cosin
scientific software

Gipser + Hofmann, Ingenieure, Partnerschaft

Hans-Stützle-Str. 20
81249 München
GERMANY

info@cosin.eu
www.cosin.eu

CTI

Cosin Tire Interface

API Reference and User's Guide

Contents

Document Revision:2018-4-r18799

1. Overview on Interfaces to the FTire Tire Model Family

This documentation chapter describes the program interfaces to the FTire Tire Model Family. The preferred generic interface is called CTI (cosin Tire Interface). For more material about FTire, HTire, and other members of the FTire Tire Model Family, as well as about related tools, please visit www.cosin.eu.

1.1. Reference Platforms for the FTire Tire Model Family

All members of the FTire Tire Model Family, comprising HTire, FTire, and FETire, are available for Windows XP,7 8,10 or later, RHEL5,6,7 or later, Mac OS X 10.7,8,9,10,11 or later. All platforms are supported for 32bit and 64bit operating systems.

1.2. Tire Model Calling Modes and Interface Types

There are four classes of interfaces tlo the FTire Tire Model Family available:

- a **time-discrete basic internal interface** that fits, in an optimized way, to the program structure of the FTire Tire Model Family. This interface is used in *cosin/mbs* and all FTire tools and add-ons, but neither recommended nor documented for use outside the cosin environment;
- a **time-discrete generic interface (CTI)**, being an 'easy-to-use' and very comprehensive API which reduces the implementation effort to a minimum. CTI provides unrestricted access to all FTire features, without requiring any knowledge of internal implementation specialties. This interface is used by the FTire Tire Model Family implementations in all Adams variants, Simpack, DADS/VirtualLab Motion, Altair MotionSolve, RecurDyn, FEDEM, CarSim/TruckSim/BikeSim, Matlab/Simulink (inside FTire/link, see below), PAM-CRASH, and many others;
- the **time-continuous TYDEX/STI interface** (STI Version 1.4). This interface had been defined for use in commercial MBS-codes (cf.:**TYDEX-Format (Release 1.3) and Standard Tyre Interface (Release 1.4)**. Presented at: 2^{nd} International Colloquium on Tyre Models for Vehicle Dynamics Analysis. February 20-21, 1997). All existing implementations of STI with respect to the FTire Tire Model Family are nothing but additional program layers, internally using and calling CTI functions;
- an interface to Matlab/Simulink (FTire/link). Again, the implementation discussed here is nothing but another layer in terms of an S-function, using and calling appropriate CTI functions. This S-function is completed by a respective Simulink block-set.

All interfaces mentioned are such that arbitrarily many instances of the same or different members of the FTire Tire Model Family can be run simultaneously, either in sequential or in parallel (multi-threaded) mode. In either case, the coupling to the vehicle or suspension model of the calling program is done by the **rigid body state variables** of the rim, that is

- **position** of the rim center in the global coordinates,
- **translational velocity vector** of the rim center,
- **angular orientation** of the rim, defined by the transformation matrix from the rim-fixed frame to global coordinates. If needed, the FTire Tire Model Family provides routines to calculate this transformation matrix out of the Euler angles, Cardan angles, or Euler parameters of the rim, depending on what is available in the calling simulation model,
- **rotational velocity vector** of the rim.

Document Revision:2018-4-r18799

These values are the inputs to the respective member of the FTire Tire Model Family. On the other hand, all supported tire models provide as output

- the **tire force vector**, acting on the rim
- the **tire torque vector**, acting on the rim.

Point of reference for forces and moments is chosen to be the rim center.

Alternatively, all members of the FTire Tire Model Family can be used to simultaneously integrate the rim rotation w.r.t. the hub-carrier. In this use mode, not the rigid-body states of the rim, but those of the **wheel-carrier** are the inputs, together with the driving and the maximum absolute braking torque. The output torque vector then does not contain the share in the direction of the wheel rotation. All members of the FTire Tire Model Family eventually **modulate** the braking torque, if the wheel is blocked, in order to exactly maintain this blocking as long as it is necessary.

Document Revision:2018-4-r18799

2. CTI Features and Overview

The cosin Tire Interface **(CTI)** is the preferred interface to all members of the FTire Tire Model Family. Additionally, **CTI** can be used to connect to certain user-definable types of tire models.

