



---

# OpenClovis Software Development Kit (SDK) Service Description and API Reference for Timer Service

For OpenClovis SDK Release 2.3 V0.4  
Document Revision Date: March 08, 2007

---

**Copyright © 2007 OpenClovis Inc.**

**All rights reserved**

This document contains proprietary and confidential information of OpenClovis Inc., and may not be used, modified, copied, reproduced, disclosed or distributed in whole or in part except as authorized by OpenClovis Inc. This document is intended for informational use and planning purposes only. All planned features, specifications, and content are subject to change without notice.

**Third-Party Trademarks**

Sun, Sun Microsystems, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark of The Open Group. Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. CLEI is a trademark of Telcordia Technologies, Inc. Adobe, Acrobat, and Acrobat Reader are registered trademarks of Adobe Systems, Inc. All other trademarks, service marks, product names, or brand names mentioned in this document are the property of their respective owners.

**Government Use**

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 12.212 (Commercial Computer Software-Restricted Rights) and DFAR 227.7202 (Rights in Technical Data and Computer Software), as applicable.

**Note:** This document is not subject of the GPL license, even if you have obtained this document as a part of the GPL-ed version of OpenClovis SDK.

# Contents

<b>1</b>	<b>Functional Overview</b>	<b>1</b>
<b>2</b>	<b>Service Model</b>	<b>3</b>
<b>3</b>	<b>Service APIs</b>	<b>5</b>
3.1	Type Definitions . . . . .	5
3.1.1	ClTimerTimeOutT . . . . .	5
3.1.2	ClTimerTypeT . . . . .	5
3.1.3	ClTimerContextT . . . . .	5
3.1.4	ClTimerCallBackT . . . . .	6
3.1.5	ClTimerHandleT . . . . .	6
3.2	Library Life Cycle functions . . . . .	7
3.2.1	clTimerConfigInitialize . . . . .	7
3.2.2	clTimerInitialize . . . . .	8
3.2.3	clTimerFinalize . . . . .	9
3.3	Functional functions . . . . .	10
3.3.1	clTimerCreate . . . . .	10
3.3.2	clTimerDelete . . . . .	11
3.3.3	clTimerStart . . . . .	12
3.3.4	clTimerStop . . . . .	13
3.3.5	clTimerCreateAndStart . . . . .	14
3.3.6	clTimerRestart . . . . .	15
3.3.7	clTimerUpdate . . . . .	16
3.3.8	clTimerTypeGet . . . . .	17
<b>4</b>	<b>Service Management Information Model</b>	<b>19</b>
<b>5</b>	<b>Service Notifications</b>	<b>21</b>
<b>6</b>	<b>Debug CLIs</b>	<b>23</b>



# Chapter 1

## Functional Overview

The OpenClovis Timer Library enables you to create multiple timers to execute application specific functionality at certain intervals. It performs the following functions:

- Creates multiple timers.
- Specifies a time-out value for each timer.
- Creates one-shot or repetitive timers.
- Specifies an application specific function that should be executed every time the timer fires or expires.

The operating systems such as Linux or BSD have a limit on the number of timers that can be created in an application. However, an application such as OSPF and BGP require a lot of timers to be created at various points during the execution of the application. The Timer Library enables you to create multiple timers that executes a user-defined function when the timer expires. The timer can be of two types: one-shot or repetitive. One-shot timers expire once and are not automatically restarted. Repetitive timers are automatically restarted by the timer library every time they expire.

A user application needs to link to the timer library to utilize it.

The timer library provides APIs to create, delete, start, stop, and restart a timer.

The timer library creates a thread that handles expired timers for each application with which it is linked. As currently implemented, this thread processes expired timers every 10ms.



## **Chapter 2**

# **Service Model**

TBD





# Chapter 3

## Service APIs

### 3.1 Type Definitions

#### 3.1.1 CTimerTimeOutT

```
typedef struct {  
    CUInt32T tsMilliSec;  
    CUInt32T tsSec;  
} CTimerTimeOutT;
```

The structure `CTimerTimeOutT` contains the timeout value in seconds and milliseconds.

