// GPS-100/200/200A statuses

GPS_STATUS_FREEWHEEL 0x00000001 // unit is freewheeling

// User callback routine function type

typedef DWORD (__stdcall *PTCRASYNC_CALLBACK_ROUTINE)(DWORD dwStatus, PVOID data, DWORD dataLen);

Relationship of TCR Interrupts to API Asynchronous Notifications

This section describes the relationship between the interrupts that the TCR throws in response to time code events and the API asynchronous notification hooks. The TCR driver is an intermediary between the aforementioned objects but this is transparent to the application programmer. Knowledge of the information in this section is not required to use the API but should be considered in scenarios where multi-process/thread use of API will be implemented.

TCR interrupt	API callback hook
Time Set Interrupt (Once/minute interrupt thrown when time code seconds= 30, frame = 0)	ASYNC_TIMESET
Once/Second Interrupt (Once/second interrupt thrown when time code frame = 0)	ASYNC_PERSEC
End Time Code Interrupt (Once/frame interrupt thrown after time code data decoded, before sync word)	ASYNC_ENDTC
Start Time Code Interrupt (Once/frame interrupt thrown before time code data decoded, after sync word [top of the frame])	ASYNC_STARTTC
Time Code Loss (Thrown when time code is lost from an active decoding state)	ASYNC_TIMELOSS
Time Gain Interrupt (Thrown when time code is regained from a lost state)	ASYNC_TIMEGAIN

The *Time Set Interrupt*, *Time Loss Interrupt*, and *Time Gain Interrupt* are always thrown by the TCR when said event occurs and cannot be disabled. These interrupts are supported by the TCR in all modes of time code decoding operation.

When the *Once/Second Interrupt* is enabled the *Time Set Interrupt* is preempted and will not be passed by the TCR driver to the API. When the *Start Time Code Interrupt* is enabled both the *Time Set Interrupt* and *Once/Second Interrupt* are preempted. The reason for the preemption is that all three interrupts, regardless of frequency, will fall at the same time code transition point in real-time. The preemption is internal to the TCR. When a thread enables an asynchronous notification callback hook that maps to a TCR "enhanced interrupt" that interrupt will be enabled on time code reader card. It follows that one can break the operation of another thread/process's callback functions by enabling an "enhanced interrupt" on the TCR that preempts another interrupt (as previously discussed). Caution should be exercised in planning what callback hooks will be used by an application calling the API's. **In environments where multithreaded/multiprocess use of the API is planned it is strongly recommended that all applications utilize only the ASYNC_TIMESET callback hook (TCR *Time Set Interrupt*).**

Legal Information

The information contained in this document is subject to change without notice. Masterclock, Inc. (hereinafter Masterclock) provides the API development software package on an as-is basis. Masterclock makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Masterclock shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

API/Document Revision History

- New release September 1, 2011

Contacting Us

Masterclock, Inc.
2484 West Clay Street
St. Charles, MO 63301 USA

website
www.masterclock.com

USA and Canada
1-800-940-2248
1-636-724-3666
1-636-724-3776 (fax)

International
1-636-724-3666
1-636-724-3776 (fax)

Sales
sales@masterclock.com

Technical Support
support@masterclock.com

Masterclock, Inc.
2484 West Clay St.
St. Charles, MO 63301
+1-(636)-724-3666
1-800-724-1778
support@masterclock.com
sales@masterclock.com