CTI is designed and optimized especially for time-discrete implementations of tire models, being called by arbitrary types of integrators, using both fixed step-size or controlled step-size. There is, however, also support for 3^{rd}-party tire models which require integration of state variables by the calling software, and which are connected via the STI interface. **CTI** provides simple access to the FTire model family not only for commercial MBS software, but also for customer-specific 'in-house' simulation packages.

CTI is implemented in terms of an API (Application Programming Interface), consisting of several routines, which can be called from C and C++ programs. In Windows, the API is realized as a dynamic link library (.dll). For Unix and Linux platforms, it is a shared object (.so, .sl), whereas, on Apple Macintosh, it is a dynamic library (.dylib).

CTI and all members of the FTire Tire Model Family support all *cosin/road* road models, as described in the respective documentation chapter.

The API provides external entries as listed below. Most important entries are `ctiComputeForces` and its alternatives `ctiComputeForcesTimeContinuous`, `ctiComputeForcesOnWheelCarrier` and `ctiComputeForcesOnCarBody`.

CTI keeps and manages all parameters, state variables, working arrays etc. internally, for up to 100 tire instances. It is very easy to increase this maximum number of tires if necessary. Memory for all arrays is allocated dynamically. By this, **CTI** uses dynamic memory allocation and is designed for a small RAM footprint.

Pre-processing, initialization, time step management etc. is done automatically, inside the core routines `ctiComputeForces`, `ctiComputeForcesTimeContinuous`, `ctiComputeForcesWithExtRoad` and `ctiComputeForcesOnWheelCarrier`. In most cases, only one of these core routines will be used at a time. However, they could be used simultaneously as well.

The two routines `ctiLoadTireData` and `ctiLoadRoadData` are called only once for each tire instance. They provide tire or road data, respectively, by opening, reading, and pre-processing the respective data files. Both routines automatically recognize whether a file already had been opened by another tire instance. If so, the pre-processed data will just be copied instead of being re-calculated.

In case **CTI** is used to call a 3^{rd}-party, STI-compatible tire model, the two routines mentioned above have to be completed by additional ones: `ctiLoadSTITireModel` and `ctiLoadSTIRoadModel`. These routines specify dynamically linkable libraries and modul names, containing the standardized STI routines `dtyre` and `road`, respectively.

Routine `ctiReadOperatingConditions` reads and loads operating conditions that may optionally be specified in the tire data file.

With routine `ctiGetTireKeyData`, several important tire properties can be queried, like static, dynamic, and maximum tire radius, mass, moments of inertia, etc. These data might be needed by the calling vehicle simulation program.

The three routines `ctiSetInflationPressure`, `ctiSetTreadDepth` and `ctiSetAmbientTemperature` are used to set and control tire operating conditions (inflation pressure, tread depth, and temperature), which might vary with time. Typically, they are called before, or occasionally during, a running simulation.

The two optional routines `ctiSetRoadMotionData` and `ctiGetRoadForces` are used to introduce an arbitrary body as 'moving road', and to apply the tire reaction forces to this body. Note that, due to belt inertia forces

Document Revision:2018-4-r18799

and tire weight, the forces and moments acting on the rim will differ not only in sign from the ones acting on the road.

The routines `ctiOpenOutputFile`, `ctiVerbose`, `ctiGetTydexSignals`, `ctiGetOutputSignal`, `ctiGetNodePositions`, `ctiUpdateWheelEnvelope` and `ctiWriteWheelEnvelope` are optional as well, and are used for auxiliary or diagnostic output specification. With `ctiOpenOutputFile`, the name of an additional output file may be defined. This file will contain several columns of additional output signals and can be loaded into Matlab, say, for further analysis. `ctiVerbose` allows to set or unset verbosity mode. `ctiGetTydexSignals` provides the standard TYDEX/STI output variables (named`VARINF` in TYDEX/STI) together with additional tire-model-specific output signals. `ctiGetOutputSignal` searches for, and eventually puts out, a single output signal. `ctiGetNodePositions` returns the actual position of tire surface nodes in several co-ordinate systems to be used for customer-specific animation, etc.

The routines `ctiReadStates` and `ctiWriteStates` can be used to save and restore complete tire model state information. Note that each tire instance will need a separate file to keep the states.

With `ctiSetRunTimeMode`, certain settings can be chosen which influence the speed of computation. In many cases, the most restrictive setting will make the tire model real-time capable. This is achieved through switching off all non-standard model extensions, suppressing all extra output, and reducing geomatry as well as time resolution.