- *tsMilliSec* - The timeout value in milliseconds.
- *tsSec* - The timeout value in seconds.

#### 3.1.2 CTimerTypeT

```
typedef enum {  
    CL_TIMER_ONE_SHOT=0,  
    CL_TIMER_REPETITIVE  
} CTimerTypeT;
```

The enumeration, `CTimerTypeT`, contains the type of action to be performed on timer expiry. The attributes of the enumeration are:

- *CL\_TIMER\_ONE\_SHOT* - Timer starts automatically after timeout.
- *CL\_TIMER\_REPETITIVE* - Timer has not started after timeout.

#### 3.1.3 CTimerContextT

```
typedef enum {  
    CL_TIMER_TASK_CONTEXT=0,  
    CL_TIMER_SEPARATE_CONTEXT  
} CTimerContextT;
```

The enumeration, `ClTimerContextT`, contains the method of invocation of the timer callback function on timer expiry. The attributes of the enumeration are:

- `CL_TIMER_TASK_CONTEXT` - Timer callback function is called in the same context.
- `CL_TIMER_SEPARATE_CONTEXT` - New thread is created to invoke the callback.

### **3.1.4 CTimerCallBackT**

```
typedef ClRcT(*CTimerCallBackT)(void *);
```

Type of the function that is called on expiration of timer.

### **3.1.5 CTimerHandleT**

```
typedef ClHandleT CTimerHandleT;
```

The type of the callback function that is called on timer expiry.

## 3.2 Library Life Cycle functions

### 3.2.1 clTimerConfigInitialize

#### clTimerConfigInitialize

**Synopsis:**

Configures the Timer library.

**Header File:**

clTimerApi.h

**Syntax:**

```
ClRcT clTimerConfigInitialize(  
    CL_IN void* pConfigData);
```

**Parameters:**

**pConfigData:** Pointer to instance of configuration structure. You must pass `ClTimerConfigT` as an input.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_TIMER\_ERR\_INVLD\_PARAM:** An invalid parameter has been passed to the function.

**CL\_ERR\_NULL\_POINTER:** pConfigData contains a NULL pointer.

**Description:**

This function is used to configure the timer service library. The configurable parameters are:

**1. Timer Resolution**

This value is in milliseconds and cannot be less than 10 milliseconds. Default value is 10 milliseconds.

**2. Timer Task Priority**

This value can vary between 1 and 160. Default value is 150.

**Library File:**

libClTimer

**Related Function(s):**

None.

### 3.2.2 cTimerInitialize

#### cTimerInitialize

**Synopsis:**

Initializes the Timer library.

**Header File:**

clTimerApi.h

**Syntax:**

```
ClRcT cTimerInitialize (void);
```

**Parameters:**

None.

**Return values:**

**CL\_OK:** The function executed successfully.

**ERROR:** Failure initializing the Timer.

**Description:**

This function is used to initialize the timer service library. After invoking this function the application can invoke other timer related functions.

**Library File:**

libClTimer

**Related Function(s):**

[cTimerFinalize](#)

## 3.2 Library Life Cycle functions

---

### 3.2.3 clTimerFinalize

#### clTimerFinalize

**Synopsis:**

Frees the Timer library.

**Header File:**

clTimerApi.h

**Syntax:**

```
ClRcT clTimerFinalize (void);
```

**Parameters:**

None.

**Return values:**

**CL\_OK:** The function executed successfully.

**ERROR:** Failure finalizing the Timer.

**Description:**

This function is used to free the resources acquired during the initialization of the timer service library. This function is invoked during system shutdown or when the timer is not required. All timers that are currently running are stopped and the data structures required to hold the timer information are deleted.

