rexroth
A Bosch Company

NYCe 4000

Linear motion system
Software Reference Manual

Reference
R911425760

Edition 02

Title

NYCe 4000
Linear motion system
Software Reference Manual

Type of Documentation

Reference

Document Typecode

DOK-NY4000-LMS*REF*V54-RE02-EN-E

Internal File Reference

RS-9e1f909b7fc815c50a347e8628a5fb74-2-en-US-3

Record of Revision

Edition	Release Date	Notes
DOK-NY4000-LMS*REF*V54-RE01-EN-E	08/2024	First edition 54V02
DOK-NY4000-LMS*REF*V54-RE02-EN-E	04/2025	2nd edition 54V04

Published by

Bosch Rexroth AG

P.O. Box 7170 ■ 5605 JD Eindhoven ■ The Netherlands

http://www.boschrexroth.com

Contents

1 Carrier Management

Carrier Management (CM) contains the functions available to drive the Linear Motion System (LMS).

1.1 Connection

The functions within this module are used during connection to CM.

1.1.1 CmGetCarriersOnTrack

```
NYCE_STATUS CmGetCarriersOnTrack(CM_INT    trackId,
                                 CM_INT    carrierIds[carrierIdsLength],
                                 CM_UDINT  carrierIdsLength,
                                 CM_UDINT* pNrOfCarriers);
```

Get the carriers on track.

Return the number of carriers on the specified track along with their carrier identifiers. The identifiers are stored in the output array. The carrier identifiers are ordered by increasing carrier position. The identifier of the carrier with the lowest position is stored at array index 0. The identifier of non existing carriers or carriers for which no identifier is assigned is set to CM_NO_ID.

Parameters:

in	trackId	Identification of track.
out	carrierIds	Carrier identifiers.
in	carrierIdsLength	Allocated size of the array carrierIds in number of elements.
out	pNrOfCarriers	Number of available carriers.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_ERROR
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_NR_OF_CARRIERS_UNKNOWN
- CM_WRN_CARRIER_IDS_NOT_ASSIGNED
- NYCE_ERR_INVALID_OUTPUT_SIZE
  - (carrierIdsLength is zero) or (carrierIdsLength greater than CM_MAX_NR_OF_CARRIERS) or (number of carriers greater than carrierIdsLength)

1.1.2 CmGetTracks

```
NYCE_STATUS CmGetTracks(CM_INT    trackIds[trackIdsLength],
                        CM_UDINT  trackIdsLength,
                        CM_UDINT* pNrOfTracks);
```

Get the available tracks.

Return the number of tracks along with their identifiers. The identifiers are stored in the output array. The identifier of non existing tracks is CM_NO_ID.

Parameters:

out	trackIds	Track identifiers.

Carrier Management

in	trackIdsLength	Allocated size of the array trackIds in number of elements.
out	pNrOfTracks	Number of available tracks.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_ERROR
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- NYCE_ERR_INVALID_OUTPUT_SIZE
  - (trackIdsLength is zero) or (trackIdsLength greater than CM_MAX_NR_OF_TRACKS) or (number of tracks greater than trackIdsLength)

1.2 Configuration

The functions within this module are used to specify the system configuration.

1.2.1 CmInitialize

```
NYCE_STATUS CmInitialize(const char* machineDataFile);
```

Initialize the LMS.

Parameters:

in	machineDataFile	Name of the machine data file.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_ERROR
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_FILE_OPEN_ERROR
- CM_ERR_FILE_CONTENTS_ERROR
- CM_ERR_INVALID_PARAMETER

State:

CM_SYSTEM_POWERUP

Transition:

CM_SYSTEM_POWERUP -> CM_SYSTEM_INITIALIZED

All tracks: CM_TRACK_POWERUP -> CM_TRACK_IDLE

1.2.2 CmShutdown

```
NYCE_STATUS CmShutdown();
```

Shut down the LMS.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED

- CM_ERR_SYSTEM_ERROR
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_HOME_TRACK_EXECUTING

State:

CM_SYSTEM_INITIALIZED

Transition:

CM_SYSTEM_INITIALIZED -> CM_SYSTEM_POWERUP

All tracks: current state -> CM_TRACK_POWERUP

1.3 Runtime Track Layout Changes (RTLC)

The functions within this module are used to modify the track layout by adding or removing coils.

1.3.1 CmAddFirstCoil

```
NYCE_STATUS CmAddFirstCoil(CM_INT trackId);
```

Add the first coil or set of coils to a track.

The first coil or set of coils to be added must not be in use of any track.

When command CmAddFirstCoil() is successfully completed, the first coil or set of coils is added to the track and the track begin position is updated. The carrier controlled by the added coil or carrier(s) controlled by the set of added coils is/are also added to the track. The added coil(s) adopts the feed override and start trigger settings of the track.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_COIL_ALREADY_ADDED
- CM_ERR_MOVE_TO_TRACK_EXECUTING

State:

CM_TRACK_IDLE or CM_TRACK_ACTIVE

if CM_TRACK_IDLE: coil state is SAC_INACTIVE

if CM_TRACK_ACTIVE: coil state is SAC_FREE (if coil is not controlling a carrier)

if CM_TRACK_ACTIVE: coil state is SAC_READY (if coil is controlling a carrier)

Transition:

CM_TRACK_IDLE -> CM_TRACK_FATAL_ERROR (if an added coil is in error due to execution of error handler with severity 2..9)

CM_TRACK_ACTIVE -> CM_TRACK_ERROR (if an added coil is in error due to execution of error handler with severity 2..7)

CM_TRACK_ACTIVE -> CM_TRACK_FATAL_ERROR (if an added coil is in error due to execution of error handler with severity 8 or 9)

none (if added coil(s) is/are not in error)

Carrier Management

1.3.2 CmAddLastCoil

```
NYCE_STATUS CmAddLastCoil(CM_INT trackId);
```

Add the last coil or set of coils to a track.

The last coil or set of coils to be added must not be in use of any track.

When command CmAddLastCoil() is successfully completed, the last coil or set of coils is added to the track and the track end position is updated. The carrier controlled by the added coil or carrier(s) controlled by the set of added coils is/are also added to the track. The added coil(s) adopts the feed override and start trigger settings of the track.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_COIL_ALREADY_ADDED
- CM_ERR_MOVE_TO_TRACK_EXECUTING

State:

CM_TRACK_IDLE or CM_TRACK_ACTIVE

if CM_TRACK_IDLE: coil state is SAC_INACTIVE

if CM_TRACK_ACTIVE: coil state is SAC_FREE (if coil is not controlling a carrier)

if CM_TRACK_ACTIVE: coil state is SAC_READY (if coil is controlling a carrier)

Transition:

CM_TRACK_IDLE -> CM_TRACK_FATAL_ERROR (if an added coil is in error due to execution of error handler with severity 2..9)

CM_TRACK_ACTIVE -> CM_TRACK_ERROR (if an added coil is in error due to execution of error handler with severity 2..7)

CM_TRACK_ACTIVE -> CM_TRACK_FATAL_ERROR (if an added coil is in error due to execution of error handler with severity 8 or 9)

none (if added coil(s) is/are not in error)

1.3.3 CmRemoveFirstCoil

```
NYCE_STATUS CmRemoveFirstCoil(CM_INT trackId);
```

Remove the first coil or set of coils from a track.

The first coil or set of coils to be removed must be in use by the track.

When command CmRemoveFirstCoil() is successfully completed, the first coil or set of coils is removed from the track and the track begin position is updated. The carrier controlled by the removed coil or carrier(s) controlled by the set of removed coils is/are also removed from the track.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED

- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_COIL_ALREADY_REMOVED
- CM_ERR_UNABLE_TO_REMOVE_COIL
- CM_ERR_MOVE_TO_TRACK_EXECUTING

State:

CM_TRACK_IDLE or CM_TRACK_ACTIVE

if CM_TRACK_ACTIVE: CM_CARRIER_STOPPED (for a carrier controlled by a removed coil)

1.3.4 CmRemoveLastCoil

```
NYCE_STATUS CmRemoveLastCoil(CM_INT trackId);
```

Remove the last coil or set of coils from a track.

The last coil or set of coils to be removed must be in use by the track.

When command CmRemoveLastCoil() is successfully completed, the last coil or set of coils is removed from the track and the track end position is updated. The carrier controlled by the removed coil or carrier(s) controlled by the set of removed coils is/are also removed from the track.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_COIL_ALREADY_REMOVED
- CM_ERR_UNABLE_TO_REMOVE_COIL
- CM_ERR_MOVE_TO_TRACK_EXECUTING

State:

CM_TRACK_IDLE or CM_TRACK_ACTIVE

if CM_TRACK_ACTIVE: CM_CARRIER_STOPPED (for a carrier controlled by a removed coil)

1.3.5 CmSelectFirstCoilAlternative

```
NYCE_STATUS CmSelectFirstCoilAlternative(CM_INT trackId,
                                         CM_INT alternative);
```

Select a specific alternative for use at the beginning of the track.

Parameters:

in	trackId	Identification of track.
in	alternative	Which alternative is selected.

Preconditions:

- The involved coils are not in use by any track.
- The specified track is in the state: CM_TRACK_IDLE or CM_TRACK_ACTIVE.
- If the coils are controlling a carrier, it must be in the CM_CARRIER_STOPPED state.

Postconditions:

- When command CmSelectFirstCoilAlternative() is successfully completed, the specified alternative is selected for use on the specified track.
- If the coils are controlling a carrier, this remains the case. NOTE: The coils are not yet added to the track. NOTE2: For RTLC implementations with only one physical coil alternative for all shared coil positions it is not necessary to use this function. The default selected alternative will be 0.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_COIL_ALTERNATIVE
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR

State:

CM_TRACK_IDLE or CM_TRACK_ACTIVE

1.3.6 CmSelectLastCoilAlternative

```
NYCE_STATUS CmSelectLastCoilAlternative(CM_INT trackId,
                                        CM_INT alternative);
```

Select a specific alternative for use at the end of the track.

Parameters:

in	trackId	Identification of track.
in	alternative	Which alternative is selected.

Preconditions:

- The involved coils are not in use by any track.
- The specified track is in the state: CM_TRACK_IDLE or CM_TRACK_ACTIVE.
- If the coils are controlling a carrier, it must be in the CM_CARRIER_STOPPED state.

Postconditions:

- When command CmSelectLastCoilAlternative() is successfully completed, the specified alternative is selected for use on the specified track.
- If the coils are controlling a carrier, this remains the case. NOTE: The coils are not yet added to the track. NOTE2: For RTLC implementations with only one physical coil alternative for all shared coil positions it is not necessary to use this function. The default selected alternative will be 0.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_COIL_ALTERNATIVE

- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR

State:

CM_TRACK_IDLE or CM_TRACK_ACTIVE

1.4 Homing

The functions within this module are related to homing of the track.

1.4.1 CmAssignCarrierIds

```
NYCE_STATUS CmAssignCarrierIds(CM_INT       trackId,
                               const CM_INT carrierIds[nrOfCarriers],
                               CM_UDINT     nrOfCarriers);
```

Assign identifiers to the carriers on a track.

The identifiers from the array are assigned in the order of increasing position.

Parameters:

in	trackId	Identification of track.
in	carrierIds	Carrier identifiers.
in	nrOfCarriers	Number of carrier identifiers in carrierIds.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_PARAMETER_OUT_OF_RANGE
- CM_ERR_PARAMETER_ERROR
  - (carrier id in array not unique within LMS) or (carrier id in array invalid)
- CM_ERR_WRONG_NR_OF_CARRIERS
  - (nrOfCarriers < actual number of carriers on track)

State:

CM_TRACK_HOMED or CM_TRACK_INACTIVE

Transition:

CM_TRACK_HOMED -> CM_TRACK_INACTIVE

1.4.2 CmCheckArea

```
NYCE_STATUS CmCheckArea(CM_INT         trackId,
                        const CM_AREA* pCmArea,
                        CM_BOOL*       pCarrierPresent);
```

Check the presence of a carrier in an area.

Check if one of the sensors in the specified area of the track detects the presence of a carrier. The result of this check is returned in pCarrierPresent.

Carrier Management

Parameters:

in	trackId	Identification of track.
in	pCmArea	Area description.
out	pCarrierPresent	Is a carrier present in the specified area.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_ERROR
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_NO_SENSOR_IN_AREA
- CM_ERR_INVALID_POSITIONS
- CM_ERR_INVALID_MIN_POSITION
- CM_ERR_INVALID_MAX_POSITION
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.4.3 CmEnableBumper

```
NYCE_STATUS CmEnableBumper(CM_INT  trackId,
                           CM_INT  bumperId,
                           CM_BOOL onOff);
```

Enable or disable the bumper active status.

Parameters:

in	trackId	Identification of track.
in	bumperId	Identification of bumper.
in	onOff	Indicating if bumper is enabled (TRUE) or disabled (FALSE).

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_BUMPER_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_HOME_TRACK_EXECUTING

1.4.4 CmFreeArea

```
NYCE_STATUS CmFreeArea(CM_INT                    trackId,
                       const CM_FREE_AREA_PARS* pCmFreeAreaPars);
```

Move the carriers inside an area of a track until a part of that area is freed from carriers.

Parameters:

in	trackId	Identification of track.
in	pCmFreeAreaPars	Free area parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_FREE_AREA_SIZE
- CM_ERR_INVALID_BEGIN_POSITION
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_NO_COIL_IN_AREA
- CM_ERR_NO_SENSOR_IN_AREA
- CM_ERR_NO_FREE_SENSOR_OUTSIDE_AREA
- CM_ERR_CARRIER_MOVED_OUTSIDE_AREA
- CM_ERR_NO_MOVING_CARRIER_IN_AREA
- CM_ERR_ASYNC_ERROR_OCCURRED
- CM_ERR_MR_SENSOR_COMMUTATION_NOT_DETERMINED

Transition:

current state -> CM_TRACK_VELOCITY_MODE -> CM_TRACK_IDLE

1.4.5 CmHomeTrack

```
NYCE_STATUS CmHomeTrack(CM_INT trackId,
                        CM_INT direction);
```

Start the homing procedure for a track.

The procedure determines the number of carriers and their positions on the track. The number of carriers can be retrieved by a call to CmGetCarriersOnTrack().

Parameters:

in	trackId	Identification of track.
in	direction	Direction of home search (+1 = positive, -1 = negative).

Retvals:

- NYCE_OK
  - homing successfully started
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR
  - direction not +1 or -1
- CM_ERR_HOME_TRACK_EXECUTING
- CM_ERR_MOVE_TO_ENDSTOP_EXECUTING
- CM_ERR_MOVE_TO_TRACK_EXECUTING

Transition:

current state -> CM_TRACK_IDLE -> CM_TRACK_HOMED

Remarks:

The actual homing action is performed in a separate thread because it can take quite some time. The following asynchronous errors can occur:

- CM_ERR_NOT_END_OF_STROKE
- CM_ERR_INVALID_CARRIER_ID

Carrier Management

- CM_ERR_INVALID_CARRIER_INDEX
- CM_ERR_INVALID_POSITION
- CM_ERR_HOME_MODE_7_IMPOSSIBLE
- CM_ERR_HOME_MODE_8_IMPOSSIBLE
- CM_ERR_PRECONDITION_NOT_OK
- CM_ERR_HOME_MODE_8_FAILED
- CM_ERR_HOME_MODE_9_FAILED
- CM_ERR_HOME_MODE_10_FAILED
- CM_ERR_TOO_MANY_CARRIERS_ON_TRACK
- CM_ERR_CARRIER_POSITION_OUTSIDE_SENSOR_RANGE
- CM_ERR_TIMEOUT
- CM_ERR_SENSOR_STATUS_CRITERION
- CM_ERR_CARRIER_BUMPER_INTERSECTION
- CM_ERR_INACTIVE_SENSOR
- CM_ERR_NO_FREE_SENSORS_TO_MOVE
- CM_ERR_NO_FREE_COIL_BETWEEN_CARRIERS
- CM_ERR_CARRIER_AGAINST_ENDSTOP
- CM_ERR_NR_OF_CARRIERS_INCONSISTENT
- CM_ERR_ASYNC_ERROR_OCCURRED
- CM_ERR_CARRIERS_NOT_MOVING
- CM_ERR_MR_SENSOR_COMMUTATION_NOT_DETERMINED

1.4.6 CmReadBumperPars

```
NYCE_STATUS CmReadBumperPars(CM_INT          trackId,
                             CM_INT          bumperId,
                             CM_BUMPER_PARS* pCmBumperPars);
```

Read the parameters of a bumper.

Parameters:

in	trackId	Identification of track.
in	bumperId	Identification of bumper.
out	pCmBumperPars	Bumper parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_BUMPER_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.4.7 CmReadHomePars

```
NYCE_STATUS CmReadHomePars(CM_INT         trackId,
                           CM_HOME_PARS* pCmHomePars);
```

Read the home parameters of a track.

Parameters:

in	trackId	Identification of track.
out	pCmHomePars	Home parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.4.8 CmWriteHomePars

```
NYCE_STATUS CmWriteHomePars(CM_INT              trackId,
                            const CM_HOME_PARS* pCmHomePars);
```

Write a set of home parameters for a track.

Parameters:

in	trackId	Identification of track.
in	pCmHomePars	Home parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_PARAMETER
- CM_ERR_HOME_TRACK_EXECUTING

1.5 Carrier Position Accuracy (CPA)

The functions within this module are related CPA.

1.5.1 CmDownloadCpaTable

```
NYCE_STATUS CmDownloadCpaTable(const char* fileName,
                               CM_INT      tableId);
```

Download a CPA table from a file.

Parameters:

in	fileName	XML-file name.
in	tableId	Identification of CPA table.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_PARAMETER_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_FILE_CONTENTS_ERROR
- CM_ERR_FILE_OPEN_ERROR

- CM_ERR_FILE_READ_ERROR

1.5.2 CmEnableCpa

```
NYCE_STATUS CmEnableCpa(CM_BOOL onOff);
```
Enable or disable the CPA functionality.

Parameters:

in	onOff	Indicating if CPA is enabled (TRUE) or disabled (FALSE).

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_PARAMETER_ERROR

1.5.3 CmLinkCpaTable

```
NYCE_STATUS CmLinkCpaTable(CM_INT carrierId,
                           CM_INT tableId);
```
Link a carrier to a CPA table.

