cosTime Application

version 1.1
Typeset in \LaTeX{} from SGML source using the DOCBUILDER 3.4 Document System.
Chapter 1

cosTime User’s Guide

The cosTime application is an Erlang implementation of the OMG CORBA Time and TimerEvent Services.

1.1 The cosTime Application

1.1.1 Content Overview

The cosTime documentation is divided into three sections:

- PART ONE - The User’s Guide
  Description of the cosTime Application including services and a small tutorial demonstrating the development of a simple service.

- PART TWO - Release Notes
  A concise history of cosTime.

- PART THREE - The Reference Manual
  A quick reference guide, including a brief description, to all the functions available in cosTime.

1.1.2 Brief Description of the User’s Guide

The User’s Guide contains the following parts:

- cosTime overview
- cosTime installation
- A tutorial example

1.2 Introduction to cosTime

1.2.1 Overview

The cosTime application is Time and TimerEvent Services compliant with the OMG\(^1\) Services CosTime and CosTimerEvent.

\(^1\)URL: http://www.omg.org
Chapter 1: cosTime User's Guide

Purpose and Dependencies

This application uses calendar:now_to_universal_time(Now) to create a UTC. Hence, the underlying OS must deliver a correct result when calling erlang:now().

cosTime is dependent on Orber, which provides CORBA functionality in an Erlang environment.
cosTimerEvent is dependent on Orber and cosNotification, which provides CORBA functionality and Event handling in an Erlang environment.

Prerequisites

To fully understand the concepts presented in the documentation, it is recommended that the user is familiar with distributed programming, CORBA, the Orber and cosNotification applications.

Recommended reading includes CORBA Fundamentals and Programming - Jon Siegel and Open Telecom Platform Documentation Set. It is also helpful to have read Concurrent Programming in Erlang.

1.3 Installing cosTime

1.3.1 Installation Process

This chapter describes how to install cosTime [page 17] in an Erlang Environment.

Preparation

Before starting the installation process for cosTime, the application Orber must be running.

Configuration

When using both the Time and TimerEvent Services the cosTime application first must be installed using cosTime:install_time() and cosTime:install_timerevent(), followed by cosTime:start().

Now we can choose which can start the servers by using cosTime:start_time_service(Tdf, Inaccuracy) and cosTime:start_timerevent_service(TimeService).

1.4 cosTime Examples

1.4.1 A Tutorial on How to Create a Simple Service

Initiate the Application

To use the complete cosTime application Time and Timer Event Services must be installed. The application is then started by using cosTime:start(). To get access to Time Service or Timer Event Service, use start_time_service/2 or start_timerevent_service/1.

The Time Service are global, i.e., there may only exist one instance per Orber domain.
The Timer Event Service is locally registered, i.e., there may only exist one instance per node.
Note:
The Time and Timer Event Service use the time base 15 october 1582 00:00. Performing operations using other time bases will not yield correct result. Furthermore, time and inaccuracy must be expressed in 100 nano seconds.

How to Run Everything

Below is a short transcript on how to run cosTime.

%%% Start Mnesia and Orber
mnesia:delete_schema([node()]),
mnesia:create_schema([node()]),
orber:install([node()]),
mnesia:start(),
orber:start(),

%%% Install Time Service in the IFR.
cosTime:install_time(),

%%% Install Timer Event Service in the IFR. Which, require
%%% the Time Service and cosEvent or cosNotification
%%% application to be installed.
cosNotification:install(),
cosTime:install_timerevent(),

%%% Now start the application and necessary services.
cosTime:start(),
% Tdf == Time displacement factor
% Inaccuracy measured in 100 nano seconds
TS=cosTime:start_time_service(TDF, Inaccuracy),
TES=cosTime:start_timerevent_service(TS),

%%% Access a cosNotification Proxy Push Consumer. How this is
%%% done is implementation specific.
ProxyPushConsumer = ....

%%% How we construct the event is also implementation specific.
AnyEvent = ....