`ctiEvaluateRoadHeight` and `ctiEvaluateRoadCourse` make cosin/road's internal road evaluation routine available to the calling software. Typically, the tire model will be the only subsystem which really 'sees the road. It might be useful, however, that the same road can be evaluated outside the tire as well. As an example, the road height profile sometimes is to be displayed in an animation scene, or a driver model is to follow the road course.

Finally, `ctiClose` performs all necessary termination actions, like closing all files, closing the graphics window, etc. and has to be called after the simulation has finished.

By default, all log and error messages are put out to stdout `printf(..)`, and user input (if any) is read from stdin `scanf(..)`. Users can as well use their own messaging and logging routines by registering a message handler callback function (see ctiInit, typedef CTIINIT and typedef UMSGF).

A demo application source code ctiDemo.c is included with the distribution package.

For all platforms, when calling FTire, **CTI** provides on-line animation of the FTire structure and the contact patch force, friction, and temperature distribution. This animation is based upon OpenGL rendering.

3. cosin Tire Interface (CTI)

3.1. Tire handles

Tire handles are unique identifiers of the current tire. The tire handles are freely definable by the user (also negative values are allowed).

3.2. Program Structure of CTI Applications

Typical **CTI** application will be structured as follows:

1. Call `ctiInit` to initialize the interface.
2. Loop over all tire instances to be computed, loading tire and road data with routines `ctiLoadTireData` and `ctiLoadRoadData`.
3. Enter the time loop. In every time step, loop over all tire instances and evaluate tire force/torques by passing wheel position and velocity variables by calling `ctiComputeForces` or `ctiComputeForcesOnWheelCarrier`
4. Terminate all threads and **CTI** functions by calling `ctiClose`.

3.3. API Function Reference

3.3.1. ctiAdjustTwinTireWheelSpeed

Adjust twin or dual tires wheel speeds.

Prototype:

void ctiAdjustTwinTireWheelSpeed (int th1, int th2);

Parameters:

`in`	*th1*	tire handles left and right wheel
`in`	*th2*	tire handles left and right wheel

3.3.2. ctiAnimate

Set/unset animation mode.

Prototype:

void ctiAnimate (int th, int an);

Parameters:

`in`	*th*	tire handle

Document Revision:2018-4-r18799

in	*an*	animation flag • < 0 exit without any chages • 0 online animation for tire th off • 1 online animation for tire th on • 2 online animation for tire th as before • 3 online animation for tire th off, offline animation on • 4 online animation for tire th on, offline animation on • 5 online animation for tire th on, use new settings from an only if animation was off before

3.3.3. ctiAnimateOnly

Set/unset animation mode, only one tire.

Prototype:

void ctiAnimateOnly (int th, int an);

Parameters:

in	*th*	tire handle
in	*an*	animation flag • 0 animation for all tires off • 1 animation for tire th on, for all others off • 2 animation for tire th as before, for all others off

3.3.4. ctiAnimateScene

Animate complete scene show actual frame for all tires.

Prototype:

void ctiAnimateScene (void);

Parameters:

none

3.3.5. ctiAnimateSceneWithExtRoad

Animate complete scene show actual frame for all tires with external road function.

Prototype:

Document Revision:2018-4-r18799

void ctiAnimateSceneWithExtRoad (void* road);

Parameters:

in	*road*	name of road model subroutine used in EVRMR

3.3.6. ctiCheckLicense

Check license.

Prototype:

void ctiCheckLicense (int∗mode, char∗featur);

Parameters:

out	*mode*	• $<$ 9 licensed • 9 not licensed
in	*featur*	feature to check

3.3.7. ctiClose

Close interface.

Prototype:

void ctiClose (void);

Parameters:

none

3.3.8. ctiCloseTire

Close tire handle.

Prototype:

void ctiCloseTire (int th);

Parameters:

in	*th*	tire handle

3.3.9. ctiComputeForces

Main routine.