Parameters:

in	carrierId	Identification of carrier.
in	tableId	Identification of CPA table.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_PARAMETER_ERROR
- CM_ERR_INVALID_CARRIER_ID

1.5.4 CmUnlinkCpaTable

```
NYCE_STATUS CmUnlinkCpaTable(CM_INT carrierId);
```
Unlink a carrier from a CPA table.

Parameters:

in	carrierId	Identification of carrier.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID

1.6 Track

The functions within this module operate at track level.

1.6.1 CmActivateTrack

```
NYCE_STATUS CmActivateTrack(CM_INT trackId);
```

Activate a track.

After successful completion of the command, each coil on the track starts operating in closed loop when a carrier is, or arrives in the coil control range.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_NR_OF_CARRIERS_INCONSISTENT
- CM_ERR_CARRIER_POSITION_ERROR
  - Carrier not in the control range of a coil
- CM_ERR_DUPLICATE_CARRIER_ID

State:

CM_TRACK_INACTIVE

Transition:

CM_TRACK_INACTIVE -> CM_TRACK_ACTIVE

All carriers: CM_CARRIER_POWERUP -> CM_CARRIER_STOPPED

1.6.2 CmDeactivateTrack

```
NYCE_STATUS CmDeactivateTrack(CM_INT trackId);
```

Deactivate a track.

After successful completion of the command, each coil on the track is in open loop and the power is disabled.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_MOVE_TO_TRACK_EXECUTING

Carrier Management

State:

CM_TRACK_ACTIVE

Transition:

CM_TRACK_ACTIVE -> CM_TRACK_INACTIVE

All carriers: current state -> CM_CARRIER_POWERUP

1.6.3 CmEcgLink

```
NYCE_STATUS CmEcgLink(CM_INT                trackId,
                      const SAC_ECG_MASTER* pSacEcgMaster);
```

Establish a link between the coils of the track and the master axis.

Parameters:

in	trackId	Identification of track.
in	pSacEcgMaster	Master description.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_PARAMETER
- CM_ERR_MASTER_AXIS_TYPE_NOT_SUPPORTED
- CM_ERR_ECG_SAMPLE_FREQUENCY_HIGHER_THAN_1KHZ

1.6.4 CmEcgResetMasterModuloValue

```
NYCE_STATUS CmEcgResetMasterModuloValue(CM_INT trackId);
```

Reset modulo value of master position for the coils of the track.

Parameters:

in	trackId	Identification of track.

Preconditions:

The master modulo parameter can only be changed when a master is linked, and the identified cam table has been loaded, and the track has no camming carriers.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID

1.6.5 CmEcgSetMasterModuloValue

```
NYCE_STATUS CmEcgSetMasterModuloValue(CM_INT   trackId,
                                      CM_UDINT tableId);
```

Set modulo value of master position for the coils of the track.

Parameters:

in	trackId	Identification of track.
in	tableId	Identifier of cam table.

Preconditions:

The master modulo parameter can only be changed when a master is linked, and the identified cam table has been loaded, and the track has no camming carriers.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID

1.6.6 CmEcgSetMasterPos

```
NYCE_STATUS CmEcgSetMasterPos(CM_INT   trackId,
                              CM_LREAL masterPos);
```

Set position of master axis for the coils of the track.

Parameters:

in	trackId	Identification of track.
in	masterPos	Position of master axis.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID

1.6.7 CmEcgSetMasterSign

```
NYCE_STATUS CmEcgSetMasterSign(CM_INT         trackId,
                               SAC_DIRECTION masterSign);
```

Set sign of master displacement for the coils of the track.

Parameters:

in	trackId	Identification of track.
in	masterSign	Master sign.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR

Carrier Management

1.6.8 CmEcgUnlink

```
NYCE_STATUS CmEcgUnlink(CM_INT trackId);
```

Remove the link between the coils of the track and the master axis.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID

1.6.9 CmEnableStartTrigger

```
NYCE_STATUS CmEnableStartTrigger(CM_INT  trackId,
                                 CM_BOOL onOff);
```

Enable or disable the start on trigger for all coils of the track.

Parameters:

in	trackId	Identification of track.
in	onOff	Indicating if start on trigger must be enabled (TRUE) or disabled (FALSE).

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID

Transition:

CM_CARRIER_MOVING -> CM_CARRIER_STOPPED (if carrier is not physically moving and trigger is disabled)

1.6.10 CmSetFeedOverride

```
NYCE_STATUS CmSetFeedOverride(CM_INT   trackId,
                              CM_LREAL feedOverride);
```

Change the feed override value for a track.

The command is ignored if the feed override value to be set, equals the current feed override value, while the feed override is constant.

Parameters:

in	trackId	Identification of track.
in	feedOverride	Feed override.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED

- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_SERVER_STATE_ERROR
- CM_ERR_PARAMETER_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_MOVE_TO_TRACK_EXECUTING

State:

any track state

If track state is CM_TRACK_ERROR and CM_TRACK_FATAL_ERROR; feed override must be 0

If track state is CM_TRACK_ACTIVE; no CmMoveToTrack must be busy on the track and all carriers on the track must be in the CM_CARRIER_STOPPED or CM_CARRIER_SUSPENDED state.

Transition:

If CM_TRACK_ACTIVE: CM_CARRIER_SUSPENDED -> CM_CARRIER_MOVING (if feed override from 0 to any other value)

1.6.11 CmSetTrackStateToIdle

```
NYCE_STATUS CmSetTrackStateToIdle(CM_INT trackId);
```

Set the track state to IDLE.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR

State:

CM_TRACK_INACTIVE or CM_TRACK_IDLE

Transition:

CM_TRACK_INACTIVE -> CM_TRACK_IDLE

1.6.12 CmStartVelocityMode

```
NYCE_STATUS CmStartVelocityMode(CM_INT   trackId,
                                CM_LREAL velocity);
```

Start the control of the carriers on the track in velocity mode.

After completion of this command the carriers are moving with the specified velocity.

Parameters:

in	trackId	Identification of track.
in	velocity	Velocity of the carriers.

Retvals:

Carrier Management

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_MOVE_TO_TRACK_EXECUTING
- CM_ERR_MR_SENSOR_COMMUTATION_NOT_DETERMINED

Transition:

current state -> CM_TRACK_VELOCITY_MODE

1.6.13 CmStartVelocityModeCoils

```
NYCE_STATUS CmStartVelocityModeCoils(CM_INT   trackId,
                                     CM_DINT  startCoilIndex,
                                     CM_DINT  endCoilIndex,
                                     CM_LREAL velocity);
```

Start the control of the carriers on the track in velocity mode.

The velocity control loop with specified velocity is activated on a range of successive coils while the velocity control loop with velocity 0 is activated for all other coils of the track.

After completion of this command the carriers are moving with the specified velocity or are held in place with velocity 0.

Parameters:

in	trackId	Identification of track.
in	startCoilIndex	Coil index of start coil of the range of successive coils.
in	endCoilIndex	Coil index of end coil of the range of successive coils.
in	velocity	Velocity of the carriers inside the range of successive coils.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_INVALID_COIL_INDEX
- CM_ERR_MOVE_TO_TRACK_EXECUTING
- CM_ERR_MR_SENSOR_COMMUTATION_NOT_DETERMINED

Transition:

current state -> CM_TRACK_VELOCITY_MODE

1.6.14 CmStopVelocityMode

```
NYCE_STATUS CmStopVelocityMode(CM_INT trackId);
```

Stop the control of the carriers on the track in velocity mode.

When the command is completed the carriers are at standstill.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR

State:

CM_TRACK_VELOCITY_MODE

Transition:

CM_TRACK_VELOCITY_MODE -> CM_TRACK_IDLE

1.7 Carrier

The functions within this module operate at carrier level.

1.7.1 CmAddCarrier

```
NYCE_STATUS CmAddCarrier(CM_INT   carrierId,
                         CM_INT   trackId,
                         CM_LREAL carrierAddPosition);
```

Add a carrier to a track.

The exact position, where the carrier is added, is determined on the basis of the approximate position, input to this function, and the Hall sensor information.

Parameters:

in	carrierId	Identification of carrier.
in	trackId	Identification of track.
in	carrierAddPosition	Approximate position on the track where the carrier is added.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_PARAMETER_ERROR
  - add position out of track range or carrier edge too close to sensor edge
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_TOO_MANY_CARRIERS_ON_TRACK
- CM_ERR_CARRIER_POSITION_OUTSIDE_SENSOR_RANGE
- CM_ERR_SENSOR_DETECTION_NOT_DISABLED

State:

CM_TRACK_INACTIVE or CM_TRACK_ACTIVE

Transition:

Carrier Management

If CM_TRACK_ACTIVE: -> CM_CARRIER_STOPPED (if sensor detection enabled)
If CM_TRACK_ACTIVE: -> CM_CARRIER_POWERUP (if sensor detection disabled)
If CM_TRACK_INACTIVE: -> CM_CARRIER_POWERUP.

1.7.2 CmAddMovingCarrier

```
NYCE_STATUS CmAddMovingCarrier(CM_INT              trackId,
                               CM_INT              carrierId,
                               CM_TRACK_POS        trackPositionId,
                               const CM_MOVE_PARS* pCmMovePars);
```

Add a moving carrier to the system while the track is active.

Call this function to add a moving carrier to the system while the track is active. Ideally the carrier should be moving at a constant velocity. This same velocity should be specified in the movement parameters. The end position specified in the movement parameters should be chosen so the carrier is completely on the track. As soon as the carrier has been detected it will be added to the carrier administration.

Parameters:

in	trackId	Identification of track.
in	carrierId	Identification of carrier.
in	trackPositionId	Identification of track position. Side where the carrier enters the track.
in	pCmMovePars	Movement parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_ADD_MOVING_CARRIER_ALREADY_PENDING
  - a CmAddMovingCarrier is already commanded
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_NOT_SUPPORTED
  - invalid track type
- CM_ERR_TOO_MANY_CARRIERS_ON_TRACK
- CM_ERR_MOVE_TO_TRACK_EXECUTING
- CM_ERR_INVALID_TRACK_POSITION_ID
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_PARAMETER

State:

CM_TRACK_ACTIVE and feed override should not be 0!

Transition:

-> CM_CARRIER_MOVING (if carrier is detected by the outer sensor)

1.7.3 CmCancelAddMovingCarrier

```
NYCE_STATUS CmCancelAddMovingCarrier(CM_INT       trackId,
                                     CM_INT       carrierId,
                                     CM_TRACK_POS trackPositionId);
```

Cancel the addition of a moving carrier to the system.

Call this function to cancel the addition of a moving carrier to the system before this carrier has been added.

Parameters:

in	trackId	Identification of track.
in	carrierId	Identification of carrier.
in	trackPositionId	Identification of beginning or end of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
  - carrier is already added
- CM_ERR_INVALID_TRACK_ID
  - invalid track id specified
- CM_ERR_INVALID_CARRIER_ID
  - invalid carrier id specified
- CM_ERR_INVALID_POSITION
  - invalid position specified

1.7.4 CmCancelMoveAcrossTrackBounds

```
NYCE_STATUS CmCancelMoveAcrossTrackBounds(CM_INT trackId,
                                          CM_INT carrierId);
```

Cancel the movement of a carrier across bounds of a track.

When the command returns successfully, a previously initiated but not started CmMoveAcrosssTrackBounds for the track/carrier combination is canceled.

Parameters:

in	trackId	Identification of track.
in	carrierId	Identification of carrier.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_MOVEMENT_STARTED
  - carrier is executing move across track bounds movement
- CM_ERR_NO_COMMAND_INITIATED
  - no move across track bounds command initiated for carrier

State:

Carrier Management

Any

1.7.5 CmChangeCarrierId

```
NYCE_STATUS CmChangeCarrierId(CM_INT carrierId,
                              CM_INT newCarrierId);
```

Change identifier of a carrier after it has been assigned.

Parameters:

in	carrierId	carrier identifier to be changed.
in	newCarrierId	new carrier identifier.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
  - (carrier id invalid) or (new carrier id not unique within LMS)

State:

CM_TRACK_INACTIVE or CM_TRACK_ACTIVE

CM_CARRIER_POWERUP or CM_CARRIER_STOPPED

Remarks:

When the carrier is used in collision avoidance functions, the carrier may not be in CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS state. In that case a CM_ERR_CARRIER_STATE_ERROR is returned.

1.7.6 CmDefineSyncGroup

```
NYCE_STATUS CmDefineSyncGroup(const CM_INT carrierIds[nrOfCarriers],
                              CM_UDINT     nrOfCarriers,
                              CM_INT*      pGroupId);
```

Define a synchronization group of carriers.

Parameters:

in	carrierIds	Carrier identifiers.
in	nrOfCarriers	Number of carrier identifiers in carrierIds.
out	pGroupId	Identifier of the group.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_CARRIER_ALREADY_BELONGS_TO_A_SYNC_GROUP
- CM_ERR_TOO_MANY_SYNC_GROUPS
- CM_ERR_CARRIER_STATE_ERROR

- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- CM_ERR_PARAMETER_OUT_OF_RANGE
- CM_ERR_TOO_MANY_COILS_IN_SYNC_GROUP
- CM_ERR_DUPLICATE_CARRIER_ID

State:

CM_TRACK_ACTIVE

CM_CARRIER_STOPPED

1.7.7 CmDeleteSyncGroup

```
NYCE_STATUS CmDeleteSyncGroup(CM_INT groupId);
```

Delete a synchronize group.

Parameters:

in	groupId	Identification of group.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_SYNC_GROUP_ID

1.7.8 CmEcgLockCam

```
NYCE_STATUS CmEcgLockCam(CM_INT                        carrierId,
                         const SAC_ECG_LOCK_CAM_PARS* pLockCamPars);
```

Establish the (cam) lock between the carrier and the master axis.

Parameters:

in	carrierId	Identification of carrier.
in	pLockCamPars	Parameters for the locking procedure.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_CARRIER_ID_NOT_FOUND
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_ECG_LOCK_MODE_NOT_SUPPORTED

Carrier Management

1.7.9 CmGetAxisIdOfCoil

```
NYCE_STATUS CmGetAxisIdOfCoil(CM_INT    trackId,
                              CM_DINT   coilIndex,
                              SAC_AXIS* pAxisId);
```

Get the axis identifier of a coil.

Parameters:

in	trackId	Identification of track.
in	coilIndex	Coil index of the coil whose axis identifier to be retrieved.
out	pAxisId	Retrieved axis identifier of the coil.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
  - Application is not a CM client
- CM_ERR_SYSTEM_STATE_ERROR
  - LMS is not initialized
- CM_ERR_SHUTDOWN_IN_PROGRESS
  - Shutdown is in progress
- CM_ERR_INVALID_TRACK_ID
  - Invalid track identifier given
- CM_ERR_INVALID_COIL_INDEX
  - Invalid coil index given
- CM_ERR_INVALID_PARAMETER
  - pAxisId is NULL

State:

CM_SYSTEM_INITIALIZED

Remarks:

With the axis identifier the functions of the SAC and MAC interface can be called. Several of these functions influence the functionality of CM. The application must call the SAC and MAC functions with care.

1.7.10 CmGetCarrierId

```
NYCE_STATUS CmGetCarrierId(CM_UDINT carrierIndex,
                           CM_INT*  pCarrierId);
```

Convert a carrier index to a carrier identifier.

Parameters:

in	carrierIndex	Index of carrier
out	pCarrierId	Identification of carrier addressed by carrierIndex

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_ERROR
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_INDEX
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.7.11 CmGetCarrierIndex

```
NYCE_STATUS CmGetCarrierIndex(CM_INT    carrierId,
                              CM_UDINT* pCarrierIndex);
```

Convert a carrier identifier to a carrier index.

Parameters:

in	carrierId	Identification of carrier
out	pCarrierIndex	Index of carrier associated to carrierId

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_ERROR
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.7.12 CmGetSyncGroup

```
NYCE_STATUS CmGetSyncGroup(CM_INT    groupId,
                           CM_INT    carrierIds[carrierIdsLength],
                           CM_UDINT  carrierIdsLength,
                           CM_UDINT* pNrOfCarriers);
```

Get the carrier identifiers that belong to a synchronization group.

Parameters:

in	groupId	Identification of group.
out	carrierIds	Carrier identifiers belonging to the sync group.
in	carrierIdsLength	Allocated size of the array carrierIds in number of elements.
out	pNrOfCarriers	Number of carrier identifiers in carrierIds.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_SYNC_GROUP_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- NYCE_ERR_INVALID_OUTPUT_SIZE
  - (carrierIdsLength is zero) or (carrierIdsLength greater than CM_MAX_NR_OF_CARRIERS) or (number of carriers greater than carrierIdsLength)

1.7.13 CmJog

```
NYCE_STATUS CmJog(CM_INT   carrierId,
                  CM_LREAL velocity,
                  CM_LREAL acceleration);
```

Start the movement of a carrier with constant velocity.

When the command is successfully completed, the carrier has started to move, unless the movement is suspended, due to the fact that the feed override is zero.

Parameters:

in	carrierId	Identification of carrier.
in	velocity	Jogging velocity in pu/s.
in	acceleration	Acceleration in pu/s2.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR

State:

CM_TRACK_ACTIVE

CM_CARRIER_STOPPED or CM_CARRIER_SUSPENDED

Transition:

CM_CARRIER_STOPPED -> CM_CARRIER_SUSPENDED (if feed override equals 0)

CM_CARRIER_STOPPED -> CM_CARRIER_MOVING (when starting the acceleration movement)

CM_CARRIER_MOVING -> CM_CARRIER_JOGGING (when acceleration movement completed)

1.7.14 CmMoveAcrossTrackBounds

```
NYCE_STATUS CmMoveAcrossTrackBounds(CM_INT              trackId,
                                    CM_INT              carrierId,
                                    CM_LREAL            beginPosition,
                                    const CM_MOVE_PARS* pCmMovePars);
```

Start the movement of a carrier across bounds of a track.

When the command is successfully completed, and the carrier is located on the track, the carrier has started to move, unless the movement is suspended, due to the fact that the feed override is zero. When the command is successfully completed, and the carrier is NOT located on the track, the carrier will start to move, in accordance with the move parameters towards the (virtual) end position. The track starts generation of the setpoint from the time that matches with the position at the moment the carrier is being controlled.