**Library File:**

libClTimer

**Related Function(s):**

[clTimerInitialize](#)

## 3.3 Functional functions

### 3.3.1 clTimerCreate

#### clTimerCreate

##### Synopsis:

Creates a timer.

##### Header File:

clTimerApi.h

##### Syntax:

```
ClRcT clTimerCreate(
    CL_IN      ClTimerTimeoutT  timeout,
    CL_IN      ClTimerTypeT    type,
    CL_IN      ClTimerContextT  timerTaskSpawn,
    CL_IN      ClTimerCallbackT fpAction,
    CL_IN      void*            pActionArgument,
    CL_INOUT   ClTimerHandleT*  pTimerHandle);
```

##### Parameters:

**timeout:** (in) Timeout value of the timer.

**type:** (in) Type of the timer to be created. It can be either one-shot or repetitive.

**timerTaskSpawn:** (in) Determines whether the user-function invoked is in a separate task or in the same context as the timer-task.

**fpAction:** (in) Function to be called after timer expiry.

**pActionArgument:** (in) Argument to be passed to the callback function (fpAction).

**pTimerHandle:** (out) Pointer to the memory location where the timer handle created is being copied.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_TIMER\_ERR\_INVLD\_PARAM:** An invalid parameter has been passed to the function.

**CL\_ERR\_NO\_MEMORY:** Memory allocation failure.

**CL\_ERR\_NULL\_POINTER:** pActionArgument or pTimerHandle contains a NULL pointer.

**CL\_TIMER\_ERR\_NULL\_CALLBACK\_FUNCTION:** The callback function passed to this function is NULL.

**CL\_TIMER\_ERR\_INVALID\_TYPE:** The type of the timer is invalid.

**CL\_TIMER\_ERR\_INVALID\_CONTEXT\_TYPE:** The context is invalid.

##### Description:

This function is used to create a new timer. This timer remains inactive until the timer is started. The callback function is executed in the context of the timer task, when the timer expires. This function returns a handle that needs to be specified when you need to start, stop, restart, or destroy the timer.

##### Library File:

libClTimer

##### Related Function(s):

[clTimerDelete](#) , [clTimerStart](#) , [clTimerRestart](#) , [clTimerCreateAndStart](#)

### 3.3 Functional functions

---

#### 3.3.2 cITimerDelete

##### cITimerDelete

**Synopsis:**

Deletes a timer.

**Header File:**

cITimerApi.h

**Note:**

If the timer being deleted is active, then it is made inactive and deleted.

**Syntax:**

```
CL_RcT cITimerDelete ( CL_INOUT CITimerHandleT* pTimerHandle );
```

**Parameters:**

**pTimerHandle** : (in/out) Pointer to timer handle being deleted. The contents are set to zero.

**Return values:**

**CL\_OK**: The function executed successfully.

**CL\_ERR\_NULL\_POINTER**: pTimerHandle contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE**: timerHandle is an invalid handle.

**CL\_TIMER\_ERR\_INVALID**: The internal timer representation is invalid.

**Description:**

This function is used to delete an existing timer. It is invoked by the application after the timer has expired. This function is usually called during the application exit, but it can also be called at any other time.

**Library File:**

libCITimer

**Related Function(s):**

[cITimerCreate](#) , [cITimerRestart](#) , [cITimerCreateAndStart](#) , [cITimerStop](#)

### 3.3.3 clTimerStart

#### clTimerStart

**Synopsis:**

Starts a timer.

**Header File:**

clTimerApi.h

**Syntax:**

```
ClRcT clTimerStart (
    ClTimerHandleT timerHandle);
```

**Parameters:**

***timerHandle***: Handle of the timer being started.

**Return values:**

***CL\_OK***: The function executed successfully.

***CL\_ERR\_INVALID\_HANDLE***: *timerHandle* is an invalid handle.

***CL\_TIMER\_ERR\_INVALID***: The internal timer representation is invalid.

**Description:**

This function is used to start a timer. Before the timer can be started, the timer must be created and made active. The callback function is executed when the timeout occurs. The callback function would be executed in the context of the timer task.