Prototype:

void ctiComputeForces (int th, double t, double r0r0[], double a0r[], double v0r0[], double w0r0[], int mode, double fr0[], double trr0[], int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time[s]
in	*r0r0*	rigid-body state of rim: position of rim center in global coordinates[m]
in	*a0r*	rigid-body state of rim: 3x3 orthogonal transformation matrix from rim-fixed frame to global coordinates. Vectors in rim-fixed frame are to be multiplied by a0r to result in the representation in global coordinates. Example: a0r∗ (0 1 0)' = 2nd column of a0r = direction vector of wheel spinning axis in global coordinates. Note: a0r is stored column-wise (like matrices are stored in FORTRAN, but unlike it is done in C), a0r = (a0r_11, a0r_21, a0r_31, a0r_12, ...)
in	*v0r0*	rigid-body state of rim: translational velocity of rim center in global coordinates: v0r0 = d/dt(r0r0)[m/s]
in	*w0r0*	rigid-body state of rim: angular velocity vector of rim relative to global coordinates, represented in global coordinates[rad/s]
in	*mode*	job control • .....0 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have not yet been accepted by external integrator • .....1 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have been accepted by external integrator • .....2 unconditionally (re-)calculate tire forces • .....3 calculate steady-state tire forces • .....4 calculate static tire forces, avgd. road • .....5 calculate static tire forces, enhanced accuracy, avgd. road • .....6 calculate static tire forces, time-/loc-dependent road • .....9 reset (prepare for next time loop w/o closing tire handle) • ....k. compute steady states first, if not yet done (only in dynamic case); repeat this in the first k>0 dynamic steps • .0.... standard cosimulation mode (calling solver waits for forces) • .3.... fast cosimuation mode (calling solver runs in parallel with CTI) • 0..... standard accuracy in steady-state computation • 1..... high accuracy in steady-state computation (required for linearization)
out	*fr0*	force acting on rim center (point of attack = wheel center) expressed in global coordinates[N]
out	*trr0*	torque acting on rim center, expressed in global coordinates[Nm]

out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

3.3.10. ctiComputeForcesOnCarBody

Alternative main routine, computing forces and moments on car-body, taking into account the suspension kinematics and dynamics.

This routine combines a dynamic MBS suspension model with an FTire model. It requires loading of two data files: one for the cosin/mbs suspension model (with `ctiLoadSuspensionData`), another one for FTire (with `ctiLoadTireData`). The routine will compute all forces and moments that are transferred from the tire model via the suspension model to the car-body.

Prototype:

void ctiComputeForcesOnCarBody (int th, double t, double r0b0[], double a0b[], double v0b0[], double w0b0[], double tdr, double tbr, int nin, double in[], int mode, double fv0[], double tvv0[], int nout, double out[], int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time
in	*r0b0*	rigid-body state of car-body, using cosin/mbs point of reference
in	*a0b*	rigid-body state of car-body, using cosin/mbs point of reference
in	*v0b0*	rigid-body state of car-body, using cosin/mbs point of reference
in	*w0b0*	rigid-body state of car-body, using cosin/mbs point of reference
in	*tdr*	drive torque as put out by the drive-train model. Only scalar component in direction of spindle. The calling program will have to take care that the reaction torque of tdr is applied to the appropriate part of the drive-train model[Nm]
in	*tbr*	brake torque as put out by the brake model. Only scalar component in direction of spindle. tbr is understood to be the maximum absolute brake torque which is in effect when the wheel is rolling. The tire model will compute and apply the effective brake torque. This will be negative when the wheel is rolling backward, and have smaller absolute value when the wheel rotation is locked. The calling program does not need to compute any reaction torque. In contrast to tdr, CTI treats tbr as an inner torque, acting between rim and wheel carrier[Nm]
in	*nin*	number of additional input signals
in	*in*	array of additional input signals, to be used in suspension model (signal names cti_ix, x=1,2,..)
in	*mode*	job control, cf. ctiComputeForces
out	*fv0*	force acting on car-body (point of attack = wheel center in desgin position), expressed in global coordinates[N]
out	*tvv0*	torque acting on car-body, expressed in global coordinates[Nm]
in	*nout*	number of additional output signals

out	*out*	array of additional output signals, to be provided by suspension model (signal names cti_ox, x=1,2,..)
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

3.3.11. ctiComputeForcesOnWheelCarrier

Alternative main routine, coupling the tire model to wheel carrier instead of rim. Rim rotation will be integrated by the tire model.