Parameters:

in	trackId	Identification of track.
in	carrierId	Identification of carrier.
in	beginPosition	Begin position or virtual begin position.
in	pCmMovePars	Movement parameters, including end position or virtual end position.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID

- CM_ERR_NOT_SUPPORTED
  - invalid track type
- CM_ERR_INVALID_BEGIN_POSITION
- CM_ERR_INVALID_POSITIONS
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_TOO_MANY_CARRIERS_ON_TRACK
- CM_ERR_INITIAL_POSITION_MISMATCH
- CM_ERR_CARRIER_NOT_ON_TRACK

State:

CM_TRACK_ACTIVE

CM_CARRIER_STOPPED

Transition:

Only if the carrier is on the track CM_CARRIER_STOPPED -> CM_CARRIER_SUSPENDED (if feed override equals 0)

CM_CARRIER_STOPPED -> CM_CARRIER_MOVING

CM_CARRIER_MOVING -> CM_CARRIER_STOPPED (after completion of movement)

1.7.15 CmMoveAdjacent

```
NYCE_STATUS CmMoveAdjacent(CM_INT              carrierId,
                           const CM_MOVE_PARS* pCmMovePars,
                           CM_LREAL            mutualDistance);
```

Start a "move adjacent" of a carrier to a specified end position.

When the command is successfully completed, the carrier has started to bridge the gap with its predecessor, unless the movement is suspended, due to the fact that the feed override is set to zero after the start of the movement.

Parameters:

in	carrierId	Identification of carrier.
in	pCmMovePars	Movement parameters.
in	mutualDistance	Requested mutual distance.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
  - system error or feed override not constant
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_MUTUAL_DISTANCE
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR

Carrier Management

- CM_ERR_ORPHAN_CARRIER
  - no coil reacts on move
- CM_ERR_CARRIER_ID_NOT_FOUND

State:

CM_TRACK_ACTIVE

CM_CARRIER_STOPPED

predecessor state: CM_CARRIER_STOPPED or CM_CARRIER_JOGGING or CM_CARRIER_MOVING (at constant velocity)

feed override must be constant

Transition:

CM_CARRIER_STOPPED -> CM_CARRIER_MOVING

CM_CARRIER_MOVING -> CM_CARRIER_STOPPED (after completion of movement; predecessor state == CM_CARRIER_STOPPED)

CM_CARRIER_MOVING -> CM_CARRIER_JOGGING (after completion of movement; predecessor state == CM_CARRIER_JOGGING or CM_CARRIER_MOVING)

Remarks:

: CmMoveAdjacent only supports the 2nd order profile, so maxJerk must be NYCE_INFINITE.

1.7.16 CmMoveCarriersAgainstEndstop

```
NYCE_STATUS CmMoveCarriersAgainstEndstop(CM_INT   trackId,
                                         CM_INT   direction,
                                         CM_LREAL velocity);
```

Start moving all carriers on the track against the end stop.

The procedure moves all carriers against the end stop and determines the number of carriers against the end stop. The number of carriers can be retrieved by the function CmGetCarriersOnTrack.

Parameters:

in	trackId	Identification of track.
in	direction	Direction of movement (+1 = positive, -1 = negative).
in	velocity	Velocity of the carriers.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_HOME_TRACK_EXECUTING
- CM_ERR_MOVE_TO_ENDSTOP_EXECUTING

State:

CM_TRACK_IDLE or CM_TRACK_INACTIVE

Remarks:

The actual move to endstop action is performed in a separate thread because it can take quite some time. The following asynchronous errors can occur:

- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_TIMEOUT

- CM_ERR_NOT_END_OF_STROKE
- CM_ERR_ASYNC_ERROR_OCCURRED
- CM_ERR_MR_SENSOR_COMMUTATION_NOT_DETERMINED

1.7.17 CmMoveProfile

```
NYCE_STATUS CmMoveProfile(CM_INT                 carrierId,
                          const CM_PROFILE_PARS* pCmProfilePars);
```

Start the movement of a carrier along a profile.

The profile must start from standstill. The end velocity may be unequal to zero, which implies that the carriers keep moving at constant velocity, theoretically infinitely, practically until the end of the track. The profile must be monotonous in the position, which means that the velocity is always in one direction or zero. Note that the sign of the specified velocity value defines the direction of the movement. A negative velocity value defines a profile with a movement in the negative direction (from high to low position). On the basis of the profile points, the constant acceleration and time between two successive points is calculated. The acceleration is checked not to exceed the maximum track acceleration.

When the command is successfully completed, the carrier has started to move, unless the movement is suspended, due to the fact that the feed override is zero.

Parameters:

in	carrierId	Identification of carrier.
in	pCmProfilePars	Profile parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_POSITIONS
- CM_ERR_PRF_INVALID_NR_OF_PROFILE_POINTS
- CM_ERR_PRF_ALL_VELOCITIES_ZERO
- CM_ERR_PRF_INVALID_POSITION
- CM_ERR_PRF_VELOCITY_LIMIT_EXCEEDED
- CM_ERR_PRF_MOVEMENT_DIRECTION_NOT_CONSISTENT
- CM_ERR_PRF_ACCELERATION_LIMIT_EXCEEDED
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR

State:

CM_TRACK_ACTIVE

CM_CARRIER_STOPPED or CM_CARRIER_SUSPENDED

Transition:

CM_CARRIER_STOPPED -> CM_CARRIER_SUSPENDED (if feed override equals 0)

CM_CARRIER_STOPPED -> CM_CARRIER_MOVING (otherwise)

CM_CARRIER_MOVING -> CM_CARRIER_STOPPED (after completion of movement; end velocity == 0)

CM_CARRIER_MOVING -> CM_CARRIER_JOGGING (after completion of movement; end velocity != 0)

Carrier Management

1.7.18 CmMoveToPosition

```
NYCE_STATUS CmMoveToPosition(CM_INT              carrierId,
                             const CM_MOVE_PARS* pCmMovePars);
```

Start the movement of a carrier to a specified end position.

When the command is successfully completed, the carrier has started to move, unless the movement is suspended, due to the fact that the feed override is zero.

Parameters:

in	carrierId	Identification of carrier.
in	pCmMovePars	Movement parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR

State:

CM_TRACK_ACTIVE

any carrier state except CM_CARRIER_POWERUP

Transition:

CM_CARRIER_STOPPED -> CM_CARRIER_SUSPENDED (if feed override equals 0)

CM_CARRIER_STOPPED -> CM_CARRIER_MOVING

CM_CARRIER_JOGGING -> CM_CARRIER_MOVING

CM_CARRIER_MOVING -> CM_CARRIER_STOPPED (after completion of movement)

1.7.19 CmMoveToTrack

```
NYCE_STATUS CmMoveToTrack(CM_INT              carrierId,
                          const CM_MOVE_PARS* pCmMovePars,
                          CM_TRACK_POS        trackOriginPositionId,
                          CM_INT              trackDestinationId,
                          CM_TRACK_POS        trackDestinationPositionId);
```

Starts the movement of a carrier to an end position on another track.

When successfully completed, the carrier has started to move, unless the movement is suspended, due to the fact that the feed override is zero. During the movement the carrier leaves one track (origin) and enters another track (destination) in position mode.

Parameters:

in	carrierId	Identification of carrier.
in	pCmMovePars	Movement parameters. The end position is defined on the destination track.

in	trackOriginPositionId	Identification of track position. Side where the carrier leaves the origin track.
in	trackDestinationId	Identification of the destination track. This is the track, which the carrier enters.
in	trackDestinationPositionId	Identification of track position. Side where the carrier enters the destination track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_TRACK_POSITION_ID
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_MOVE_TO_TRACK_EXECUTING
- CM_ERR_NOT_SUPPORTED
- CM_ERR_TRACK_INVALID_FEED_OVERRIDE
- CM_ERR_ADD_MOVING_CARRIER_ALREADY_PENDING
- CM_ERR_CARRIER_POSITION_ERROR
- CM_ERR_OBSTRUCTED_BY_CARRIER

State:

for both tracks CM_TRACK_ACTIVE

any carrier state except CM_CARRIER_POWERUP

Transition:

CM_CARRIER_STOPPED -> CM_CARRIER_MOVING

CM_CARRIER_JOGGING -> CM_CARRIER_MOVING

CM_CARRIER_MOVING -> CM_CARRIER_STOPPED (after completion of movement)

1.7.20 CmReadCarrierData

```
NYCE_STATUS CmReadCarrierData(CM_CARRIER_DATA carrierData[carrierDataLength],
                              CM_UDINT        carrierDataLength,
                              CM_UDINT*       pNrOfCarriers);
```

Read the carrierId, position, velocity, acceleration, jerk, state and controller state of all carriers (CM_MAX_NR_OF_CARRIERS).

The data of each carrier is stored in the output array. Entries in the array of non-existent carrierIds are indicated with carrierId value CM_NO_ID. For these entries, zeros are written for carrier position, velocity, acceleration, jerk and state and CM_CARRIER_CONTROLLER_MOVING is written for carrier controller state.

Parameters:

out	carrierData	Carrier data
in	carrierDataLength	Allocated size of the array carrierData in number of elements.
out	pNrOfCarriers	Number of valid carriers in the array carrierData.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- NYCE_ERR_INVALID_OUTPUT_SIZE
  - (carrierIdsLength is zero) or (carrierIdsLength greater than CM_MAX_NR_OF_CARRIERS) or (number of carriers greater than carrierIdsLength)

State:

In any state. Although useful information for the carrier state and controller state can only be expected if the track state is CM_TRACK_ACTIVE. Information for the carrier position; velocity; acceleration and jerk is only reliable in track states CM_TRACK_INACTIVE; CM_TRACK_ACTIVE and CM_TRACK_ERROR.

1.7.21 CmReadCarrierPars

```
NYCE_STATUS CmReadCarrierPars(CM_INT          carrierId,
                              CM_CARRIER_PARS* pCarrierPars);
```

Read the carrier parameters of a specified carrier.

Parameters:

in	carrierId	Identification of carrier.
out	pCarrierPars	Carrier parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_CARRIER_ID_NOT_FOUND

State:

-

1.7.22 CmReadSingleCarrierData

```
NYCE_STATUS CmReadSingleCarrierData(CM_INT          carrierId,
                                    CM_CARRIER_DATA* pCarrierData);
```

Read the carrierId, position, velocity, acceleration, jerk, state and controller state of a specific carrier.

Parameters:

in	carrierId	Identification of carrier.
out	pCarrierData	Carrier data.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR

- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

State:

In any state. Although useful information for the carrier state and controller state can only be expected if the track state is CM_TRACK_ACTIVE. Information for the carrier position; velocity; acceleration and jerk is only reliable in track states CM_TRACK_INACTIVE; CM_TRACK_ACTIVE and CM_TRACK_ERROR.

1.7.23 CmRemoveCarrier

```
NYCE_STATUS CmRemoveCarrier(CM_INT carrierId);
```

Mark a carrier for removal.

The coil controlling the carrier is put in an open loop state and error detection is disabled.

Parameters:

in	carrierId	Identification of carrier.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_CARRIER_ID_NOT_FOUND

State:

CM_TRACK_INACTIVE or CM_TRACK_ACTIVE

if CM_TRACK_ACTIVE: CM_CARRIER_STOPPED or CM_CARRIER_SUSPENDED

Transition:

CM_CARRIER_STOPPED -> CM_CARRIER_POWERUP

CM_CARRIER_SUSPENDED -> CM_CARRIER_POWERUP

1.7.24 CmRemoveMovingCarrier

```
NYCE_STATUS CmRemoveMovingCarrier(CM_INT                  carrierId,
                                  CM_TRACK_POS            trackPositionId,
                                  const CM_VELOCITY_PARS* pCmVelocityPars);
```

Remove a carrier from the track while it is moving in position mode.

Call this function to specify a velocity with which a carrier should move beyond the end of the track on which it is currently present. It can be used to remove a carrier from the system while it remains moving in position mode. After the carrier is no longer detected by the outer sensor of the last coil of the track it will be removed from the carrier administration.

Parameters:

in	carrierId	Identification of carrier.
in	trackPositionId	Identification of track position. Side where the carrier leaves the track.
in	pCmVelocityPars	Velocity parameters.

Carrier Management

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_INVALID_TRACK_POSITION_ID
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_NOT_SUPPORTED
  - invalid track type

State:

CM_TRACK_ACTIVE

any carrier state except CM_CARRIER_POWERUP

Transition:

CM_CARRIER_STOPPED -> CM_CARRIER_SUSPENDED (if feed override equals 0)

CM_CARRIER_STOPPED -> CM_CARRIER_MOVING

CM_CARRIER_JOGGING -> CM_CARRIER_MOVING

CM_CARRIER_MOVING -> CM_CARRIER_POWERUP (if carrier is no longer detected by the outer sensor)

1.7.25 CmStartSyncGroup

```
NYCE_STATUS CmStartSyncGroup(CM_INT groupId);
```

Start the execution of the movement for the carriers in the synchronize group.

Parameters:

in	groupId	Identification of group.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_SYNC_GROUP_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR

State:

CM_TRACK_ACTIVE

Transition:

CM_CARRIER_STOPPED -> CM_CARRIER_MOVING

1.7.26 CmStopCarrier

```
NYCE_STATUS CmStopCarrier(CM_INT              carrierId,
                          const CM_STOP_PARS* pCmStopPars);
```

Initiate a stop profile for a carrier.

When the command is successfully completed, the carrier has initiated the stop, or is already at standstill.

Parameters:

in	carrierId	Identification of carrier.
in	pCmStopPars	Stop parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_PARAMETER_ERROR
- CM_ERR_INVALID_PARAMETER

State:

CM_TRACK_ACTIVE or CM_TRACK_ERROR or CM_TRACK_FATAL_ERROR

any carrier state except CM_CARRIER_POWERUP

Transition:

If CM_TRACK_ACTIVE: CM_CARRIER_JOGGING -> CM_CARRIER_MOVING

If CM_TRACK_ACTIVE: CM_CARRIER_SUSPENDED -> CM_CARRIER_STOPPED

If CM_TRACK_ACTIVE: CM_CARRIER_MOVING -> CM_CARRIER_STOPPED (after completion of stop)

1.7.27 CmStopCarriers

```
NYCE_STATUS CmStopCarriers(const CM_INT        carrierIds[nrOfCarriers],
                           CM_UDINT            nrOfCarriers,
                           const CM_STOP_PARS* pCmStopPars);
```

Start the execution of the stop command for the carriers at the same time.

Parameters:

in	carrierIds	Carrier identifiers.
in	nrOfCarriers	Number of carrier identifiers in carrierIds.
in	pCmStopPars	Stop Parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_DUPLICATE_CARRIER_ID

Carrier Management

- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_PARAMETER
- CM_ERR_PARAMETER_OUT_OF_RANGE

State:

CM_TRACK_ACTIVE

any carrier state except CM_CARRIER_POWERUP

Transition:

CM_CARRIER_JOGGING -> CM_CARRIER_MOVING

CM_CARRIER_SUSPENDED -> CM_CARRIER_STOPPED

CM_CARRIER_MOVING -> CM_CARRIER_STOPPED (after completion of stop)

1.7.28 CmWriteCarrierPars

```
NYCE_STATUS CmWriteCarrierPars(CM_INT                 carrierId,
                               const CM_CARRIER_PARS* pCarrierPars);
```

Write the carrier parameters for a specified carrier.

Parameters:

in	carrierId	Identification of carrier.
in	pCarrierPars	Carrier parameters.

Postconditions:

If the carrierId is valid in the system, the carrierId is automatically linked to the carrier parameters after the carrier parameters are written.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_MEASUREMENT_SCALE_OFFSET
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TOO_MANY_CARRIER_PARAMETERS

State:

CM_CARRIER_POWERUP or CM_CARRIER_STOPPED

Remarks:

- If the carrierId is available in the system, then the carrier must be in POWERUP or STOPPED state.
- If the product/measurement scale offset in pCarrierPars equals to 0, and the carrierId is available in the system, it means that the specified carrier does not need to adjust product/measurement scale offset any more.

1.8 Error and Warning

The functions within this module are related to errors and warnings.

1.8.1 CmClearWarning

```
NYCE_STATUS CmClearWarning(CM_INT trackId);
```

Clear warnings of the coils on the specified track.

The track state is not changed.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID

1.8.2 CmResetError

```
NYCE_STATUS CmResetError(CM_INT trackId);
```

Reset all errors and warnings of the coils on the specified track.

Parameters:

in	trackId	Identification of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_MOVE_TO_TRACK_EXECUTING

Transition:

CM_TRACK_FATAL_ERROR -> CM_TRACK_IDLE

CM_TRACK_ERROR or CM_TRACK_ACTIVE -> CM_TRACK_INACTIVE

All carriers: current state -> CM_CARRIER_POWERUP

1.9 Miscellaneous

1.9.1 CmConnect

```
NYCE_STATUS CmConnect();
```

Connect a user application to the CM layer.

The user application becomes a CM client.

Retvals:

- NYCE_OK
  - successful
- NYCE_ERR_WRONG_DLL_VERSION

Carrier Management

- CM_ERR_CM_SERVICE_NOT_AVAILABLE
- CM_ERR_SERVICE_COMMUNICATION_ERROR
- CM_WRN_ALREADY_CONNECTED
- CM_ERR_TOO_MANY_CLIENTS

1.9.2 CmDisconnect

```
NYCE_STATUS CmDisconnect();
```

Disconnect a user application from the CM layer.

The user application is no longer a CM client.

Retvals:

- NYCE_OK
  - successful
- CM_WRN_ALREADY_DISCONNECTED
- CM_ERR_HOME_TRACK_EXECUTING

1.9.3 CmEnableSensorDetection

```
NYCE_STATUS CmEnableSensorDetection(CM_INT          trackId,
                                    const CM_AREA* pCmArea,
                                    CM_BOOL         onOff);
```

Enable or disable the sensor detection of sensors is a specific area.

The sensors have a sensor range overlapping the specified area.

Parameters:

in	trackId	Identification of track.
in	pCmArea	Area description.
in	onOff	Indicating if sensor detection must be enabled (TRUE) or disabled (FALSE).