**Library File:**

libClTimer

**Related Function(s):**

[clTimerCreate](#) , [clTimerRestart](#) , [clTimerCreateAndStart](#) , [clTimerStop](#)



### 3.3 Functional functions

---

#### 3.3.4 clTimerStop

##### clTimerStop

##### Synopsis:

Stops a timer.

##### Header File:

clTimerApi.h

##### Note:

This function only stops the timer and does not destroy it.

##### Syntax:

```
ClRcT clTimerStop (
    CL_IN ClTimerHandleT timerHandle);
```

##### Parameters:

**timerHandle:** (in) Handle of the timer being stopped.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_ERR\_INVALID\_HANDLE:** timerHandle is an invalid handle.

**CL\_TIMER\_ERR\_INVALID:** The internal timer representation is invalid.

##### Description:

This function is used to stop a timer. After invoking this function, the timer becomes inactive, and can be started with `clTimerStart()` or `clTimerRestart()` functions.

##### Library File:

libClTimer

##### Related Function(s):

[clTimerCreate](#) , [clTimerStart](#) , [clTimerCreateAndStart](#) , [clTimerDelete](#)

### 3.3.5 cTimerCreateAndStart

#### cTimerCreateAndStart

**Synopsis:**

Creates a new timer and activates it.

**Header File:**

cTimerApi.h

**Syntax:**

```
ClRcT cTimerCreateAndStart (
    CL_IN          ClTimerTimeoutT  timeout,
    CL_IN          ClTimerTypeT  type,
    CL_IN          ClTimerContextT timerTaskSpawn,
    CL_IN          ClTimerCallbackT fpAction,
    CL_IN          void *pActionArgument,
    CL_INOUT       ClTimerHandleT *pTimerHandle);
```

**Parameters:**

**timeout:** (in) Timeout value of the timer.

**type:** (in) Type of the timer to be created. It can be either one-shot or repetitive.

**timerTaskSpawn:** (in) Determines if the user-function invoked is in a separate task or in the same context as the timer-task.

**fpAction:** (in) Function to be called after timer expiry.

**actionArgument:** (in) Argument to be passed to the callback function (`fpAction`).

**pTimerHandle:** (in/out) Pointer to the memory location where the timer handle created is being copied.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_INVALID\_PARAMETER:** An invalid parameter has been passed to the function.

**CL\_ERR\_NO\_MEMORY:** Memory allocation failure.

**CL\_ERR\_NULL\_POINTER:** `pActionArgument` or `pTimerHandle` contains a NULL pointer.

**CL\_TIMER\_ERR\_NULL\_CALLBACK\_FUNCTION:** An invalid callback function has been passed to the function.

**CL\_TIMER\_ERR\_INVALID\_TYPE:** The type of timer is invalid.

**Description:**

This function is used to create a new timer and activate it. It is essentially a combination of `tsCreate()` and `tsStart()` functions.

This function is useful when you have to create a new timer and activate it at the time of its creation.

**Library File:**

libCITimer

**Related Function(s):**

[cTimerCreate](#) , [cTimerStart](#) , [cTimerStop](#) , [cTimerDelete](#)

### 3.3 Functional functions

---

#### 3.3.6 clTimerRestart

##### clTimerRestart

##### Synopsis:

Restarts a timer.

##### Header File:

clTimerApi.h

##### Syntax:

```
ClRcT clTimerRestart (
    CL_IN      ClTimerHandleT  timerHandle);
```

##### Parameters:

***timerHandle:*** (in) Handle of the timer being restarted.

##### Return values:

***CL\_OK:*** The function executed successfully.

***CL\_ERR\_INVALID\_HANDLE:*** *timerHandle* is an invalid handle.

***CL\_TIMER\_ERR\_INVLD\_STATE:*** Timer is in an invalid state.

***CL\_TIMER\_ERR\_INVALID:*** The internal timer representation is invalid.

##### Description:

This function is used to restart a timer that is created using the `clTimerCreate()`, `clTimerStart()`, or `clTimerCreateAndStart()` functions.