%%% Create a new relative universal time.
%%% Time measured in 100 nano seconds.
UTO='CosTime_TimeService':
    new_universal_time(TS, Time, Inaccuracy, TDF),
EH='CosTimerEvent_TimerEventService':
    register(TES, ProxyPushConsumer, AnyEvent),

%%% If we want to trigger one event Time*10^-7 seconds from now:
    'CosTimerEvent_TimerEventHandler':set_timer(EH, 'TTRelative', UTO),

%%% If we want to trigger an event every Time*10^-7 seconds, starting
%% Time*10^-7 seconds from now:
'CosTimerEvent_TimerHandler':set_timer(EH, 'TTPeriodic', UTO),

%% If we want to use absolute time we must retrieve such an object.
%% One way is to convert the one we got, UTO, by using:
UTO2='CosTime_UTO':absolute_time(UTO),
%% If any other way is used, the correct time base MUST be used, i.e.,
%% 15 october 1582 00:00.
'CosTimerEvent_TimerHandler':set_timer(EH, 'TTAbsolute', UTO2),
cosTime Reference Manual

Short Summaries

- Erlang Module **CosTime::TIO** [page 8] – This module implements the OMG CosTime::TIO interface.
- Erlang Module **CosTime::TimeService** [page 10] – This module implements the OMG CosTime::TimeService interface.
- Erlang Module **CosTime::UTO** [page 12] – This module implements the OMG CosTime::UTO interface.
- Erlang Module **CosTimerEvent::TimerEventHandler** [page 14] – This module implements the OMG CosTimerEvent::TimerEventHandler interface.
- Erlang Module **CosTimerEvent::TimerEventService** [page 16] – This module implements the OMG CosTimerEvent::TimerEventService interface.
- Erlang Module **cosTime** [page 17] – The main module of the cosTime application

**CosTime::TIO**

The following functions are exported:

- `_get_time_interval`(TIO) -> TimeInterval
  [page 8] Return the interval associated with the target object
- `spans`(TIO, UTO) -> Reply
  [page 8] Return an OverlapType which describe how the interval in the target object and the timerange represented by the UTO object overlap
- `overlaps`(TIO, OtherTIO) -> Reply
  [page 8] Return an OverlapType which describe how the interval in the target object and the timerange represented by the TIO object overlap
- `time`(TIO) -> UTO
  [page 9] Return a UTO in which the interval equals the time interval in the target object and time value is the midpoint of the interval

**CosTime::TimeService**

The following functions are exported:

- `universal_time`(TimeService) -> Reply
  [page 10] Return the current time and the Inaccuracy given when starting this application in a UTO

---

CosTime Application
CosTime UTO

The following functions are exported:

- `new_universal_time(TimeService, Time, Inaccuracy, Tdf)` -> UTO
  [page 10] Create a new UTO object representing the time parameters given
- `uto_from_utc(TimeService, Utc)` -> UTO
  [page 10] Create a UTO representing the given time in Utc form
- `new_interval(TimeService, Lower, Upper)` -> TIO
  [page 10] Create a new TIO object representing the input parameters

CosTime_U_TO

The following functions are exported:

- `'get_time'(UTO)` -> ulonglong()
  [page 12] Return the time associated with the target object
- `'get_inaccuracy'(UTO)` -> ulonglong()
  [page 12] Return the inaccuracy associated with the target object
- `'get_tdf'(UTO)` -> short()
  [page 12] Return the time displacement factor associated with the target object
- `'get_utc_time'(UTO)` -> UtcT
  [page 12] Return the data associated with the target object in Utc form
- `absolute_time(UTO)` -> OtherUTO
  [page 12] Create a new UTO object representing the time in the target object added to current time (UTC)
- `compare_time(UTO, ComparisonType, OtherUTO)` -> Reply
  [page 13] Compare the time associated with the target object and the given UTO object
- `time_to_interval(UTO, OtherUTO)` -> TIO
  [page 13] Create a TIO representing the interval between the target object and the given UTO midpoint times
- `interval(UTO)` -> TIO
  [page 13] Create a TIO object representing the error interval around the time value represented by the target object

CosTimerEvent_TimerEventHandler

The following functions are exported:

- `'get_status'(TimerEventHandler)` -> Reply
  [page 14] Return the status of the target object
- `time_set(TimerEventHandler)` -> Reply
  [page 14] Return true if the time has been set for an event that is yet to be triggered, false otherwise. The outparameter represents the current time value of the target object
- `set_timer(TimerEventHandler, TimeType, TriggerTime)` -> void()
  [page 14] Terminate terminate any previous set trigger, and set a new trigger specified by the TimeType and UTO objects
- `cancel_timer(TimerEventHandler)` -> boolean()
  [page 15] Cancel, if possible, triggering of event(s). Return true if an event is actually cancelled, false otherwise
- `set_data(TimerEventHandler, EventData)` -> ok
  [page 15] Change the event data sent when triggered
CosTimerEvent_TimerEventService