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_POSITIONS
- CM_ERR_INVALID_MIN_POSITION
- CM_ERR_INVALID_MAX_POSITION
- CM_ERR_INVALID_PARAMETER

1.9.4 CmReadSpecialControllerPars

```
NYCE_STATUS CmReadSpecialControllerPars(CM_INT               trackId,
                                        const CM_AREA*       pCmArea,
                                        CM_CONTROLLER_PARS* pControllerPars,
                                        CM_BOOL*             pUsed);
```

Read the set of special controller parameters of an area of a track and if they are used.

Parameters:

in	trackId	Identification of track.
in	pCmArea	Area description.
out	pControllerPars	Controller parameters.

out	pUsed	Indicating if the special controller parameters are used.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_POSITIONS
- CM_ERR_INVALID_MIN_POSITION
- CM_ERR_INVALID_MAX_POSITION
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.9.5 CmResetLogFunction

```
NYCE_STATUS CmResetLogFunction();
```

Reset the function pointer in DEH to point to the default log function DehDefaultLogFunction.

This function also deletes the log socket between the CM application and CM service.

Retvals:

- NYCE_OK
  if the reset and the deletion of log socket is successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_WRN_LOG_SOCKET_CLOSE_FAILED
  if the deletion of log socket failed

1.9.6 CmSelectSpecialControllerPars

```
NYCE_STATUS CmSelectSpecialControllerPars(CM_INT          trackId,
                                          const CM_AREA* pCmArea,
                                          CM_BOOL         use);
```

Select if the set of special controller parameters for an area of a track must be used.

Parameters:

in	trackId	Identification of track.
in	pCmArea	Area description.
in	use	Indicating if the special controller parameters must be used.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_POSITIONS
- CM_ERR_INVALID_MIN_POSITION
- CM_ERR_INVALID_MAX_POSITION

1.9.7 CmSetLogFunction

```
NYCE_STATUS CmSetLogFunction(DEH_LOG_FUNC pfnDehLogFunction);
```

Carrier Management

Set the function pointer in DEH to point to the user-defined log function.

This function also creates the log socket between the CM application and CM service.

Parameters:

in pfnDehLogFunction Implementation of the log function.

Retvals:

- NYCE_OK
  if the set is successful and the log socket is created successfully
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_INVALID_PARAMETER
  if pfnDehLogFunction is NULL
- CM_ERR_LOG_SOCKET_SETUP_ERROR
  if the creation of log socket failed

Remarks:

Registered log function will influence the product performance.

1.9.8 CmSynchronize

```
NYCE_STATUS CmSynchronize(CM_ID           cmId,
                          CM_SYNC_REQUEST syncRequest,
                          CM_LREAL        timeout);
```

Wait until a synchronize request has been completed.

With timeout = 0, CmSynchronize checks on completion of the request. With timeout = NYCE_INFINITE, CmSynchronize waits indefinitely on completion of the request.

Parameters:

in	cmId	Identification of CM object. The identification refers to a track or carrier.
in	syncRequest	Synchronize request.
in	timeout	Time-out [s].

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_REQUEST_TIMEOUT
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_CARRIER_NOT_MARKED_FOR_ADDITION
- CM_ERR_CARRIER_NOT_MARKED_FOR_REMOVAL
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_REQUEST_ABORTED
- CM_ERR_ASYNC_ERROR_OCCURRED

1.9.9 CmWriteSpecialControllerPars

```
NYCE_STATUS CmWriteSpecialControllerPars(CM_INT                 trackId,
                                         const CM_AREA*         pCmArea,
                                         const CM_CONTROLLER_PARS*
                                                                pControllerPars);
```

Write a set of special controller parameters for an area of a track.

Parameters:

in	trackId	Identification of track.
in	pCmArea	Area description.
in	pControllerPars	Controller parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_POSITIONS
- CM_ERR_INVALID_MIN_POSITION
- CM_ERR_INVALID_MAX_POSITION
- CM_ERR_INVALID_PARAMETER

1.10 Conversion

The functions within this module convert strings to enumeration values and vice versa.

1.10.1 CmCarrierControllerStateFromString

```
NYCE_STATUS CmCarrierControllerStateFromString(const char*     valueStr,
                                               CM_CARRIER_CONTROLLER_STATE*  pValue);
```

Convert the given string to an CM_CARRIER_CONTROLLER_STATE.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_CARRIER_CONTROLLER_STATE.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_CARRIER_CONTROLLER_STATE.

1.10.2 CmCarrierControllerStateIsValid

```
BOOL CmCarrierControllerStateIsValid(CM_CARRIER_CONTROLLER_STATE value);
```

Check whether value is a valid CM_CARRIER_CONTROLLER_STATE.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_CARRIER_CONTROLLER_STATE.
- FALSE
  value is not a valid CM_CARRIER_CONTROLLER_STATE.

1.10.3 CmCarrierControllerStateToDescription

```
const char* CmCarrierControllerStateToDescription(CM_CARRIER_CONTROLLER_STATE
                                                   value);
```

Convert type CM_CARRIER_CONTROLLER_STATE to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.4 CmCarrierControllerStateToString

```
const char* CmCarrierControllerStateToString(CM_CARRIER_CONTROLLER_STATE    value);
```

Convert type CM_CARRIER_CONTROLLER_STATE to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.5 CmCarrierControllerStateToUserString

```
const char* CmCarrierControllerStateToUserString(CM_CARRIER_CONTROLLER_STATE value);
```

Convert type CM_CARRIER_CONTROLLER_STATE to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.6 CmCarrierStateFromString

```
NYCE_STATUS CmCarrierStateFromString(const char*       valueStr,
                                     CM_CARRIER_STATE* pValue);
```

Convert the given string to an CM_CARRIER_STATE.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_CARRIER_STATE.

- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_CARRIER_STATE.

1.10.7 CmCarrierStateIsValid

```
BOOL CmCarrierStateIsValid(CM_CARRIER_STATE value);
```
Check whether value is a valid CM_CARRIER_STATE.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_CARRIER_STATE.
- FALSE
  value is not a valid CM_CARRIER_STATE.

1.10.8 CmCarrierStateToDescription

```
const char* CmCarrierStateToDescription(CM_CARRIER_STATE value);
```
Convert type CM_CARRIER_STATE to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.9 CmCarrierStateToString

```
const char* CmCarrierStateToString(CM_CARRIER_STATE value);
```
Convert type CM_CARRIER_STATE to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.10 CmCarrierStateToUserString

```
const char* CmCarrierStateToUserString(CM_CARRIER_STATE value);
```
Convert type CM_CARRIER_STATE to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

Carrier Management

1.10.11 CmCollAvoidCarrierStateFromString

```
NYCE_STATUS CmCollAvoidCarrierStateFromString(const char*      valueStr,
                                              CM_COLL_AVOID_CARRIER_STATE*   pValue);
```

Convert the given string to an CM_COLL_AVOID_CARRIER_STATE.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_COLL_AVOID_CARRIER_STATE.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_COLL_AVOID_CARRIER_STATE.

1.10.12 CmCollAvoidCarrierStateIsValid

```
BOOL CmCollAvoidCarrierStateIsValid(CM_COLL_AVOID_CARRIER_STATE  value);
```

Check whether value is a valid CM_COLL_AVOID_CARRIER_STATE.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_COLL_AVOID_CARRIER_STATE.
- FALSE
  value is not a valid CM_COLL_AVOID_CARRIER_STATE.

1.10.13 CmCollAvoidCarrierStateToDescription

```
const char* CmCollAvoidCarrierStateToDescription(CM_COLL_AVOID_CARRIER_STATE value);
```

Convert type CM_COLL_AVOID_CARRIER_STATE to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.14 CmCollAvoidCarrierStateToString

```
const char* CmCollAvoidCarrierStateToString(CM_COLL_AVOID_CARRIER_STATE      value);
```

Convert type CM_COLL_AVOID_CARRIER_STATE to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.15 CmCollAvoidCarrierStateToUserString

```
const char* CmCollAvoidCarrierStateToUserString(CM_COLL_AVOID_CARRIER_STATE  value);
```

Convert type CM_COLL_AVOID_CARRIER_STATE to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.16 CmCollAvoidProcessAreaStateFromString

```
NYCE_STATUS CmCollAvoidProcessAreaStateFromString(const char*  valueStr,
                                                  CM_COLL_AVOID_PROCESS_AREA_STATE*
                                                  pValue);
```

Convert the given string to an CM_COLL_AVOID_PROCESS_AREA_STATE.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_COLL_AVOID_PROCESS_AREA_STATE.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_COLL_AVOID_PROCESS_AREA_STATE.

1.10.17 CmCollAvoidProcessAreaStateIsValid

```
BOOL CmCollAvoidProcessAreaStateIsValid(CM_COLL_AVOID_PROCESS_AREA_STATE     value);
```

Check whether value is a valid CM_COLL_AVOID_PROCESS_AREA_STATE.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_COLL_AVOID_PROCESS_AREA_STATE.
- FALSE
  value is not a valid CM_COLL_AVOID_PROCESS_AREA_STATE.

1.10.18 CmCollAvoidProcessAreaStateToDescription

```
const char* CmCollAvoidProcessAreaStateToDescription(CM_COLL_AVOID_PROCESS_AREA_STATE
                                                     value);
```

Convert type CM_COLL_AVOID_PROCESS_AREA_STATE to a descriptive string.

Parameters:

in	value	The value to convert.

Carrier Management

Returns:
The description of value.

1.10.19 CmCollAvoidProcessAreaStateToString

```
const char* CmCollAvoidProcessAreaStateToString(CM_COLL_AVOID_PROCESS_AREA_STATE
                                                value);
```

Convert type CM_COLL_AVOID_PROCESS_AREA_STATE to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:
The string representation of value.

1.10.20 CmCollAvoidProcessAreaStateToUserString

```
const char* CmCollAvoidProcessAreaStateToUserString(CM_COLL_AVOID_PROCESS_AREA_STATE
                                                    value);
```

Convert type CM_COLL_AVOID_PROCESS_AREA_STATE to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:
The human-readable string representation of value.

1.10.21 CmEventByIndex

```
NYCE_STATUS CmEventByIndex(uint32_t  index,
                           CM_EVENT* pValue);
```

Retrieve the n-th CM event.

Parameters:

in	index	The index within CM_NR_OF_EVENTS to retrieve.
out	pValue	The value of the n-th CM event.

Retvals:

- NYCE_OK
  event successfully retrieved.
- CM_ERR_INVALID_PARAMETER
  pValue is NULL.
- CM_ERR_PARAMETER_OUT_OF_RANGE
  index is out of range.

1.10.22 CmEventFromString

```
NYCE_STATUS CmEventFromString(const char* valueStr,
                              CM_EVENT*   pValue);
```

Convert the given string to an CM_EVENT.

Parameters:

in	valueStr	The string to convert.

out	pValue	The event id of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_EVENT.
- CM_ERR_INVALID_PARAMETER
  valueStr or pValue is NULL.
- CM_ERR_INVALID_STRING
  valueStr could not be converted to CM_EVENT.

1.10.23 CmEventGetCategory

```
CM_EVENT_CATEGORY CmEventGetCategory(CM_EVENT value);
```

Retrieve the category of CM_EVENT.

Parameters:

in	value	The event.

Returns:

The category of the event.

1.10.24 CmEventIsValid

```
BOOL CmEventIsValid(CM_EVENT value);
```

Check whether value is a valid CM_EVENT.

Parameters:

in	value	The event to check.

Retvals:

- TRUE
  value is a valid cm event.
- FALSE
  value is an invalid cm event.

1.10.25 CmEventNrOfEvents

```
uint32_t CmEventNrOfEvents();
```

Retrieve the number of CM_EVENTs.

Returns:

Number of CM_EVENTs.

1.10.26 CmEventToDescription

```
const char* CmEventToDescription(CM_EVENT value);
```

Retrieve the description of CM_EVENT.

Parameters:

in	value	The event id.

Returns:

The description of the event.

Carrier Management

1.10.27 CmEventToString

```
const char* CmEventToString(CM_EVENT value);
```
Convert type CM_EVENT to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.28 CmObjectTypeFromString

```
NYCE_STATUS CmObjectTypeFromString(const char*      valueStr,
                                   CM_OBJECT_TYPE* pValue);
```
Convert the given string to an CM_OBJECT_TYPE.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_OBJECT_TYPE.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_OBJECT_TYPE.

1.10.29 CmObjectTypeIsValid

```
BOOL CmObjectTypeIsValid(CM_OBJECT_TYPE value);
```
Check whether value is a valid CM_OBJECT_TYPE.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_OBJECT_TYPE.
- FALSE
  value is not a valid CM_OBJECT_TYPE.

1.10.30 CmObjectTypeToDescription

```
const char* CmObjectTypeToDescription(CM_OBJECT_TYPE value);
```
Convert type CM_OBJECT_TYPE to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.31 CmObjectTypeToString

```
const char* CmObjectTypeToString(CM_OBJECT_TYPE value);
```
Convert type CM_OBJECT_TYPE to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.32 CmObjectTypeToUserString

```
const char* CmObjectTypeToUserString(CM_OBJECT_TYPE value);
```
Convert type CM_OBJECT_TYPE to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.33 CmParIdByIndex

```
NYCE_STATUS CmParIdByIndex(uint32_t   varIndex,
                           CM_PAR_ID* pValue);
```
Retrieve the n-th CM_PAR_ID.

Parameters:

in	varIndex	The index within CM_PAR_NR_OF_IDS to retrieve.
out	pValue	The value of the n-th cm parameter.

Retvals:

- NYCE_OK
  parameter successfully retrieved.
- CM_ERR_INVALID_PARAMETER
  pValue is NULL.
- CM_ERR_PARAMETER_OUT_OF_RANGE
  index is out of range.

1.10.34 CmParIdFromString

```
NYCE_STATUS CmParIdFromString(const char* valueStr,
                              CM_PAR_ID*  pValue);
```
Convert the given string to an CM_PAR_ID.

Parameters:

in	valueStr	The string to convert.
out	pValue	The parameter id of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_PAR_ID.

- CM_ERR_INVALID_PARAMETER
  valueStr or pValue is NULL.
- CM_ERR_INVALID_STRING
  valueStr could not be converted to CM_PAR_ID.

1.10.35 CmParIdGetCategory

```
CM_PAR_CATEGORY CmParIdGetCategory(CM_PAR_ID value);
```
Retrieve the category of CM_PAR_ID.

Parameters:

in	value	The parameter id.

Returns:

The category of the parameter.

1.10.36 CmParIdGetDataType

```
CM_TYPE_ID CmParIdGetDataType(CM_PAR_ID value);
```
Retrieve the data type of CM_PAR_ID.

Parameters:

in	value	The parameter id.

Returns:

The data type of the parameter.

1.10.37 CmParIdGetUnit

```
const char* CmParIdGetUnit(CM_PAR_ID value);
```
Retrieve the unit type of a CM_PAR_ID.

Parameters:

in	value	The parameter id.

Returns:

The unit of the parameter.

1.10.38 CmParIdIsValid

```
BOOL CmParIdIsValid(CM_PAR_ID value);
```
Check whether value is a valid CM_PAR_ID.

Parameters:

in	value	The parameter to check.

Retvals:

- TRUE
  value is a valid cm parameter.
- FALSE
  value is an invalid cm parameter.

1.10.39 CmParIdNrOfIds

```
uint32_t CmParIdNrOfIds();
```
Retrieve the number of CM_PAR_IDs.

Returns:

Number of CM_PAR_IDs.

1.10.40 CmParIdToDescription

```
const char* CmParIdToDescription(CM_PAR_ID value);
```
Retrieve the description of CM_PAR_ID.

Parameters:

in	value	The parameter id.

Returns:

The description of the parameter.

1.10.41 CmParIdToString

```
const char* CmParIdToString(CM_PAR_ID value);
```
Convert type CM_PAR_ID to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.42 CmParIdToUserString

```
const char* CmParIdToUserString(CM_PAR_ID value);
```
Convert type CM_PAR_ID to a human-readable string.

Parameters:

in	value	The parameter to convert.

Returns:

The human-readable string representation of value.

1.10.43 CmParSectionTypeFromString

```
NYCE_STATUS CmParSectionTypeFromString(const char*           valueStr,
                                       CM_PAR_SECTION_TYPE* pValue);
```
Convert the given string to an CM_PAR_SECTION_TYPE.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

Carrier Management

- NYCE_OK
  valueStr is successfully converted to a valid CM_PAR_SECTION_TYPE.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_PAR_SECTION_TYPE.

1.10.44 CmParSectionTypeIsValid

```
BOOL CmParSectionTypeIsValid(CM_PAR_SECTION_TYPE value);
```
Check whether value is a valid CM_PAR_SECTION_TYPE.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_PAR_SECTION_TYPE.
- FALSE
  value is not a valid CM_PAR_SECTION_TYPE.

1.10.45 CmParSectionTypeToDescription

```
const char* CmParSectionTypeToDescription(CM_PAR_SECTION_TYPE value);
```
Convert type CM_PAR_SECTION_TYPE to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.46 CmParSectionTypeToString

```
const char* CmParSectionTypeToString(CM_PAR_SECTION_TYPE value);
```
Convert type CM_PAR_SECTION_TYPE to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.47 CmParSectionTypeToUserString

```
const char* CmParSectionTypeToUserString(CM_PAR_SECTION_TYPE value);
```
Convert type CM_PAR_SECTION_TYPE to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.48 CmSensorIdFromString

```
NYCE_STATUS CmSensorIdFromString(const char*   valueStr,
                                 CM_SENSOR_ID* pValue);
```

Convert the given string to an CM_SENSOR_ID.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_SENSOR_ID.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_SENSOR_ID.

1.10.49 CmSensorIdIsValid

```
BOOL CmSensorIdIsValid(CM_SENSOR_ID value);
```

Check whether value is a valid CM_SENSOR_ID.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_SENSOR_ID.
- FALSE
  value is not a valid CM_SENSOR_ID.

1.10.50 CmSensorIdToDescription

```
const char* CmSensorIdToDescription(CM_SENSOR_ID value);
```

Convert type CM_SENSOR_ID to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.51 CmSensorIdToString

```
const char* CmSensorIdToString(CM_SENSOR_ID value);
```

Convert type CM_SENSOR_ID to the exact string representation.

Parameters:

in	value	The value to convert.