Prototype:

void ctiComputeForcesOnWheelCarrier (int th, double t, double r0c0[], double a0c[], double v0c0[], double w0c0[], double tdr, double tbr, int mode, double fc0[], double tcc0[], int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time
in	*r0c0*	rigid-body state of wheel carrier r0c0 = position of wheel carrier in global coordinates[m]
in	*a0c*	rigid-body state of wheel carrier a0c = 3x3 orthogonal transformation matrix from wheel-carrier- fixed frame to global coordinates. Vectors in wheel-carrier-fixed frame are to be multiplied by a0c to result in the representation in global coordinates. Example: a0c∗ (0 1 0)' = 2nd column of a0c = direction vector of wheel-spin axis in global coordinates. Note: a0c is stored column-wise (like matrices are stored in FORTRAN, but unlike it is done in C), a0c = (a0c_11, a0c_21, a0c_31, a0c_12, ...)
in	*v0c0*	rigid-body state of wheel carrier v0c0 = translational velocity of wheel carrier in global coordinates: v0c0 = d/dt(r0c0)[m/s]
in	*w0c0*	rigid-body state of wheel carrier w0c0 = angular velocity vector of wheel carrier relative to global coordinates, represented in global coordinates[rad/s]
in	*tdr*	drive torque as put out by the drive-train model. Only scalar component in direction of spindle. The calling program will have to take care that the reaction torque of tdr is applied to the appropriate part of the drive-train model[Nm]
in	*tbr*	brake torque as put out by the brake model. Only scalar component in direction of spindle. tbr is understood to be the maximum absolute brake torque which is in effect when the wheel is rolling. The tire model will compute and apply the effective brake torque. This will be negative when the wheel is rolling backward, and have smaller absolute value when the wheel rotation is locked. The calling program does not need to compute any reaction torque. In contrast to tdr, CTI treats tbr as an inner torque, acting between rim and wheel carrier[Nm]
in	*mode*	job control, cf. ctiComputeForces

out	*fc0*	force acting on wheel carrier (point of attack = wheel center), expressed in global coordinates[N]
out	*tcc0*	torque acting on wheel carrier, expressed in global coordinates[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

3.3.12. ctiComputeForcesPosition

Main routine, simplified version: ony rim position, no rim velocity input; transformation matrix replaced by toe, camber, and wheel roation angle.

Prototype:

void ctiComputeForcesPosition (int th, double t, double r0r0[], double camber, double toe, double wrot, int mode, double fr0[], double trr0[], int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time[s]
in	*r0r0*	rigid-body state of rim: position of rim center in global coordinates[m]
in	*camber*	wheel carrier camber angle (rotation angle about wheel-carrier-fixed x-axis)[deg]
in	*toe*	wheel carrier toe angle (rotation angle about global z-axis)[deg]
in	*wrot*	wheel rotation angle (wheel rotation angle about wheel spin axis)[deg]
in	*mode*	job control, cf. ctiComputeForces
out	*fr0*	force acting on rim center (point of attack = wheel center) expressed in global coordinates[N]
out	*trr0*	torque acting on rim center, expressed in global coordinates[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

3.3.13. ctiComputeForcesTimeContinuous

Alternative main routine, for calling time-continuous tire models, driven by STI interface.

Prototype:

void ctiComputeForcesTimeContinuous (int th, double t, double r0r0[], double a0r[], double v0r0[], double w0r0[], double s[], int mode, double fr0[], double trr0[], double sdot[], int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time[s]
in	*r0r0*	rigid-body state of rim
in	*a0r*	rigid-body state of rim
in	*v0r0*	rigid-body state of rim
in	*w0r0*	rigid-body state of rim
in,out	*s*	tire state array, to be integrated by calling solver . output in first step, input in all following steps
in	*mode*	job control, cf. ctiComputeForces
out	*fr0*	force acting on rim; point of attack = wheel center; expressed in global coordinates[N]
out	*trr0*	torque acting on rim, expressed in global coordinates[Nm]
out	*sdot*	array of tire state derivatives
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

3.3.14. ctiComputeForcesTimeContinuousWithExtRoad

Alternative main routine with external road function, for calling time-continuous tire models, driven by STI interface.

Prototype:

void ctiComputeForcesTimeContinuousWithExtRoad (int th, double t, double r0r0[], double a0r[], double v0r0[], double w0r0[], double s[], void* road, int mode, double fr0[], double trr0[], double sdot[], int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time[s]
in	*r0r0*	rigid-body state of rim
in	*a0r*	rigid-body state of rim
in	*v0r0*	rigid-body state of rim
in	*w0r0*	rigid-body state of rim
in,out	*s*	tire state array, to be integrated by calling solver . output in first step, input in all following steps
in	*road*	name of road model subroutine used in EVRMR
in	*mode*	job control, cf. ctiComputeForces
out	*fr0*	force acting on rim; point of attack = wheel center; expressed in global coordinates[N]
out	*trr0*	torque acting on rim, expressed in global coordinates[Nm]
out	*sdot*	array of tire state derivatives

Document Revision:2018-4-r18799

out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

3.3.15. ctiComputeForcesWithExtRoad

Main routine with external road function.