The following functions are exported:

- `register(TimerEventService, CosEventCommPushConsumer, Data) -> TimerEventHandler`
  [page 16] Create a new TimerEventHandler object which push the given Data to given CosEventCommPushConsumer after the timer have been set

- `unregister(TimerEventService, TimerEventHandler) -> ok`
  [page 16] Terminate the target TimerEventHandler object

- `event_time(TimerEventService, TimerEvent) -> UTO`
  [page 16] Return a UTO containing the time at which the associated event was triggered

cosTime

The following functions are exported:

- `install_time() -> Return`
  [page 17] Install the cosTime Time Service part application

- `uninstall_time() -> Return`
  [page 17] Uninstall the cosTime Time Service part application

- `install_timerevent() -> Return`
  [page 17] Install the cosTime Timer Event Service part application

- `uninstall_timerevent() -> Return`
  [page 18] Uninstall the cosTime Timer Event Service part application

- `start() -> Return`
  [page 18] Start the cosTime application

- `stop() -> Return`
  [page 18] Stop the cosTime application

- `start_time_service(Tdf, Inaccuracy) -> Return`
  [page 18] Start a Time Service object

- `stop_time_service(TimeService) -> ok`
  [page 18] Stop the target Time Service object

- `start_timerevent_service(TimeService) -> ok`
  [page 18] Start a Timer Event Service object

- `stop_timerevent_service(TimerEventService) -> ok`
  [page 19] Stop the target Timer Event Service object
CosTime_TIO

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosTime/include/*.hrl").

Exports

‘_get_time_interval'(TIO) -> TimeInterval
Types:
- TIO = #objref
- TimeInterval = #TimeBase.IntervalT(lower_bound, upper_bound)
  lower_bound = upper_bound = ulonglong
This operation returns the interval associated with the target object.

spans(TIO, UTO) -> Reply
Types:
- TIO = UTO = OtherTIO = #objref
- Reply = {OverlapType, OtherTIO}
  OverlapType = 'OTContainer' | 'OTContained' | 'OTO verlap' | 'OTN oO verlap'
This operation returns a OverlapType depending on how the interval in the target object and the timerange represented by the UTO object overlap. If the OverlapType is 'OTN oO verlap' the out parameter represents the gap between the two intervals. If OverlapType is one of the others, the out parameter represents the overlap interval. The definitions of the OverlapType's are:
- 'OTContainer' - target objects lower and upper limits are, respectively, less or equal to and greater or equal to given object's.
- 'OTContained' - target objects lower and upper limits are, respectively, greater or equal to and less or equal to given object's.
- 'OTO verlap' - target objects interval overlap given object's.
- 'OTN oO verlap' - target objects interval do not overlap given object's.

overlaps(TIO, OtherTIO) -> Reply
Types:
- TIO = OtherTIO = AnotherTIO = #objref
- Reply = {OverlapType, AnotherTIO}
  OverlapType = 'OTContainer' | 'OTContained' | 'OTO verlap' | 'OTN oO verlap'
This operation returns an `OverlapType` depending on how the interval in the target object and the timerange represented by the TIO object overlap. The `OverlapType`'s are described under `spans/2`.

time(TIO) -> UTO

Types:
- `TIO = UTO = #objref`

This operation returns a `UTO` in which the interval equals the time interval in the target object and time value is the midpoint of the interval.
CosTime_TimeService

Erlang Module

To get access to the record definitions for the structures use:
~include_lib("cosTime/include/*.hrl").

Exports

universal_time(TimeService) -> Reply
Types:
  * TimeService = #objcref
  * Reply = UTO | {'EXCEPTION', #TimerService_TimeUnavailable'}
  * UTO = #objcref

This operation returns the current time and the Inaccuracy given when starting this application in a UTO. The time base is 15 october 1582 00:00. Comparing two time objects which use different time base is, by obvious reasons, pointless.

new_universal_time(TimeService, Time, Inaccuracy, Tdf) -> UTO
Types:
  * TimeService = UTO = #objcref
  * Time = Inaccuracy = ulonglong()
  * Tdf = short()

This operation creates a new UTO object representing the time parameters given. This is the only way to create a UTO with an arbitrary time from its components. This is useful when using the Timer Event Service.