Carrier Management

Returns:

The string representation of value.

1.10.52 CmSensorIdToUserString

```
const char* CmSensorIdToUserString(CM_SENSOR_ID value);
```

Convert type CM_SENSOR_ID to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.53 CmSyncRequestFromString

```
NYCE_STATUS CmSyncRequestFromString(const char*      valueStr,
                                    CM_SYNC_REQUEST* pValue);
```

Convert the given string to an CM_SYNC_REQUEST.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_SYNC_REQUEST.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_SYNC_REQUEST.

1.10.54 CmSyncRequestIsValid

```
BOOL CmSyncRequestIsValid(CM_SYNC_REQUEST value);
```

Check whether value is a valid CM_SYNC_REQUEST.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_SYNC_REQUEST.
- FALSE
  value is not a valid CM_SYNC_REQUEST.

1.10.55 CmSyncRequestToDescription

```
const char* CmSyncRequestToDescription(CM_SYNC_REQUEST value);
```

Convert type CM_SYNC_REQUEST to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.56 CmSyncRequestToString

```
const char* CmSyncRequestToString(CM_SYNC_REQUEST value);
```

Convert type CM_SYNC_REQUEST to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.57 CmSyncRequestToUserString

```
const char* CmSyncRequestToUserString(CM_SYNC_REQUEST value);
```

Convert type CM_SYNC_REQUEST to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.58 CmSystemStateFromString

```
NYCE_STATUS CmSystemStateFromString(const char*      valueStr,
                                    CM_SYSTEM_STATE* pValue);
```

Convert the given string to an CM_SYSTEM_STATE.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_SYSTEM_STATE.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_SYSTEM_STATE.

1.10.59 CmSystemStateIsValid

```
BOOL CmSystemStateIsValid(CM_SYSTEM_STATE value);
```

Check whether value is a valid CM_SYSTEM_STATE.

Parameters:

Carrier Management

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_SYSTEM_STATE.
- FALSE
  value is not a valid CM_SYSTEM_STATE.

1.10.60 CmSystemStateToDescription

```
const char* CmSystemStateToDescription(CM_SYSTEM_STATE value);
```

Convert type CM_SYSTEM_STATE to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.61 CmSystemStateToString

```
const char* CmSystemStateToString(CM_SYSTEM_STATE value);
```

Convert type CM_SYSTEM_STATE to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.62 CmSystemStateToUserString

```
const char* CmSystemStateToUserString(CM_SYSTEM_STATE value);
```

Convert type CM_SYSTEM_STATE to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.63 CmTrackPosFromString

```
NYCE_STATUS CmTrackPosFromString(const char*   valueStr,
                                 CM_TRACK_POS* pValue);
```

Convert the given string to an CM_TRACK_POS.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_TRACK_POS.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_TRACK_POS.

1.10.64 CmTrackPosIsValid

```
BOOL CmTrackPosIsValid(CM_TRACK_POS value);
```
Check whether value is a valid CM_TRACK_POS.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_TRACK_POS.
- FALSE
  value is not a valid CM_TRACK_POS.

1.10.65 CmTrackPosToDescription

```
const char* CmTrackPosToDescription(CM_TRACK_POS value);
```
Convert type CM_TRACK_POS to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.66 CmTrackPosToString

```
const char* CmTrackPosToString(CM_TRACK_POS value);
```
Convert type CM_TRACK_POS to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.67 CmTrackPosToUserString

```
const char* CmTrackPosToUserString(CM_TRACK_POS value);
```
Convert type CM_TRACK_POS to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

Carrier Management

The human-readable string representation of value.

1.10.68 CmTrackStateFromString

```
NYCE_STATUS CmTrackStateFromString(const char*      valueStr,
                                   CM_TRACK_STATE* pValue);
```

Convert the given string to an CM_TRACK_STATE.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_TRACK_STATE.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_TRACK_STATE.

1.10.69 CmTrackStateIsValid

```
BOOL CmTrackStateIsValid(CM_TRACK_STATE value);
```

Check whether value is a valid CM_TRACK_STATE.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_TRACK_STATE.
- FALSE
  value is not a valid CM_TRACK_STATE.

1.10.70 CmTrackStateToDescription

```
const char* CmTrackStateToDescription(CM_TRACK_STATE value);
```

Convert type CM_TRACK_STATE to a descriptive string.

Parameters:

in	value	The value to convert.

Returns:

The description of value.

1.10.71 CmTrackStateToString

```
const char* CmTrackStateToString(CM_TRACK_STATE value);
```

Convert type CM_TRACK_STATE to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.72 CmTrackStateToUserString

```
const char* CmTrackStateToUserString(CM_TRACK_STATE value);
```

Convert type CM_TRACK_STATE to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.73 CmTypeIdFromString

```
NYCE_STATUS CmTypeIdFromString(const char* valueStr,
                               CM_TYPE_ID* pValue);
```

Convert the given string to an CM_TYPE_ID.

Parameters:

in	valueStr	The string to convert.
out	pValue	The enum value of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_TYPE_ID.
- CM_ERR_INVALID_PARAMETER
  valueStr is NULL.
- CM_ERR_INVALID_OUTPUT_ARGUMENT
  pValue is NULL.
- CM_ERR_UNKNOWN_ENUM_STRING
  valueStr could not be converted to CM_TYPE_ID.

1.10.74 CmTypeIdIsValid

```
BOOL CmTypeIdIsValid(CM_TYPE_ID value);
```

Check whether value is a valid CM_TYPE_ID.

Parameters:

in	value	The value to check.

Retvals:

- TRUE
  value is a valid CM_TYPE_ID.
- FALSE
  value is not a valid CM_TYPE_ID.

1.10.75 CmTypeIdToDescription

```
const char* CmTypeIdToDescription(CM_TYPE_ID value);
```

Convert type CM_TYPE_ID to a descriptive string.

Parameters:

Carrier Management

in	value	The value to convert.

Returns:

The description of value.

1.10.76 CmTypeIdToString

```
const char* CmTypeIdToString(CM_TYPE_ID value);
```

Convert type CM_TYPE_ID to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.77 CmTypeIdToUserString

```
const char* CmTypeIdToUserString(CM_TYPE_ID value);
```

Convert type CM_TYPE_ID to a human-readable string.

Parameters:

in	value	The value to convert.

Returns:

The human-readable string representation of value.

1.10.78 CmVarIdByIndex

```
NYCE_STATUS CmVarIdByIndex(uint32_t   varIndex,
                           CM_VAR_ID* pValue);
```

Retrieve the n-th CM_VAR_ID.

Parameters:

in	varIndex	The index within CM_VAR_NR_OF_IDS to retrieve.
out	pValue	The value of the n-th cm variable.

Retvals:

- NYCE_OK
  variable successfully retrieved.
- CM_ERR_INVALID_PARAMETER
  pValue is NULL.
- CM_ERR_PARAMETER_OUT_OF_RANGE
  index is out of range.

1.10.79 CmVarIdFromString

```
NYCE_STATUS CmVarIdFromString(const char* valueStr,
                              CM_VAR_ID*  pValue);
```

Convert the given string to an CM_VAR_ID.

Parameters:

in	valueStr	The string to convert.

out	pValue	The variable id of string.

Retvals:

- NYCE_OK
  valueStr is successfully converted to a valid CM_VAR_ID.
- CM_ERR_INVALID_PARAMETER
  valueStr or pValue is NULL.
- CM_ERR_INVALID_STRING
  valueStr could not be converted to CM_VAR_ID.

1.10.80 CmVarIdGetCategory

```
CM_VAR_CATEGORY CmVarIdGetCategory(CM_VAR_ID value);
```
Retrieve the category of CM_VAR_ID.

Parameters:

in	value	The variable id.

Returns:

The category of the variable.

1.10.81 CmVarIdGetDataType

```
NYCE_DATA_TYPE CmVarIdGetDataType(CM_VAR_ID value);
```
Retrieve the data type of CM_VAR_ID.

Parameters:

in	value	The variable id.

Returns:

The data type of the variable.

1.10.82 CmVarIdGetUnit

```
const char* CmVarIdGetUnit(CM_VAR_ID value);
```
Retrieve the unit type of a CM_VAR_ID.

Parameters:

in	value	The variable id.

Returns:

The unit of the variable.

1.10.83 CmVarIdIsValid

```
BOOL CmVarIdIsValid(CM_VAR_ID value);
```
Check whether value is a valid CM_VAR_ID.

Parameters:

in	value	The variable to check.

Retvals:

- TRUE
  value is a valid cm variable.
- FALSE
  value is an invalid cm variable.

1.10.84 CmVarIdNrOfIds

```
uint32_t CmVarIdNrOfIds();
```
Retrieve the number of CM_VAR_IDs.

Returns:

Number of CM_VAR_IDs.

1.10.85 CmVarIdToDescription

```
const char* CmVarIdToDescription(CM_VAR_ID value);
```
Retrieve the description of CM_VAR_ID.

Parameters:

in	value	The variable id.

Returns:

The description of the variable.

1.10.86 CmVarIdToString

```
const char* CmVarIdToString(CM_VAR_ID value);
```
Convert type CM_VAR_ID to the exact string representation.

Parameters:

in	value	The value to convert.

Returns:

The string representation of value.

1.10.87 CmVarIdToUserString

```
const char* CmVarIdToUserString(CM_VAR_ID value);
```
Convert type CM_VAR_ID to a human-readable string.

Parameters:

in	value	The variable to convert.

Returns:

The human-readable string representation of value.

1.11 Variables

The functions within this module are used to access CM variables.

1.11.1 CmReadCarrierVariable

```
NYCE_STATUS CmReadCarrierVariable(CM_INT    carrierId,
                                  CM_VAR_ID varId,
                                  CM_LREAL* pValue);
```

Read the value of the specified carrier variable.

Parameters:

in	carrierId	Identification of carrier.
in	varId	Identification of variable.
out	pValue	Value of the variable.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_VAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- CM_ERR_CARRIER_STATE_ERROR
  - if the variable is CM_VAR_CARRIER_SETPOINT_STOPPED_POSITION and the carrier state is not in CM_CARRIER_STOPPED state.

1.11.2 CmReadCoilVariable

```
NYCE_STATUS CmReadCoilVariable(CM_INT    trackId,
                               CM_DINT   coilIndex,
                               CM_VAR_ID varId,
                               CM_LREAL* pValue);
```

Read the value of the specified coil variable.

Parameters:

in	trackId	Identification of track.
in	coilIndex	Index of coil.
in	varId	Identification of variable.
out	pValue	Value of the variable.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_COIL_INDEX
- CM_ERR_INVALID_VAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.11.3 CmReadSystemVariable

```
NYCE_STATUS CmReadSystemVariable(CM_VAR_ID varId,
                                 CM_LREAL* pValue);
```

Read the value of the specified system variable.

Carrier Management

Parameters:

in	varId	Identification of variable.
out	pValue	Value of the variable.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_VAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.11.4 CmReadTrackVariable

```
NYCE_STATUS CmReadTrackVariable(CM_INT    trackId,
                                CM_VAR_ID varId,
                                CM_LREAL* pValue);
```

Read the value of the specified track variable.

Parameters:

in	trackId	Identification of track.
in	varId	Identification of variable.
out	pValue	Value of the variable.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_VAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

1.12 Event functionality

The module contains the events and functions of the CM event functionality.

1.12.1 CmDefineEventEnrolment

```
NYCE_STATUS CmDefineEventEnrolment(CM_ID            cmId,
                                   CM_EVENT         eventId,
                                   CM_EVENT_HANDLER handler,
                                   void*            pUserData);
```

Define a coupling between an enrolment function and an event.

When the event occurs the enrolment function with appropriate parameters is called. For every event multiple enrolment functions can be defined. The order of execution is the same as the order in which they are defined.

Parameters:

in	cmId	Identification of CM object. The identification refers to a system, track or carrier.
in	eventId	Identification of CM event.
in	handler	Event enrolment function.

in pUserData User specific data.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_EVENT_ID
- CM_ERR_INVALID_PARAMETER
- CM_ERR_ENROLMENT_DEFINED_FOR_TOO_MANY_CARRIERS
- CM_WRN_ENROLMENT_ALREADY_DEFINED

1.12.2 CmDeleteEventEnrolment

```
NYCE_STATUS CmDeleteEventEnrolment(CM_ID            cmId,
                                   CM_EVENT         eventId,
                                   CM_EVENT_HANDLER handler,
                                   void*            pUserData);
```

Delete a coupling between an enrolment function and an event.

Parameters:

in cmId Identification of CM object. The identification refers to a system, track or carrier.
in eventId Identification of CM event.
in handler Event enrolment function.
in pUserData User specific data.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_EVENT_ID
- CM_ERR_INVALID_PARAMETER
- CM_ERR_ENROLMENT_NOT_DEFINED

1.13 Collision avoidance functionality

The functions within this module are used to move a carrier with collision avoidance.

1.13.1 CmCollAvoidAddCarrierWithProfile

```
NYCE_STATUS CmCollAvoidAddCarrierWithProfile(CM_INT            trackId,
                                             CM_INT            carrierId,
                                             CM_LREAL          beginPosition,
                                             CM_LREAL          handoverPosition,
                                             CM_LREAL          firstSafePosition,
                                             const CM_MOVE_PARS* pCmMovePars);
```

Add a moving carrier to a track with collision avoidance.

Carrier Management

The carrier moves from a virtual begin position to an end position on the receiving track. Depending of the profile the carrier enters the receiving tracks while it is accelerating or decelerating.

The function provides collision avoidance during the path from the first safe position to the end position. The function does not provide collision avoidance during the path from the entering side of the track to the first safe position. It is the responsibility of the application that the path from the entering side of the track to the first safe position is clear.

When the carrier is entering the track, it starts moving to the first safe position. When the carrier moves beyond the handover position and the path to the end position is clear, the carrier starts moving automatically to the end position and comes to a standstill. This handover position is a position on the track where the carrier is only controlled by the receiving track. If the path to the end position is not clear, the carrier continues to move to the first safe position or starts moving to a calculated safe position beyond the first safe position.

Parameters:

in	trackId	Identification of the track.
in	carrierId	Identification of carrier.
in	beginPosition	Virtual begin position.
in	handoverPosition	Position on the track where the carrier is completely controlled by the receiving track.
in	firstSafePosition	Position on destination track where carrier can safely move to.
in	pCmMovePars	Movement parameters. The end position is defined on the track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_TRACK_INVALID_FEED_OVERRIDE
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_FIRST_SAFE_POSITION
- CM_ERR_INVALID_BEGIN_POSITION
- CM_ERR_INVALID_HANDOVER_POSITION
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TOO_MANY_CARRIERS_IN_PROCESS_AREA
  - firstSafePosition lies in or beyond an occupied process area.

State:

CM_TRACK_ACTIVE and feed override should be 1

Transition:

CM_COLL_AVOID_CARRIER_POWERUP -> CM_COLL_AVOID_CARRIER_MOVING (if carrier is detected by the outer sensor)

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED (if carrier stops at end position)

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if carrier stops at safe position)

CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS -> CM_COLL_AVOID_CARRIER_MOVING

1.13.2 CmCollAvoidAddMovingCarrier

```
NYCE_STATUS CmCollAvoidAddMovingCarrier(CM_INT              trackId,
                                        CM_INT              carrierId,
                                        CM_TRACK_POS        trackPositionId,
                                        CM_LREAL            firstSafePosition,
                                        const CM_MOVE_PARS* pCmMovePars);
```

Add a moving carrier to the system while the track is active with collision avoidance.

Call this function to add a moving carrier to the system while the track is active. Ideally the carrier should be moving at a constant velocity. This same velocity should be specified in the movement parameters. The end position should be chosen so the carrier is completely on the track. As soon as the carrier has been detected it will be added to the carrier administration.

The carrier will always move at least to the firstSafePosition (regardless whether this position is free). At this position the carrier state is CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS. If the path to the end position is clear, the carrier will immediately move to the end position. When the path is not clear, a safe position will be calculated and the carrier will stop at this intermediate position. At this position the carrier state is CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS. When the path is cleared, the carrier starts to move again. When the carrier stops at the end position the state will be CM_COLL_AVOID_CARRIER_STOPPED.

Parameters:

in	trackId	Identification of track.
in	carrierId	Identification of carrier.
in	trackPositionId	Identification of beginning or end of track.
in	firstSafePosition	Position on track where carrier can safely move to.
in	pCmMovePars	Movement parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_TRACK_INVALID_FEED_OVERRIDE
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_INVALID_TRACK_POSITION_ID
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_FIRST_SAFE_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_TOO_MANY_CARRIERS_IN_PROCESS_AREA
  - firstSafePosition lies in or beyond an occupied process area.

State:

CM_TRACK_ACTIVE and feed override should be 1

Transition:

CM_COLL_AVOID_CARRIER_POWERUP -> CM_COLL_AVOID_CARRIER_MOVING (if carrier is detected by the outer sensor)

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED (if carrier stops at end position)

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if carrier stops at intermediate position)

CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS -> CM_COLL_AVOID_CARRIER_MOVING

Carrier Management

1.13.3 CmCollAvoidCancelAddCarrierWithProfile

```
NYCE_STATUS CmCollAvoidCancelAddCarrierWithProfile(CM_INT trackId,
                                                   CM_INT carrierId);
```

Cancels the adding of a moving carrier to a track.

Call this function to cancel the addition of a moving carrier to the track before this carrier enters the track.

Parameters:

in	trackId	Identification of the track.
in	carrierId	Identification of carrier.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID

1.13.4 CmCollAvoidCancelAddMovingCarrier

```
NYCE_STATUS CmCollAvoidCancelAddMovingCarrier(CM_INT       trackId,
                                              CM_INT       carrierId,
                                              CM_TRACK_POS trackPositionId);
```

Cancel the addition of a moving carrier to the system with collision avoidance.

Call this function to cancel the addition of a moving carrier to the system before this carrier has been detected.

Parameters:

in	trackId	Identification of track.
in	carrierId	Identification of carrier.
in	trackPositionId	Identification of beginning or end of track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID

1.13.5 CmCollAvoidDefineProcessArea

```
NYCE_STATUS CmCollAvoidDefineProcessArea(CM_INT         processAreaId,
                                         CM_INT         trackId,
                                         const CM_AREA* pArea);
```

Define a collision avoidance process area on a track or redefine an existing process area.