##### Library File:

libClTimer

##### Related Function(s):

[clTimerCreate](#), [clTimerStart](#), [clTimerCreateAndStart](#), [clTimerStop](#), [clTimerDelete](#)

### 3.3.7 cITimerUpdate

#### cITimerUpdate

**Synopsis:**

Updates a timer.

**Header File:**

cITimerApi.h

**Syntax:**

```
ClRcT cITimerUpdate(  
    CL_IN timerHandle,      ClTimerHandleT timerHandle,  
    CL_IN newTimeout);      ClTimerTimeOutT newTimeout);
```

**Parameters:**

**timerHandle:** (in) Handle of the timer being updated.

**newTimeout:** (in) New timeout value for the timer.

**Return values:**

**CL\_OK:** The function executed successfully.

**CL\_ERR\_INVALID\_HANDLE:** timerHandle is an invalid handle.

**CL\_TIMER\_ERR\_INVALID:** The internal timer representation is invalid.

**Description:**

This function is used to update the timeout value of a timer. For this, a timer must be created using `cITimerCreate()` or `cITimerCreateAndStart()` functions. After the timeout value is updated, the timer function is called according to the new timer value.

**Library File:**

libCITimer

**Related Function(s):**

[cITimerCreate](#) , [cITimerStart](#) , [cITimerCreateAndStart](#) , [cITimerStop](#) , [cITimerDelete](#)

### 3.3 Functional functions

---

#### 3.3.8 clTimerTypeGet

##### clTimerTypeGet

##### Synopsis:

Returns the timer type.

##### Header File:

clTimerApi.h

##### Syntax:

```
CL_RcT clTimerTypeGet (
    CL_IN      ClTimerHandleT timerHandle,
    CL_INOUT   ClUInt32T* pTimerType);
```

##### Parameters:

**timerHandle:** (in) Handle of the timer.

**pTimerType:** (in/out) The pointer to the location to which the type of the timer is being copied. It can have two values: If the value is:

- 0: The timer type is one-shot.
- 1: The timer type is repetitive.

##### Return values:

**CL\_OK:** The function executed successfully.

**CL\_ERR\_NULL\_POINTER:** pTimerType contains a NULL pointer.

**CL\_ERR\_INVALID\_HANDLE:** timerHandle is an invalid handle.

**CL\_TIMER\_ERR\_INVALID:** The internal timer representation is invalid.

##### Description:

This function is used to query and return the type of timer, one-shot or repetitive. A one-shot timer is invoked only once and then the timer stops operating. For a repetitive timer, the timeout occurs periodically until the timer is stopped.

##### Library File:

libClTimer

##### Related Function(s):

[clTimerCreate](#) , [clTimerStart](#) , [clTimerCreateAndStart](#) , [clTimerStop](#) , [clTimerDelete](#)



## **Chapter 4**

# **Service Management Information Model**

TBD





## **Chapter 5**

# **Service Notifications**

TBD



## **Chapter 6**

# **Debug CLIs**

TBD

# Index

cllocCommPortDelete, [9](#)  
CITimerCallBackT, [6](#)  
clTimerConfigInitialize, [7](#)  
CITimerContextT, [5](#)  
clTimerCreate, [10](#)  
clTimerCreateAndStart, [14](#)  
clTimerDelete, [11](#)  
CITimerHandleT, [6](#)  
clTimerInitialize, [8](#)  
clTimerRestart, [15](#)  
clTimerStart, [12](#)  
clTimerStop, [13](#)  
CITimerTimeOutT, [5](#)  
clTimerTypeGet, [17](#)  
CITimerTypeT, [5](#)  
clTimerUpdate, [16](#)