uto_from_utc(TimeService, Utc) -> UTO
Types:
  * TimeService = UTO = #objcref
  * Utc = #TimeBase_UtcT 'time, inacclo, inacchi, tdf'
  * time = ulonglong()
  * inacclo = ulong()
  * inacchi = ushort()
  * tdf = short()

This operation is used to create a UTO given a time in the Utc form.

new_interval(TimeService, Lower, Upper) -> TIO
Types:
- TimeService = TIO = #objref
- Lower = Upper = ulonglong()

This operation is used to create a new TIO object, representing the input parameters. If Lower is greater than Upper BAD_PARAM is raised.
CosTime_UTO

Erlang Module

To get access to the record definitions for the structures use:
~include_lib("cosTime/include/*.hrl").

Exports

'get_time'(UTO) -> ulonglong()
Types:
  • UTO = #objref
This operation returns the time associated with the target object.

'get_inaccuracy'(UTO) -> ulonglong()
Types:
  • UTO = #objref
This operation returns the inaccuracy associated with the target object.

'get_tdf'(UTO) -> short()
Types:
  • UTO = #objref
This operation returns the time displacement factor associated with the target object.

'get_utc_time'(UTO) -> UtcT
Types:
  • UTO = #objref
  • Utc = #TimeBase_UtcT{time, inacclo, inacchi, tdf}
    • time = ulonglong()
    • inacclo = ulong()
    • inacchi = ushort()
    • tdf = short()
This operation returns the data associated with the target object in Utc form.

absolute_time(UTO) -> OtherUTO
Types:
  • UTO = OtherUTO = #objref
This operation creates a new UTO object representing the time in the target object added to current time (UTC). The time base is 15 October 1582 00:00. Comparing two time objects which use different time base is, by obvious reasons, pointless. Raises DATA_CONVERSION if causes an overflow. This operation is only useful if the target object represents a relative time.

\[
\text{compare\_time}(\text{UTO, ComparisonType, OtherUTO}) \rightarrow \text{Reply}
\]

**Types:**
- \( \text{UTO} = \text{OtherUTO} = \#\text{objref} \)
- \( \text{ComparisonType} = \text{'IntervalC'} \mid \text{'MidC'} \)
- \( \text{Reply} = \text{'TCEqualTo'} \mid \text{'TCLessThan'} \mid \text{'TCGreaterThan'} \mid \text{'TCIndeterminate'} \)

This operation compares the time associated with the target object and the given UTO object. The different ComparisonType are:

- \text{‘MidC’} - only compare the time represented by each object. Furthermore, the target object is always used as the first parameter in the comparison, i.e., if the target object’s time is larger ‘TCGreaterThan’ will be returned.
- \text{‘IntervalC’} - also takes the inaccuracy into consideration, i.e., if the two objects interval overlaps ‘TCIndeterminate’ is returned, otherwise the as for ‘MidC’.

\[
\text{time\_to\_interval}(\text{UTO, OtherUTO}) \rightarrow \text{TIO}
\]

**Types:**
- \( \text{UTO} = \text{OtherUTO} = \text{TIO} = \#\text{objref} \)

This operation returns a TIO representing the interval between the target object and the given UTO midpoint times. The inaccuracy in the objects are not taken into consideration.

\[
\text{interval}(\text{UTO}) \rightarrow \text{TIO}
\]

**Types:**
- \( \text{UTO} = \text{TIO} = \#\text{objref} \)

This operation creates a TIO object representing the error interval around the time value represented by the target object, i.e., \( \text{TIO.upper\_bound} = \text{UTO.time} + \text{UTO.inaccuracy} \) and \( \text{TIO.lower\_bound} = \text{UTO.time} - \text{UTO.inaccuracy} \).
CosTimerEvent_TimerEventHandler

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosTime/include/*.hrl").

Exports

\texttt{\_get\_status}(\texttt{TimerEventHandler}) \rightarrow \texttt{Reply}

Types:
- \texttt{TimerEventHandler} = \#objref
- \texttt{Reply} = 'ESTimeSet' | 'ESTimeCleared' | 'ESTriggered' | 'ESFailedTrigger'

This operation returns the status of the target object.

- 'ESTimeSet' - timer is set to trigger event(s).
- 'ESTimeCleared' - no time set or the timer have been reset.
- 'ESTriggered' - event has already been sent.
- 'ESFailedTrigger' - tried to, but failed, sending the event.