The distance between the outer bounds of two successive process areas must be minimal the carrier length extended with the safe distance. The minimum and maximum position of a process area must be within the track boundaries.

Parameters:

in	processAreaId	Identification of process area.
in	trackId	Identification of track.
in	pArea	Area description.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_PROCESS_AREA_ID
  - invalid process area identifier specified.
- CM_ERR_TOO_MANY_PROCESS_AREAS
  - too many process areas in the system.
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_PROCESS_AREA_STATE_ERROR
  - process area with identifier exists and is enabled.
- CM_ERR_PARAMETER_ERROR

State:

any track state

if redefine: CM_COLL_AVOID_PROCESS_AREA_IDLE

1.13.6 CmCollAvoidDeleteProcessArea

```
NYCE_STATUS CmCollAvoidDeleteProcessArea(CM_INT processAreaId);
```

Delete a collision avoidance process area.

Parameters:

in	processAreaId	Identification of process area.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_PROCESS_AREA_ID
  - invalid process area identifier specified.
- CM_ERR_PROCESS_AREA_STATE_ERROR
  - existing process area identifier, but state is not idle.

State:

any track state

CM_COLL_AVOID_PROCESS_AREA_IDLE

1.13.7 CmCollAvoidEnableProcessArea

```
NYCE_STATUS CmCollAvoidEnableProcessArea(CM_INT  processAreaId,
                                         CM_BOOL onOff);
```

Enable or disable a process area.

Parameters:

in	processAreaId	Identification of process area.
in	onOff	Indicating if the process area must be enabled (TRUE) or disabled (FALSE).

Retvals:

Carrier Management

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_PROCESS_AREA_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_TOO_MANY_CARRIERS_IN_PROCESS_AREA
- CM_WRN_TOO_MANY_CARRIERS_IN_PROCESS_AREA

State:

if enable: CM_TRACK_STATE_ACTIVE and if disable: any track state

any process area state

Transition:

CM_COLL_AVOID_PROCESS_AREA_IDLE -> CM_COLL_AVOID_PROCESS_AREA_EMPTY (if enabled and no carrier present)

CM_COLL_AVOID_PROCESS_AREA_IDLE -> CM_COLL_AVOID_PROCESS_AREA_OCCUPIED (if enabled and carrier(s) present)

CM_COLL_AVOID_PROCESS_AREA_EMPTY -> CM_COLL_AVOID_PROCESS_AREA_IDLE (if disabled and no carrier present)

CM_COLL_AVOID_PROCESS_AREA_OCCUPIED -> CM_COLL_AVOID_PROCESS_AREA_IDLE (if disabled and carrier(s) present)

1.13.8 CmCollAvoidGetProcessAreaData

```
NYCE_STATUS CmCollAvoidGetProcessAreaData(CM_INT                  trackId,
                                          CM_COLL_AVOID_PROCESS_AREA_DATA
                                                                  processAreaData[proces
                                          sAreaDataLength],
                                          CM_UDINT                processAreaDataLength,
                                          CM_UDINT*               pNrOfProcessAreas);
```

Read the process area identifiers, area and state of all process areas (CM_COLL_AVOID_MAX_NR_OF_PROCESS_AREAS) of this track.

The data of each process area is stored in the output array. Entries in the array of non-existent process areas are indicated with processAreaId value CM_NO_ID. For these entries, zeros are written for the area positions and the state will be set to CM_COLL_AVOID_PROCESS_AREA_IDLE.

Parameters:

in	trackId	Identification of track.
out	processAreaData	Process area data.
in	processAreaDataLength	Allocated size of the array processAreaData in number of elements.
out	pNrOfProcessAreas	Number of valid process areas in the array processAreaData.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

- NYCE_ERR_INVALID_OUTPUT_SIZE
  - (processAreaDataLength is zero) or (processAreaDataLength greater than CM_COLL_AVOID_MAX_NR_OF_PROCESS_AREAS) or (number of process areas greater than processAreaDataLength).

1.13.9 CmCollAvoidMoveBetweenTracks

```
NYCE_STATUS CmCollAvoidMoveBetweenTracks(CM_INT              carrierId,
                                         CM_TRACK_POS        trackOriginPositionId,
                                         CM_INT              trackDestinationId,
                                         CM_TRACK_POS
                                                             trackDestinationPosition
                                         Id,
                                         CM_LREAL
                                                             trackDestinationFirstSaf
                                         ePosition,
                                         CM_LREAL            trackDistance,
                                         const CM_MOVE_PARS* pCmMovePars);
```

Move a carrier between tracks with collision avoidance.

The carrier moves from the begin position on the origin track to an end position on the adjacent destination track. Depending of the profile the carrier moves over the boundaries of the tracks while it is accelerating or decelerating.

The function provides collision avoidance during the complete movement from begin position on the origin track to end position on the destination track.

Parameters:

in	carrierId	Identification of carrier.
in	trackOriginPositionId	Identification of track side. Side where the carrier leaves the origin track.
in	trackDestinationId	Identification of destination track. Track which the carrier enters.
in	trackDestinationPositionId	Identification of track side. Side where the carrier enters the destination track.
in	trackDestinationFirstSafePosition	Position on destination track where carrier can safely move to.
in	trackDistance	Physical distance between the outer sensor of the origin and destination track in pu.
in	pCmMovePars	Movement parameters. The end position is defined on the destination track.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_TRACK_INVALID_FEED_OVERRIDE
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_TRACK_POSITION_ID
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_FIRST_SAFE_POSITION
- CM_ERR_INVALID_TRACK_DISTANCE
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TOO_MANY_CARRIERS_IN_PROCESS_AREA
  - trackDestinationFirstSafePosition lies in or beyond an occupied process area.

Carrier Management

State:

CM_TRACK_ACTIVE and feed override should be 1

CM_COLL_AVOID_CARRIER_STOPPED

Transition:

CM_COLL_AVOID_CARRIER_STOPPED -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if paths for initial movement are not clear)

CM_COLL_AVOID_CARRIER_STOPPED -> CM_COLL_AVOID_CARRIER_MOVING (if paths for initial movement are clear)

CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS -> CM_COLL_AVOID_CARRIER_MOVING

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if carrier stops at intermediate position)

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED (if carrier stops at end position)

1.13.10 CmCollAvoidMoveToPosition

```
NYCE_STATUS CmCollAvoidMoveToPosition(CM_INT              carrierId,
                                      const CM_MOVE_PARS* pCmMovePars);
```

Start the movement of a carrier to a specified end position with collision avoidance.

When the command is successfully completed, the command has been accepted.

If the path to the end position is clear, the carrier will immediately move to the end position. When the path is not clear, a safe position will be calculated and the carrier will stop at this intermediate position. At this position the carrier state is CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS. When the path is cleared, the carrier starts to move again. When the carrier stops at the end position the state will be CM_COLL_AVOID_CARRIER_STOPPED.

Parameters:

in	carrierId	Identification of carrier.
in	pCmMovePars	Movement parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_TRACK_INVALID_FEED_OVERRIDE
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_PARAMETER

State:

CM_TRACK_ACTIVE and feed override should be 1

any carrier state except CM_COLL_AVOID_CARRIER_POWERUP and CM_COLL_AVOID_CARRIER_SUSPENDED and CM_COLL_AVOID_CARRIER_JOGGING

Transition:

CM_COLL_AVOID_CARRIER_STOPPED -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if carrier stops at intermediate position)

CM_COLL_AVOID_CARRIER_STOPPED -> CM_COLL_AVOID_CARRIER_MOVING

CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS -> CM_COLL_AVOID_CARRIER_MOVING

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED (if carrier stops at end position)

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if carrier stops at intermediate position)

1.13.11 CmCollAvoidReadCarrierState

```
NYCE_STATUS CmCollAvoidReadCarrierState(CM_INT                    carrierId,
                                        CM_COLL_AVOID_CARRIER_STATE*
                                                                  pCarrierState);
```

Read carrier collision avoidance state.

Parameters:

in	carrierId	Identification of carrier.
out	pCarrierState	Collision avoidance carrier state.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- CM_ERR_CARRIER_STATE_ERROR

1.13.12 CmCollAvoidRemoveCarrierWithProfile

```
NYCE_STATUS CmCollAvoidRemoveCarrierWithProfile(CM_INT          carrierId,
                                                const CM_MOVE_PARS*    pCmMovePars);
```

Remove a moving carrier from a track with collision avoidance.

The carrier moves from a begin position on the leaving track to a virtual end position.

The function provides collision avoidance during the complete path of the begin position to the leaving side of the track. The carrier only starts moving when the path from the begin position of the carrier to the leaving side of the track is clear.

Parameters:

in	carrierId	Identification of carrier.
in	pCmMovePars	Movement parameters. Containing the virtual end position.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_TRACK_INVALID_FEED_OVERRIDE
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_END_POSITION
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK
- CM_ERR_INVALID_PARAMETER

Carrier Management

State:

CM_TRACK_ACTIVE and feed override should be 1

CM_COLL_AVOID_CARRIER_STOPPED

Transition:

CM_COLL_AVOID_CARRIER_STOPPED -> CM_COLL_AVOID_CARRIER_MOVING (if path is clear)

CM_COLL_AVOID_CARRIER_STOPPED -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if path is not clear)

CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS -> CM_COLL_AVOID_CARRIER_MOVING (if path is clear)

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_POWERUP (if carrier is no longer detected by the outer sensor)

1.13.13 CmCollAvoidRemoveMovingCarrier

```
NYCE_STATUS CmCollAvoidRemoveMovingCarrier(CM_INT              carrierId,
                                           CM_TRACK_POS        trackPositionId,
                                           const CM_VELOCITY_PARS*
                                                               pCmVelocityPars);
```

Remove a carrier from the track while it is moving in position mode with collision avoidance.

Call this function to specify a velocity with which a carrier should move beyond the end of the track on which it is currently present. It can be used to remove a carrier from the system while it remains moving in position mode. After the carrier is no longer detected by the outer sensor of the last coil of the track it will be removed from the carrier administration.

If the path to the outer sensor of the last coil of the track is clear, the carrier will be removed from the track. When the path is not clear, a safe position will be calculated and the carrier will stop at this intermediate position. At this position the carrier state is CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS. When the path is cleared, the carrier starts to move again.

Parameters:

in	carrierId	Identification of carrier.
in	trackPositionId	Identification of beginning or end of track.
in	pCmVelocityPars	Velocity parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_TRACK_INVALID_FEED_OVERRIDE
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_PARAMETER
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_TRACK_POSITION_ID
- CM_ERR_INVALID_VELOCITY
- CM_ERR_INVALID_ACCELERATION
- CM_ERR_INVALID_JERK

State:

CM_TRACK_ACTIVE and feed override should be 1

any carrier state except CM_COLL_AVOID_CARRIER_POWERUP and CM_COLL_AVOID_CARRIER_SUSPENDED and CM_COLL_AVOID_CARRIER_JOGGING

Transition:

CM_COLL_AVOID_CARRIER_STOPPED -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if carrier stops at intermediate position)

CM_COLL_AVOID_CARRIER_STOPPED -> CM_COLL_AVOID_CARRIER_MOVING

CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS -> CM_COLL_AVOID_CARRIER_MOVING

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS (if carrier stops at intermediate position)

CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_POWERUP (if carrier is no longer detected by the outer sensor)

1.13.14 CmCollAvoidStopCarrier

```
NYCE_STATUS CmCollAvoidStopCarrier(CM_INT              carrierId,
                                   const CM_STOP_PARS* pCmStopPars);
```

Initiate a stop profile for a carrier with collision avoidance.

When the command is successfully completed, the carrier has initiated the stop, or is already at standstill.

Parameters:

in	carrierId	Identification of carrier.
in	pCmStopPars	Stop parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_TRACK_STATE_ERROR
- CM_ERR_CARRIER_STATE_ERROR
- CM_ERR_INVALID_PARAMETER

State:

CM_TRACK_ACTIVE or CM_TRACK_ERROR or CM_TRACK_FATAL_ERROR

any carrier state except CM_COLL_AVOID_CARRIER_POWERUP and CM_COLL_AVOID_CARRIER_SUSPENDED and CM_COLL_AVOID_CARRIER_JOGGING

Transition:

If CM_TRACK_ACTIVE: CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS -> CM_COLL_AVOID_CARRIER_STOPPED

If CM_TRACK_ACTIVE: CM_COLL_AVOID_CARRIER_MOVING -> CM_COLL_AVOID_CARRIER_STOPPED (after completion of stop)

1.13.15 CmCollAvoidSynchronize

```
NYCE_STATUS CmCollAvoidSynchronize(CM_ID           cmId,
                                   CM_SYNC_REQUEST syncRequest,
                                   CM_LREAL        timeout);
```

Wait until synchronize request has been completed.

CmCollAvoidSynchronize supports synchronize requests:

- CM_REQ_CARRIER_MOVEMENT_STOPPED,
- CM_REQ_CARRIER_MOVING_CARRIER_ADDED_AND_CONTROLLED and
- CM_REQ_CARRIER_MOVING_CARRIER_REMOVED.

With timeout = 0, CmCollAvoidSynchronize checks on completion of the request. With timeout = NYCE_INFINITE, CmCollAvoidSynchronize waits indefinitely on completion the request.

Carrier Management

For CM_REQ_CARRIER_MOVEMENT_STOPPED: CmCollAvoidSynchronize checks or waits until either timeout has expired or the carrier stops at the end position. When the carrier stops at an intermediate position the request CM_REQ_CARRIER_MOVEMENT_STOPPED is not fulfilled.

Parameters:

in	cmId	Identification of CM object. The identification refers to a track or carrier.
in	syncRequest	Synchronize request.
in	timeout	Time-out [s].

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_PARAMETER_ERROR
- CM_ERR_REQUEST_TIMEOUT
- CM_ERR_REQUEST_ABORTED
- CM_ERR_ASYNC_ERROR_OCCURRED

1.14 Marker functionality

The module contains the functions and events of the Marker functionality.

1.14.1 CmDefinePermanentMarker

```
NYCE_STATUS CmDefinePermanentMarker(CM_INT                trackId,
                                    const CM_MARKER_PARS* pCmMarkerPars);
```

Define a permanent marker.

When a carrier crosses the marker position the output is activated.

Parameters:

in	trackId	Identification of track.
in	pCmMarkerPars	Marker parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_MARKER_ID
- CM_ERR_INVALID_POSITION
- SYS_ERR_INVALID_NODE_NAME
- CM_ERR_MARKER_ID_ALREADY_DEFINED
- CM_ERR_TOO_MANY_MARKERS_DEFINED

1.14.2 CmDefineSingleShotMarker

```
NYCE_STATUS CmDefineSingleShotMarker(CM_INT                 trackId,
                                     CM_INT                 carrierId,
                                     const CM_MARKER_PARS* pCmMarkerPars);
```

Define a single shot marker.

When the carrier crosses the marker position the output is activated and the marker is automatically deleted.

Parameters:

in	trackId	Identification of track.
in	carrierId	Identification of carrier.
in	pCmMarkerPars	Marker parameters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_CARRIER_ID
- CM_ERR_INVALID_MARKER_ID
- CM_ERR_INVALID_POSITION
- CM_ERR_TRACK_STATE_ERROR
- SYS_ERR_INVALID_NODE_NAME

State:

CM_TRACK_INACTIVE or CM_TRACK_ACTIVE or CM_TRACK_ERROR.

1.14.3 CmDeletePermanentMarker

```
NYCE_STATUS CmDeletePermanentMarker(CM_INT markerId);
```

Delete a permanent marker.

Parameters:

in	markerId	Identification of marker.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_MARKER_ID

1.14.4 CmDeleteSingleShotMarkers

```
NYCE_STATUS CmDeleteSingleShotMarkers(CM_INT trackId);
```

Delete all single shot markers defined on a track.

Parameters:

in	trackId	Identification of track.

Carrier Management

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_ERROR
- CM_ERR_INVALID_TRACK_ID

1.15 Parameter functionality

The module contains the identifiers and functions of the parameter functionality.

1.15.1 CmReadLogicalCoilName

```
NYCE_STATUS CmReadLogicalCoilName(CM_INT   trackId,
                                  CM_DINT  coilIndex,
                                  CM_INT   alternative,
                                  char     coilName[coilNameLength],
                                  CM_UDINT coilNameLength);
```

Read the logical coil name.

Parameters:

in	trackId	Identification of track.
in	coilIndex	Index of coil.
in	alternative	Indication of alternative.
out	coilName	Allocated buffer which receives the coil name.
in	coilNameLength	Allocated size of coilName in number of characters.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_COIL_INDEX
- CM_ERR_INVALID_COIL_ALTERNATIVE
- CM_ERR_INVALID_OUTPUT_ARGUMENT
- NYCE_ERR_INVALID_OUTPUT_SIZE

State:

CM_SYSTEM_INITIALIZED

1.15.2 CmReadLogicalCoilParameter

```
NYCE_STATUS CmReadLogicalCoilParameter(CM_INT    trackId,
                                       CM_DINT   coilIndex,
                                       CM_INT    alternative,
                                       CM_PAR_ID parId,
                                       CM_LREAL* pValue);
```

Read the value of the specified logical coil parameter.

Parameters:

in	trackId	Identification of track.
in	coilIndex	Index of coil.
in	alternative	Indication of alternative.
in	parId	Identification of parameter.
out	pValue	Value of the parameter.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_COIL_INDEX
- CM_ERR_INVALID_COIL_ALTERNATIVE
- CM_ERR_INVALID_PAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

State:

CM_SYSTEM_INITIALIZED

1.15.3 CmReadPhysicalCoilParameter

```
NYCE_STATUS CmReadPhysicalCoilParameter(const char* coilName,
                                        CM_PAR_ID   parId,
                                        CM_LREAL*   pValue);
```

Read the value of the specified physical coil parameter.

Parameters:

in	coilName	Coil name.
in	parId	Identification of parameter.
out	pValue	Value of the parameter.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_COIL_NAME
- CM_ERR_INVALID_PAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

State:

CM_SYSTEM_INITIALIZED

1.15.4 CmReadSensorParameter

```
NYCE_STATUS CmReadSensorParameter(const char*  coilName,
                                  CM_SENSOR_ID sensorId,
                                  CM_PAR_ID    parId,
                                  CM_LREAL*    pValue);
```

Read the value of the specified sensor parameter.

Carrier Management

Parameters:

in	coilName	Coil name.
in	sensorId	Identification of sensor.
in	parId	Identification of parameter.
out	pValue	Value of the parameter.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_INVALID_PARAMETER
- CM_ERR_INVALID_COIL_NAME
- CM_ERR_INVALID_SENSOR_ID
- CM_ERR_INVALID_PAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

State:

CM_SYSTEM_INITIALIZED

1.15.5 CmReadSystemParameter

```
NYCE_STATUS CmReadSystemParameter(CM_PAR_ID parId,
                                  CM_LREAL* pValue);
```

Read the value of the specified system parameter.

Parameters:

in	parId	Identification of parameter.
out	pValue	Value of the parameter.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_INVALID_PAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

State:

CM_SYSTEM_INITIALIZED

1.15.6 CmReadTrackParameter

```
NYCE_STATUS CmReadTrackParameter(CM_INT    trackId,
                                 CM_PAR_ID parId,
                                 CM_LREAL* pValue);
```

Read the value of the specified track parameter.

Parameters:

in	trackId	Identification of track.
in	parId	Identification of parameter.
out	pValue	Value of the parameter.

Retvals:

- NYCE_OK
  - successful
- CM_ERR_CLIENT_NOT_CONNECTED
- CM_ERR_SHUTDOWN_IN_PROGRESS
- CM_ERR_SYSTEM_STATE_ERROR
- CM_ERR_INVALID_TRACK_ID
- CM_ERR_INVALID_PAR_ID
- CM_ERR_INVALID_OUTPUT_ARGUMENT

State:

CM_SYSTEM_INITIALIZED

2 NYCe4000 API datatypes and definitions

2.1 Carrier Management

Carrier Management (CM) contains the functions available to drive the Linear Motion System (LMS).

2.1.1 Conversion

The functions within this module convert strings to enumeration values and vice versa.

2.1.1.1 CM_EVENT_CATEGORY

Every event belongs to a category.

```
typedef enum cm_event_category
{
   CM_EVENT_CATEGORY_UNSPECIFIED,
   CM_EVENT_CATEGORY_SYSTEM_EVENTS,
   CM_EVENT_CATEGORY_TRACK_EVENTS,
   CM_EVENT_CATEGORY_CARRIER_EVENTS,
   CM_EVENT_CATEGORY_CARRIER_CONTROLLER_EVENTS,
   CM_EVENT_CATEGORY_MARKER_EVENTS


} CM_EVENT_CATEGORY;
```

2.1.1.2 CM_NR_OF_EVENTS

Number of CM_EVENTs.

2.1.1.3 CM_PAR_CATEGORY

Every parameter belongs to a category. These categories are used in the NYCe tools to show sets of parameters.

```
typedef enum cm_par_category
{
   CM_PAR_CATEGORY_UNSPECIFIED,
   CM_PAR_CATEGORY_SYSTEM_PARAMETERS,
   CM_PAR_CATEGORY_TRACK_PARAMETERS,
   CM_PAR_CATEGORY_PHYSICAL_COIL_PARAMETERS,
   CM_PAR_CATEGORY_LOGICAL_COIL_PARAMETERS,
   CM_PAR_CATEGORY_SENSOR_PARAMETERS


} CM_PAR_CATEGORY;
```

2.1.1.4 CM_PAR_NR_OF_IDS

Number of CM_PAR_IDs.

2.1.1.5 CM_VAR_CATEGORY

Every variable belongs to a category. These categories are used in the NYCe tools to show sets of variables.

```
typedef enum cm_var_category
{
   CM_VAR_CATEGORY_UNSPECIFIED,
   CM_VAR_CATEGORY_SYSTEM_VARIABLES,
   CM_VAR_CATEGORY_TRACK_VARIABLES,
   CM_VAR_CATEGORY_CARRIER_VARIABLES,
   CM_VAR_CATEGORY_COIL_VARIABLES


} CM_VAR_CATEGORY;
```

2.1.1.6 CM_VAR_NR_OF_IDS
Number of CM_VAR_IDs.

2.1.2 Variables

The functions within this module are used to access CM variables.

2.1.2.1 System variables
2.1.2.1.1 CM_VAR_SYSTEM_CPA_ENABLE
CPA functionality is enabled (TRUE) or disabled (FALSE).

Data type: CM_BOOL

2.1.2.1.2 CM_VAR_SYSTEM_STATE
System state.

Data type: CM_SYSTEM_STATE

2.1.2.2 Track variables
2.1.2.2.1 CM_VAR_TRACK_BEGIN_ALTERNATIVE
Track begin alternative.

Data type: CM_INT
Unit: [-]

2.1.2.2.2 CM_VAR_TRACK_BEGIN_POS
Track begin position.

Data type: CM_LREAL
Unit: [pu]

2.1.2.2.3 CM_VAR_TRACK_END_ALTERNATIVE
Track end alternative.

Data type: CM_INT
Unit: [-]

2.1.2.2.4 CM_VAR_TRACK_END_POS
Track end position.

Data type: CM_LREAL
Unit: [pu]

2.1.2.2.5 CM_VAR_TRACK_FEED_OVERRIDE_CONSTANT
Track feed override is constant indication.

Data type: CM_BOOL

2.1.2.2.6 CM_VAR_TRACK_FEED_OVERRIDE_VALUE
Track feed override value.

Data type: CM_LREAL
Unit: [-]

2.1.2.2.7 CM_VAR_TRACK_FIRST_COIL_INDEX
Track first coil index.

NYCe4000 API datatypes and definitions

Data type: CM_INT
Unit: [-]

2.1.2.2.8 CM_VAR_TRACK_FIRST_ERROR_COIL
Track first error coil.

Data type: CM_INT
Unit: [-]

2.1.2.2.9 CM_VAR_TRACK_LAST_COIL_INDEX
Track last coil index.

Data type: CM_INT
Unit: [-]

2.1.2.2.10 CM_VAR_TRACK_STATE
Track state.

Data type: CM_TRACK_STATE

2.1.2.3 Carrier variables
2.1.2.3.1 CM_VAR_CARRIER_ACCELERATION
Carrier acceleration.

Data type: CM_LREAL
Unit: [pu/s^2]

2.1.2.3.2 CM_VAR_CARRIER_CONTROLLER_STATE
Carrier controller states.

Data type: CM_CARRIER_CONTROLLER_STATE

2.1.2.3.3 CM_VAR_CARRIER_CPA_TABLE_ID
The CPA table identifier to which the carrier is linked.

Data type: CM_INT
Unit: [-]

2.1.2.3.4 CM_VAR_CARRIER_DETECTING_TRACK_ID
The track identifier of the detecting track.

Data type: CM_INT
Unit: [-]

2.1.2.3.5 CM_VAR_CARRIER_JERK
Carrier jerk.

Data type: CM_LREAL
Unit: [pu/s^3]

2.1.2.3.6 CM_VAR_CARRIER_POSITION
Carrier position.

Data type: CM_LREAL
Unit: [pu]

2.1.2.3.7 CM_VAR_CARRIER_POSITION_DETECTING_TRACK
Carrier position on detecting track.

Data type: CM_LREAL
Unit: [pu]

2.1.2.3.8 CM_VAR_CARRIER_SETPOINT_STOPPED_POSITION
Carrier setpoint stopped position.

Data type: CM_LREAL
Unit: [pu]

2.1.2.3.9 CM_VAR_CARRIER_STATE
Carrier states for track state CM_TRACK_ACTIVE.

Data type: CM_CARRIER_STATE

2.1.2.3.10 CM_VAR_CARRIER_TRACK_ID
The track identifier on which the carrier is present.

Data type: CM_INT
Unit: [-]

2.1.2.3.11 CM_VAR_CARRIER_VELOCITY
Carrier velocity.

Data type: CM_LREAL
Unit: [pu/s]

2.1.2.4 Coil variables
2.1.2.4.1 CM_VAR_COIL_ERROR_CODE
Coil error code.

Data type: NYCE_ERROR_CODE

2.1.2.4.2 CM_VAR_COIL_LEFT_SENSOR_CARRIER_DETECTED
Left sensor detected carrier.

Data type: CM_BOOL

2.1.2.4.3 CM_VAR_COIL_LEFT_SENSOR_CARRIER_MOVING
Left sensor moving carrier.

Data type: CM_BOOL

2.1.2.4.4 CM_VAR_COIL_LEFT_SENSOR_CARRIER_POSITION
Left sensor position.

Data type: CM_LREAL
Unit: [pu]

2.1.2.4.5 CM_VAR_COIL_LEFT_SENSOR_CARRIER_POS_VALID

NYCe4000 API datatypes and definitions

Left sensor position valid.

Data type: CM_BOOL

2.1.2.4.6 CM_VAR_COIL_LEFT_SENSOR_DISABLED
Left sensor disabled.

Data type: CM_BOOL

2.1.2.4.7 CM_VAR_COIL_LEFT_SENSOR_ERROR
Left sensor error detected.

Data type: CM_BOOL

2.1.2.4.8 CM_VAR_COIL_MIDDLE_SENSOR_CARRIER_DETECTED
Middle sensor detected carrier.

Data type: CM_BOOL

2.1.2.4.9 CM_VAR_COIL_MIDDLE_SENSOR_CARRIER_MOVING
Middle sensor moving carrier.

Data type: CM_BOOL

2.1.2.4.10 CM_VAR_COIL_MIDDLE_SENSOR_CARRIER_POSITION
Middle sensor position.

Data type: CM_LREAL
Unit: [pu]

2.1.2.4.11 CM_VAR_COIL_MIDDLE_SENSOR_CARRIER_POS_VALID
Middle sensor position valid.

Data type: CM_BOOL

2.1.2.4.12 CM_VAR_COIL_MIDDLE_SENSOR_DISABLED
Middle sensor disabled.

Data type: CM_BOOL

2.1.2.4.13 CM_VAR_COIL_MIDDLE_SENSOR_ERROR
Middle sensor error detected.

Data type: CM_BOOL

2.1.2.4.14 CM_VAR_COIL_POSITION
Coil position.

Data type: CM_LREAL
Unit: [pu]

2.1.2.4.15 CM_VAR_COIL_RIGHT_SENSOR_CARRIER_DETECTED
Right sensor detected carrier.

Data type: CM_BOOL

2.1.2.4.16 CM_VAR_COIL_RIGHT_SENSOR_CARRIER_MOVING
Right sensor moving carrier.

Data type: CM_BOOL

2.1.2.4.17 CM_VAR_COIL_RIGHT_SENSOR_CARRIER_POSITION
Right sensor position.

Data type: CM_LREAL
Unit: [pu]

2.1.2.4.18 CM_VAR_COIL_RIGHT_SENSOR_CARRIER_POS_VALID
Right sensor position valid.

Data type: CM_BOOL

2.1.2.4.19 CM_VAR_COIL_RIGHT_SENSOR_DISABLED
Right sensor disabled.

Data type: CM_BOOL

2.1.2.4.20 CM_VAR_COIL_RIGHT_SENSOR_ERROR
Right sensor error detected.

Data type: CM_BOOL

2.1.2.5 CM_VAR_ID
CM variable ID type.

```
typedef uint32_t CM_VAR_ID;
```

2.1.3 Event functionality

The module contains the events and functions of the CM event functionality.

2.1.3.1 System events
2.1.3.1.1 CM_EV_SYSTEM_INITIALIZED
System initialized.

2.1.3.1.2 CM_EV_SYSTEM_SHUTDOWN
System shutdown.

2.1.3.2 Track events
2.1.3.2.1 CM_EV_TRACK_ACTIVE_EN
Track state active entered.

2.1.3.2.2 CM_EV_TRACK_ACTIVE_LE
Track state active left.

2.1.3.2.3 CM_EV_TRACK_ASYNC_ERROR_OCCURRED
Asynchronous track error.

2.1.3.2.4 CM_EV_TRACK_ASYNC_WARNING_OCCURRED
Asynchronous track warning.

2.1.3.2.5 CM_EV_TRACK_ERROR_EN
Track state error entered.

2.1.3.2.6 CM_EV_TRACK_ERROR_LE
Track state error left.

2.1.3.2.7 CM_EV_TRACK_FATAL_ERROR_EN

Track state fatal error entered.

2.1.3.2.8 CM_EV_TRACK_FATAL_ERROR_LE
Track state fatal error left.

2.1.3.2.9 CM_EV_TRACK_HOMED_EN
Track state homed entered.

2.1.3.2.10 CM_EV_TRACK_HOMED_LE
Track state homed left.

2.1.3.2.11 CM_EV_TRACK_IDLE_EN
Track state idle entered.

2.1.3.2.12 CM_EV_TRACK_IDLE_LE
Track state idle left.

2.1.3.2.13 CM_EV_TRACK_INACTIVE_EN
Track state inactive entered.

2.1.3.2.14 CM_EV_TRACK_INACTIVE_LE
Track state inactive left.

2.1.3.2.15 CM_EV_TRACK_LAYOUT_CHANGED
Track layout changed.

2.1.3.2.16 CM_EV_TRACK_MOVE_CARRIERS_AGAINST_ENDSTOP_FINISHED
Move carriers against endstop finished.

2.1.3.2.17 CM_EV_TRACK_MOVING_CARRIER_EN
Moving carrier enters track.

2.1.3.2.18 CM_EV_TRACK_MOVING_CARRIER_LE
Moving carrier leaves track.

2.1.3.2.19 CM_EV_TRACK_VELOCITY_MODE_EN
Track state velocity mode entered.

2.1.3.2.20 CM_EV_TRACK_VELOCITY_MODE_LE
Track state velocity mode left.

2.1.3.3 Carrier events

2.1.3.3.1 CM_EV_CARRIER_JOGGING_EN
Carrier state jogging entered.

2.1.3.3.2 CM_EV_CARRIER_JOGGING_LE
Carrier state jogging left.

2.1.3.3.3 CM_EV_CARRIER_MOVING_CARRIER_ADDED
Moving carrier added to track.

2.1.3.3.4 CM_EV_CARRIER_MOVING_CARRIER_REMOVED
Moving carrier removed from track.

2.1.3.3.5 CM_EV_CARRIER_MOVING_EN
Carrier state moving entered.

2.1.3.3.6 CM_EV_CARRIER_MOVING_LE
Carrier state moving left.

2.1.3.3.7 CM_EV_CARRIER_POWERUP_EN
Carrier state powerup entered.

2.1.3.3.8 CM_EV_CARRIER_POWERUP_LE
Carrier state powerup left.

2.1.3.3.9 CM_EV_CARRIER_STOPPED_EN
Carrier state stopped entered.

2.1.3.3.10 CM_EV_CARRIER_STOPPED_LE
Carrier state stopped left.

2.1.3.3.11 CM_EV_CARRIER_SUSPENDED_EN
Carrier state suspended entered.

2.1.3.3.12 CM_EV_CARRIER_SUSPENDED_LE
Carrier state suspended left.

2.1.3.4 Carrier controller events

2.1.3.4.1 CM_EV_CARRIER_CONTROLLER_MOVING_EN

Carrier controller state moving entered.

2.1.3.4.2 CM_EV_CARRIER_CONTROLLER_MOVING_LE
Carrier controller state moving left.

2.1.3.4.3 CM_EV_CARRIER_CONTROLLER_SETTLING_EN
Carrier controller state settling entered.

2.1.3.4.4 CM_EV_CARRIER_CONTROLLER_SETTLING_LE
Carrier controller state settling left.

2.1.3.4.5 CM_EV_CARRIER_CONTROLLER_STABILIZING_EN
Carrier controller state stabilizing entered.

2.1.3.4.6 CM_EV_CARRIER_CONTROLLER_STABILIZING_LE
Carrier controller state stabilizing left.

2.1.3.4.7 CM_EV_CARRIER_CONTROLLER_STEADY_EN
Carrier controller state steady entered.

2.1.3.4.8 CM_EV_CARRIER_CONTROLLER_STEADY_LE
Carrier controller state steady left.

2.1.3.5 Marker events
2.1.3.5.1 CM_EV_MARKER_PERMANENT
Permanent marker.

2.1.3.5.2 CM_EV_MARKER_SINGLE_SHOT
Single shot marker.

2.1.3.6 CM_EVENT_HANDLER

```
void CM_EVENT_HANDLER(CM_ID                   cmId,
                      CM_EVENT                eventId,
                      const NYCE_EVENT_INFO* pInfo,
                      void*                   pUserData);
```

Prototype for an CM event enrolment function.

Parameters:

in	cmId	- identification of CM object. The identification refers to a system, track or carrier.
in	eventId	- CM event identifier.
in	pInfo	- additional event data provided by the CM layer.
in	pUserData	- user specific data provided on the define of the event enrolment.

2.1.3.7 CM_ID
Definition of CM_ID for use with CM events.

```
typedef struct cm_id
{
   /* CM Object type. */
   CM_OBJECT_TYPE                  type;
   /* Identifier of a system (always 0), track, or carrier or index of a coil. */
   CM_INT                          id;

} CM_ID;
```

2.1.4 Collision avoidance functionality

The functions within this module are used to move a carrier with collision avoidance.

2.1.4.1 CM_COLL_AVOID_CARRIER_STATE

```
typedef enum cm_coll_avoid_carrier_state
{
   /* Carrier powerup state. */
   CM_COLL_AVOID_CARRIER_POWERUP    = 0,
   /* Carrier stopped state. */
   CM_COLL_AVOID_CARRIER_STOPPED,
```

```
   /* Carrier suspended state. */
   CM_COLL_AVOID_CARRIER_SUSPENDED,
   /* Carrier moving state. */
   CM_COLL_AVOID_CARRIER_MOVING,
   /* Carrier jogging state. */
   CM_COLL_AVOID_CARRIER_JOGGING,
   /* Carrier stopped at safe position state. */
   CM_COLL_AVOID_CARRIER_STOPPED_SAFE_POS


} CM_COLL_AVOID_CARRIER_STATE;
```

2.1.4.2 CM_COLL_AVOID_PROCESS_AREA_DATA

```
typedef struct cm_coll_avoid_process_area_data
{
   /* Process area identifier. */
   CM_INT                          processAreaId;
   /* Area definition of process area. */
   CM_AREA                         area;
   /* State of process area. */
   CM_COLL_AVOID_PROCESS_AREA_STATE state;

} CM_COLL_AVOID_PROCESS_AREA_DATA;
```

2.1.4.3 CM_COLL_AVOID_PROCESS_AREA_STATE

```
typedef enum cm_coll_avoid_process_area_state
{
   /* Process area disabled (no checks). */
   CM_COLL_AVOID_PROCESS_AREA_IDLE  = 0,
   /* Process area enabled, no carrier present. */
   CM_COLL_AVOID_PROCESS_AREA_EMPTY,
   /* Process area enabled, carrier present. */
   CM_COLL_AVOID_PROCESS_AREA_OCCUPIED


} CM_COLL_AVOID_PROCESS_AREA_STATE;
```

2.1.5 Marker functionality

The module contains the functions and events of the Marker functionality.