If the target object is of type 'TTPeriodic' the status value 'ESTriggered' is not valid.

\texttt{time\_set}(\texttt{TimerEventHandler}) \rightarrow \texttt{Reply}

Types:
- \texttt{TimerEventHandler} = \#objref
- \texttt{Reply} = (\texttt{boolean()}, \texttt{UTO})
- \texttt{UTO} = \#objref

This operation returns \texttt{true} if the time has been set for an event that is yet to be triggered, \texttt{false} otherwise. The outparameter represents the current time value of the target object.

\texttt{set\_timer}(\texttt{TimerEventHandler}, \texttt{TimeType}, \texttt{TriggerTime}) \rightarrow \texttt{void()}

Types:
- \texttt{TimerEventHandler} = \#objref
- \texttt{TimeType} = 'TTAbsolute' | 'TTRelative' | 'TTPeriodic'
- \texttt{TriggerTime} = \texttt{UTO}
- \texttt{UTO} = \#objref
This operation terminates any previous set trigger, and set a new trigger specified by the TimeType and UTO objects.

The relation between the UTO object and the TimeTypes are:

- ‘TTAbsolute’ - the UTO object must represent absolute time, i.e., number of 100 nanoseconds passed since 15 October 1582 00:00.
- ‘TTRelative’ - the UTO object must represent the from now until when the event should be triggered, e.g., within 30*10^-7 nanoseconds.
- ‘TTPeriodic’ - the same as for ‘TTRelative’, but this option will trigger an event periodically until timer cancelled.

`cancel_timer(TimerEventHandler) -> boolean()`

Types:

- TimerEventHandler = #objref

This operation cancels, if possible, the triggering of event(s). Returns true if an event is actually cancelled, false otherwise.

`set_data(TimerEventHandler, EventData) -> ok`

Types:

- TimerEventHandler = #objref
- EventData = #any

This operation changes the event data sent when triggered.
CosTimerEvent_TimerEventService

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosTime/include/*.hrl").

Exports

register(TimerEventService, CosEventCommPushConsumer, Data) -> TimerEventHandler

Types:
- TimerEventService = CosEventCommPushConsumer = TimerEventHandler = #objref
  Data = #any
This operation will create a new TimerEventHandler object which will push given Data
to given CosEventCommPushConsumer after the timer have been set.

unregister(TimerEventService, TimerEventHandler) -> ok

Types:
- TimerEventService = TimerEventHandler = #objref
This operation will terminate the given TimerEventHandler.

event_time(TimerEventService, TimerEvent) -> UTO

Types:
- TimerEventService = #objref
- TimerEvent = #CosTimerEvent_TimerEvent’{utc, event_data}
  utc =
  event_data = #any
  UTO = #objref
This operation returns a UTO containing the time at which the associated event was
triggered.
cosTime

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosTime/include/*.hrl").

This module contains the functions for starting and stopping the application.
This application use the time base 15 october 1582 00:00. Performing operations using
other time bases will not yield correct result.
The OMG CosTime specification defines the operation secure_universal_time. As of
today we cannot provide this functionality considering the criteria demanded to fulfill
the OMG specification.

When using this application, time and inaccuracy supplied by the user must be given in
number of 100 nano seconds. The Time Displacement Factor is positive east of the
meridian, while those to the west are negative.
This application use calendar:now_to_universal_time(Now) to create a UTC. Hence,
the underlying OS must deliver a correct result when calling erlang:now().
When determining the inaccuracy of the system, the user should consider the way the
time objects will be used. Communicating with other ORB’s, add a substantial
overhead and should be taken into consideration.

Exports

install_time() -> Return
Types:
  • Return = ok | 'EXIT', Reason
This operation installs the cosTime Time Service part application.

uninstall_time() -> Return
Types:
  • Return = ok | 'EXIT', Reason
This operation uninstalls the cosTime Time Service part application.

install_timerevent() -> Return
Types:
  • Return = ok | 'EXIT', Reason
This operation installs the cosTime Timer Event Service part application.