2.1.5.1 CM_MARKER_PARS

```
typedef struct cm_marker_pars
{
   /* Marker identifier. */
   CM_INT                          markerId;
   /* Marker position. */
   CM_LREAL                        position;
   /* Maker direction. Direction is negative, positive or any direction. */
   SAC_MARKER_DIRECTION            direction;
   /* Marker output action. Action is set, reset or toggle. */
   NYCE_DIGOUT_ACTION              outputAction;
   /* The node name where the marker output is located. */
   NYCE_NAME                       outputNodeName;
   /* Marker output identifier. Consists of the slot identifier and digital I/O
      number. */
   NYCE_DIGITAL_IO_ID              outputId;
   /* Indicates if a marker event is generated. */
   CM_BOOL                         eventFlag;

} CM_MARKER_PARS;
```

2.1.6 Parameter functionality

The module contains the identifiers and functions of the parameter functionality.

2.1.6.1 System parameters

2.1.6.1.1 CM_PAR_SYSTEM_NR_OF_TRACKS

Number of tracks in the system.

Data type: CM_DINT
Unit: [-]

2.1.6.2 Track parameters

2.1.6.2.1 CM_PAR_TRACK_CARRIER_LENGTH

Length of the carrier.

Data type: CM_LREAL
Unit: [pu]

2.1.6.2.2 CM_PAR_TRACK_MEASUREMENT_SCALE_ECCENTRICITY

Eccentricity of the measurement scale.

Data type: CM_LREAL
Unit: [pu]

2.1.6.2.3 CM_PAR_TRACK_MEASUREMENT_SCALE_NR_OF_PERIODS

Number of periods of the measurement scale.

Data type: CM_UDINT
Unit: [-]

2.1.6.2.4 CM_PAR_TRACK_MOTOR_MAGNET_ECCENTRICITY

Eccentricity of the magnet plate.

Data type: CM_LREAL
Unit: [pu]

2.1.6.2.5 CM_PAR_TRACK_MOTOR_MAGNET_NR_OF_POLE_PAIRS

Number of pole pairs of the magnet plate.

Data type: CM_UDINT
Unit: [-]

2.1.6.2.6 CM_PAR_TRACK_NR_OF_COILS

Number of coils in the track.

Data type: CM_DINT
Unit: [-]

2.1.6.2.7 CM_PAR_TRACK_NR_OF_SHARED_COILS_BEGIN

Number of shared coils at the begin of the track.

Data type: CM_DINT
Unit: [-]

2.1.6.2.8 CM_PAR_TRACK_NR_OF_SHARED_COILS_END

Number of shared coils at the end of the track.

NYCe4000 API datatypes and definitions

Data type: CM_DINT
Unit: [-]

2.1.6.2.9 CM_PAR_TRACK_SENSOR_TYPE
Type of sensor. (HALL sensor = 0, MR sensor = 1)

Data type: CM_INT
Unit: [-]

2.1.6.2.10 CM_PAR_TRACK_TYPE
Type of track (linear = 0, modulo = 1).

Data type: CM_INT
Unit: [-]

2.1.6.3 Physical coil parameters
2.1.6.3.1 CM_PAR_PHYSICAL_COIL_NR_OF_COIL_TRIPLETS
Number of coil triplets.

Data type: CM_INT
Unit: [-]

2.1.6.3.2 CM_PAR_PHYSICAL_COIL_NR_OF_SENSORS
Number of sensors.

Data type: CM_DINT
Unit: [-]

2.1.6.4 Logical coil parameters
2.1.6.4.1 CM_PAR_LOGICAL_COIL_POSITION
Position of the coil.

Data type: CM_LREAL
Unit: [pu]

2.1.6.5 Sensor parameters
2.1.6.5.1 CM_PAR_SENSOR_POS
Position of the sensor. Relative to the coil position.

Data type: CM_LREAL
Unit: [pu]

2.1.6.6 CM_PAR_ID
CM parameter ID type.

```
typedef uint32_t CM_PAR_ID;
```

2.1.6.7 CM_PAR_SECTION_TYPE

```
typedef enum cm_par_section_type
{
   /* Section system. */
   CM_PAR_SECTION_SYSTEM            = 0,
   /* Section track. */
   CM_PAR_SECTION_TRACK,
   /* Section physical coil. */
   CM_PAR_SECTION_PHYSICAL_COIL,
   /* Section logical coil. */
   CM_PAR_SECTION_LOGICAL_COIL,
```

```
   /* Section sensor. */
   CM_PAR_SECTION_SENSOR,
   /* Section coll avoid. */
   CM_PAR_SECTION_COLL_AVOID


} CM_PAR_SECTION_TYPE;
```

2.1.7 CM_AREA

```
typedef struct cm_area
{
   /* Minimum position in pu */
   CM_LREAL                         minPosition;
   /* Maximum position in pu */
   CM_LREAL                         maxPosition;

} CM_AREA;
```

2.1.8 CM_BOOL

Type definition to match interface with CM software

```
typedef int8_t CM_BOOL;
```

2.1.9 CM_BUMPER_PARS

```
typedef struct cm_bumper_pars
{
   /* When active, the bumper is assumed to be present en ready to home against lower
      or upper side. Also it is assumed the lowerSidePosition and upperSidePosition
      are calibrated. */
   CM_BOOL                          active;
   /* The home position of a carrier against the lower side of the bumper */
   CM_LREAL                         lowerSidePosition;
   /* The home position of a carrier against the upper side of the bumper */
   CM_LREAL                         upperSidePosition;

} CM_BUMPER_PARS;
```

2.1.10 CM_CARRIER_CONTROLLER_STATE

```
typedef enum cm_carrier_controller_state
{
   /* Carrier controller moving state. */
   CM_CARRIER_CONTROLLER_MOVING     = 100,
   /* Carrier controller settling state. */
   CM_CARRIER_CONTROLLER_SETTLING   = 101,
   /* Carrier controller stabilizing state. */
   CM_CARRIER_CONTROLLER_STABILIZING= 102,
   /* Carrier controller steady state. */
   CM_CARRIER_CONTROLLER_STEADY     = 103


} CM_CARRIER_CONTROLLER_STATE;
```

2.1.11 CM_CARRIER_DATA

```
typedef struct cm_carrier_data
{
   /* Carrier identifier */
   CM_INT                          carrierId;
   /* Position of carrier [pu] */
   CM_LREAL                        position;
   /* Velocity of carrier [pu/s] */
   CM_LREAL                        velocity;
   /* Acceleration of carrier [pu/s^2] */
   CM_LREAL                        acceleration;
   /* Jerk of carrier [pu/s^3] */
   CM_LREAL                        jerk;
   /* State of carrier */
   CM_CARRIER_STATE                state;
   /* State of carrier controller */
   CM_CARRIER_CONTROLLER_STATE     controllerState;

} CM_CARRIER_DATA;
```

2.1.12 CM_CARRIER_PARS

```
typedef struct cm_carrier_pars
{
   /* product offset [pu] */
   CM_LREAL                        productOffset;
   /* measurement scale offset [pu] */
   CM_LREAL                        measurementScaleOffset;

} CM_CARRIER_PARS;
```

2.1.13 CM_CARRIER_STATE

```
typedef enum cm_carrier_state
{
   /* Carrier powerup state, track state must be in CM_TRACK_ACTIVE state */
   CM_CARRIER_POWERUP              = 0,
   /* Carrier stopped state, track state must be in CM_TRACK_ACTIVE state */
   CM_CARRIER_STOPPED              = 1,
   /* Carrier suspended state, track state must be in CM_TRACK_ACTIVE state */
   CM_CARRIER_SUSPENDED            = 2,
   /* Carrier moving state, track state must be in CM_TRACK_ACTIVE state */
   CM_CARRIER_MOVING               = 3,
   /* Carrier jogging state, track state must be in CM_TRACK_ACTIVE state */
   CM_CARRIER_JOGGING              = 4


} CM_CARRIER_STATE;
```

2.1.14 CM_CONTROLLER_PARS

```
typedef struct cm_controller_pars
{
   /* Integrator gain in 1/s */
   CM_LREAL                        parKi;
   /* P gain at standstill [V/pu] */
   CM_LREAL                        parPGainAtStandStill;
```

```
    /* D gain at standstill [Vs/pu] */
    CM_LREAL                        parDGainAtStandStill;
    /* Saturation level [V] */
    CM_LREAL                        parSatLevel;
    /* Maximum dynamic following error [pu] */
    CM_LREAL                        parMaxDynFollowError;
    /* Maximum steady state error [pu] */
    CM_LREAL                        parMaxSteadyStateError;

} CM_CONTROLLER_PARS;
```

2.1.15 CM_DINT

Type definition to match interface with CM software

```
typedef int32_t CM_DINT;
```

2.1.16 CM_EVENT

Type definition of CM_EVENT.

```
typedef uint32_t CM_EVENT;
```

2.1.17 CM_FREE_AREA_PARS

```
typedef struct cm_free_area_pars
{
    /* begin position of area [pu] */
    CM_LREAL                        beginPosition;
    /* end position of area [pu] */
    CM_LREAL                        endPosition;
    /* defining with beginPosition the part of the area, which is freed from carriers
       [pu] */
    CM_LREAL                        freeAreaSize;
    /* velocity [pu/s] */
    CM_LREAL                        velocity;

} CM_FREE_AREA_PARS;
```

2.1.18 CM_HOME_PARS

```
typedef struct cm_home_pars
{
    /* Home mode */
    CM_INT                          homeMode;
    /* Home velocity in pu/s */
    CM_LREAL                        homeVelocity;
    /* Home allowed collision velocity in pu/s */
    CM_LREAL                        homeAllowedCollisionVelocity;
    /* Home edge offset in pu (in direction of the endstop based upon the middle
       sensor position of the coils on the edge of the track) */
    CM_LREAL                        homeEdgeOffset;
    /* Number of carriers on the track. */
    CM_INT                          homeNrOfCarriers;
    /* Carrier positions of carriers on the track in pu. */
    CM_LREAL                        homeCarrierPositions[CM_MAX_NR_OF_CARRIERS];

} CM_HOME_PARS;
```

NYCe4000 API datatypes and definitions

2.1.19 CM_INT

Type definition to match interface with CM software

```
typedef int16_t CM_INT;
```

2.1.20 CM_LREAL

Type definition to match interface with CM software

```
typedef double CM_LREAL;
```

2.1.21 CM_MAX_COIL_NAME_LENGTH

The maximum size of a string, defining the name of a coil. String length is defined including a zero terminator

2.1.22 CM_MAX_FILENAME_LENGTH

The maximum size of a string, defining the name of a file. String length is defined including a zero terminator

2.1.23 CM_MAX_MACHINE_NAME_LENGTH

The maximum size of a string, defining the name of a machine. String length is defined including a zero terminator

2.1.24 CM_MAX_NR_OF_CLIENTS

Maximum number of clients for CM. A client of CM is a process that connects to CM.
The maximum number of CM clients

2.1.25 CM_MAX_NR_OF_CPA_TABLES

The maximum number of CPA tables.

2.1.26 CM_MAX_NR_OF_CPA_VALUES

The maximum number of adjustment values in a CPA table.

2.1.27 CM_MOVE_PARS

```
typedef struct cm_move_pars
{
   /* End position in pu */
   CM_LREAL                        endPosition;
   /* Maximum velocity in pu/s */
   CM_LREAL                        maxVelocity;
   /* Maximum acceleration in pu/s2 */
   CM_LREAL                        maxAcceleration;
   /* Maximum jerk in pu/s3 */
   CM_LREAL                        maxJerk;

} CM_MOVE_PARS;
```

2.1.28 CM_OBJECT_SHIFT

Shift value for CM object type.

2.1.29 CM_OBJECT_TYPE

```
typedef enum cm_object_type
{
   /* CM Object type System. */
   CM_OBJECT_SYSTEM                = 0,
   /* CM Object type Track. */
   CM_OBJECT_TRACK,
   /* CM Object type Carrier. */
   CM_OBJECT_CARRIER,
   /* CM Object type Coil. */
   CM_OBJECT_COIL,
   /* CM Object type Marker. */
   CM_OBJECT_MARKER


} CM_OBJECT_TYPE;
```

2.1.30 CM_PROFILE_PARS

```
typedef struct cm_profile_pars
{
   /* Number of points in profile */
   CM_UINT                         nrOfPoints;
   /* Collection of profile points to meet */
   CM_PROFILE_POINT                profilePoint[CM_MAX_NR_OF_PROFILE_POINTS];

} CM_PROFILE_PARS;
```

2.1.31 CM_PROFILE_POINT

```
typedef struct cm_profile_point
{
   /* Position in pu */
   CM_LREAL                        position;
   /* Velocity in pu/s */
   CM_LREAL                        velocity;

} CM_PROFILE_POINT;
```

2.1.32 CM_REAL

Type definition to match interface with CM software

```
typedef float CM_REAL;
```

2.1.33 CM_SECTION_SHIFT

Shift value for configuration section type.

2.1.34 CM_SENSOR_ID

```
typedef enum cm_sensor_id
{
   CM_LEFT_SENSOR_ID               = 0,
   CM_MIDDLE_SENSOR_ID,
   CM_RIGHT_SENSOR_ID
```

```
} CM_SENSOR_ID;
```

2.1.35 CM_STOP_PARS

```
typedef struct cm_stop_pars
{
   /* Stop acceleration in pu/s2 */
   CM_LREAL                         stopAcceleration;
   /* Stop jerk in pu/s3 */
   CM_LREAL                         stopJerk;

} CM_STOP_PARS;
```

2.1.36 CM_SYNC_REQUEST

Synchronization request types.

```
typedef enum cm_sync_request
{
   /* Movement of carrier is started */
   CM_REQ_CARRIER_MOVEMENT_STARTED  = 0,
   /* Movement of carrier is stopped */
   CM_REQ_CARRIER_MOVEMENT_STOPPED,
   /* Movement of carrier is jogging */
   CM_REQ_CARRIER_MOVEMENT_JOGGING,
   /* Moving carrier is added and controlled by the track */
   CM_REQ_CARRIER_MOVING_CARRIER_ADDED_AND_CONTROLLED,
   /* Moving carrier is removed from the track */
   CM_REQ_CARRIER_MOVING_CARRIER_REMOVED,
   /* Homing procedure is completed */
   CM_REQ_TRACK_HOMING_COMPLETED,
   /* Carriers are against the endstop */
   CM_REQ_TRACK_CARRIERS_AGAINST_ENDSTOP


} CM_SYNC_REQUEST;
```

2.1.37 CM_SYSTEM_STATE

```
typedef enum cm_system_state
{
   /* System not configured state. */
   CM_SYSTEM_POWERUP                = 0,
   /* System in operational state. */
   CM_SYSTEM_INITIALIZED            = 1,
   /* System in progress of initialization. */
   CM_SYSTEM_INITIALIZATION_IN_PROGRESS= 2,
   /* System in progress of shutting down. */
   CM_SYSTEM_SHUTDOWN_IN_PROGRESS   = 3


} CM_SYSTEM_STATE;
```

2.1.38 CM_TRACK_POS

```
typedef enum cm_track_pos
{
   /* Beginning of track. */
   CM_TRACK_BEGIN                    = 0,
   /* End of track. */
   CM_TRACK_END                      = 1


} CM_TRACK_POS;
```

2.1.39 CM_TRACK_STATE

```
typedef enum cm_track_state
{
   /* Track powerup state */
   CM_TRACK_POWERUP                  = 0,
   /* Track idle state */
   CM_TRACK_IDLE                     = 1,
   /* Track velocity mode state */
   CM_TRACK_VELOCITY_MODE            = 2,
   /* Track homed state */
   CM_TRACK_HOMED                    = 3,
   /* Track inactive state */
   CM_TRACK_INACTIVE                 = 4,
   /* Track active state */
   CM_TRACK_ACTIVE                   = 5,
   /* Track error state */
   CM_TRACK_ERROR                    = 6,
   /* Track fatal error state */
   CM_TRACK_FATAL_ERROR              = 7


} CM_TRACK_STATE;
```

2.1.40 CM_TYPE_ID

```
typedef enum cm_type_id
{
   CM_TYPE_BOOL                      = 0,
   CM_TYPE_INT,
   CM_TYPE_UINT,
   CM_TYPE_DINT,
   CM_TYPE_UDINT,
   CM_TYPE_REAL,
   CM_TYPE_LREAL,
   CM_TYPE_ENUM,
   CM_TYPE_UNSPECIFIED


} CM_TYPE_ID;
```

2.1.41 CM_TYPE_SHIFT

Shift value for CM data type.

2.1.42 CM_UDINT

Type definition to match interface with CM software

```
typedef uint32_t CM_UDINT;
```

2.1.43 CM_UINT

Type definition to match interface with CM software

```
typedef uint16_t CM_UINT;
```

2.1.44 CM_VELOCITY_PARS

```
typedef struct cm_velocity_pars
{
   /* Desired velocity in pu/s */
   CM_LREAL                          velocity;
   /* Maximum acceleration in pu/s2 */
   CM_LREAL                          maxAcceleration;
   /* Maximum jerk in pu/s3 */
   CM_LREAL                          maxJerk;

} CM_VELOCITY_PARS;
```

Notes

rexroth
A Bosch Company

Bosch Rexroth AG
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr a.Main
Germany
Tel. +49 9352 18 0
Fax +49 9352 18 8400
www.boschrexroth.com/electrics

R911425760

DOK-NY4000-LMS*REF*V54-RE02-EN-E