**Note:**
The Timer Event Service part requires Time Service part and cosEvent or the cosNotification application to be installed first.

```plaintext
uninstall_timerevent() -> Return

Types:
- Return = ok | {'EXIT', Reason}

This operation uninstalls the cosTime Timer Event Service part application.
```

```plaintext
start() -> Return

Types:
- Return = ok | {error, Reason}

This operation starts the cosTime application.
```

```plaintext
stop() -> Return

Types:
- Return = ok | {error, Reason}

This operation stops the cosTime application.
```

```plaintext
start_time_service(Tdf, Inaccuracy) -> Return

Types:
- Tdf = short()
- Inaccuracy = ulonglong(), eq. #100 nano seconds
- Return = ok | {'EXCEPTION', #BAD_PARAM'}

This operation starts a Time Service server. Please note that there may only be exactly one Time Service active at a time. The Inaccuracy parameter defines the inaccuracy the underlying OS will introduce. Remember to take into account latency when passing time object between nodes.
```

```plaintext
stop_time_service(TimeService) -> ok

Types:
- TimeService = #objref

This operation stops the Time Service object.
```

```plaintext
start_timerevent_service(TimeService) -> ok

Types:
- TimeService = #objref
```
This operation starts a Timer Event Service server. Please note that there may only be exactly one Timer Event Service per node active at a time. The supplied TimeServcie reference will be the object Timer Event Service contacts to get access to a new UTC.

```plaintext
stop_timer_event_service(TimerEventService) -> ok
```

Types:
- TimerEventService = #objref

This operation stops the Timer Event Service object.
Index of Modules and Functions

Modules are typed in this way.
Functions are typed in this way.

'_get_inaccuracy'/1
   CosTime_U_TO , 12
'_get_status'/1
   CosTimerEvent_TimerEventHandler , 14
'_get_tdf'/1
   CosTime_U_TO , 12
'_get_time'/1
   CosTime_U_TO , 12
'_get_time_interval'/1
   CosTime_TIO , 8
'_get.utc_time'/1
   CosTime_U_TO , 12
absolute_time/1
   CosTime_U_TO , 12
cancel_timer/1
   CosTimerEvent_TimerEventHandler , 15
compare_time/3
   CosTime_U_TO , 13
cosTime
   install_time/0, 17
   install_timer_event/0, 17
   start/0, 18
   start_time_service/2, 18
   start_timer_event_service/1, 18
   stop/0, 18
   stop_time_service/1, 18
   stop_timer_event_service/1, 19
   uninstall_time/0, 17
   uninstall_timer_event/0, 18
CosTime_TimeService
   new_interval/3, 10
   new_universal_time/4, 10
   universal_time/1, 10
   uto_from_utc/2, 10

CosTime_TIO
   '_get_time_interval'/1, 8
   overlaps/2, 8
   spans/2, 8
   time/1, 9

CosTime_U_TO
   '_get_inaccuracy'/1, 12
   '_get_tdf'/1, 12
   '_get_time'/1, 12
   '_get_utc_time'/1, 12
   absolute_time/1, 12
   compare_time/3, 13
   interval/1, 13
   time_to_interval/2, 13

CosTimerEvent_TimerEventHandler
   '_get_status'/1, 14
   cancel_timer/1, 15
   set_data/2, 15
   set_timer/3, 14
   time_set/1, 14

CosTimerEvent_TimerEventService
   event_time/2, 16
   register/3, 16
   unregister/2, 16

event_time/2
   CosTimerEvent_TimerEventService , 16

install_time/0
   cosTime , 17
install_timer_event/0
   cosTime , 17

interval/1
   CosTime_U_TO , 13

new_interval/3
   CosTime_TimeService , 10
new_universal_time/4
  CosTime_TimeService, 10

overlaps/2
  CosTime_TIO, 8

register/3
  CosTimerEvent_TimerEventService, 16

set_data/2
  CosTimerEvent_TimerEventHandler, 15

set_timer/3
  CosTimerEvent_TimerEventHandler, 14

spans/2
  CosTime_TIO, 8

start/0
  cosTime, 18

start_time_service/2
  cosTime, 18

start_timerevent_service/1
  cosTime, 18

stop/0
  cosTime, 18

stop_time_service/1
  cosTime, 18

stop_timerevent_service/1
  cosTime, 19

time/1
  CosTime_TIO, 9

time_set/1
  CosTimerEvent_TimerEventHandler, 14

time_to_interval/2
  CosTime_UTO, 13

uninstall_time/0
  cosTime, 17

uninstall_timerevent/0
  cosTime, 18

universal_time/1
  CosTime_TimeService, 10

unregister/2
  CosTimerEvent_TimerEventService, 16

uto_from_utc/2
  CosTime_TimeService, 10