Prototype:

void ctiComputeForcesWithExtRoad (int th, double t, double r0r0[], double a0r[], double v0r0[], double w0r0[], void∗road, int mode, double fr0[], double trr0[], int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time
in	*r0r0*	rigid-body state of rim r0r0 = position of rim center in global coordinates[m]
in	*a0r*	rigid-body state of rim a0r = 3x3 orthogonal transformation matrix from rim-fixed frame to global coordinates. Vectors in rim-fixed frame are to be multiplied by a0r to result in the representation in global coordinates. Example: a0r∗ (0 1 0)' = 2nd column of a0r = direction vector of wheel-spin axis in global coordinates. Note: a0r is stored column-wise (like matrices are stored in FORTRAN, but unlike it is done in C), a0r = (a0r_11, a0r_21, a0r_31, a0r_12, ...)
in	*v0r0*	rigid-body state of rim v0r0 = translational velocity of rim center in global coordinates: v0r0 = d/dt(r0r0)[m/s]
in	*w0r0*	rigid-body state of rim w0r0 = angular velocity vector of rim relative to global coordinates, represented in global coordinates[rad/s]
in	*road*	name of road model subroutine used in EVRMR
in	*mode*	job control, cf. ctiComputeForces
out	*fr0*	force acting on rim center (point of attack = wheel center) expressed in global coordinates[N]
out	*trr0*	torque acting on rim center, expressed in global coordinates[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted • 999 no license

3.3.16. ctiEnableTimeContinuous

Enable time-continuous call of tire model.

Prototype:

void ctiEnableTimeContinuous (int th, int∗nsc);

Parameters:

in	*th*	tire handle
out	*nsc*	number of time-continuous state variables

3.3.17. ctiEvaluateRoadCourse

Evaluate road course.

Prototype:

void ctiEvaluateRoadCourse (int th, double tp, double∗x, double∗y, double∗z, double∗xp, double∗yp, double∗zp, double∗mu, int∗ier);

Parameters:

in	*th*	tire handle
in	*tp*	travel path for course evaluation[m]
out	*x*	course coordinates[m]
out	*y*	course coordinates[m]
out	*z*	course coordinates[m]
out	*xp*	partial derivatives of course coordinates with respect to travel path[-]
out	*yp*	partial derivatives of course coordinates with respect to travel path[-]
out	*zp*	partial derivatives of course coordinates with respect to travel path[-]
out	*mu*	friction factor[-]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Output should not be used

3.3.18. ctiEvaluateRoadHeight

Evaluate road height.

Prototype:

void ctiEvaluateRoadHeight (int th, double t, double x, double y, double∗z, double∗vx, double∗vy, double∗vz, double∗mu, int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	time for road evaluation[s]
in	*x*	locus for road evaluation in global coordinates[m]
in	*y*	locus for road evaluation in global coordinates[m]
out	*z*	road height[m]

out	*vx*	road velocity in global coordinates[m/s]
out	*vy*	road velocity in global coordinates[m/s]
out	*vz*	road velocity in global coordinates[m/s]
out	*mu*	friction factor[-]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Output should not be used

3.3.19. ctiFindOutputSignalNumber

Find output signal number, for use inctiGetOutputSignalNumber.

Prototype:

void ctiFindOutputSignalNumber (int th, int∗io, char∗co);

Parameters:

in	*th*	tire handle
out	*io*	number of first occurence of signal label in output signal list. 0 if label was not found
in	*co*	signal label to search for, might be abbreviated. First match in label list will be returned. SeeA.1 for a full list of supported labels.

3.3.20. ctiGetArraySize

Get maximum sizes of FTire arrays

Prototype:

void ctiGetArraySize (int th, int flag, int * arraySize);

Parameters:

in	*th*	tire handle
in	*flag*	kind of array 1: scalar output signals 2: all output signals 3: plot signals in mtl file 4: TYDEX signals 5: basic parameters 6: auxiliary parameters 7: FTire states 10: All CTI states (required buffer size for use in ctiWriteStateMemory and ctiReadStateMemory) 11: Subset of CTI states (required buffer size for use in ctiWriteStateMemory and ctiReadStateMemory)

Document Revision:2018-4-r18799

out	*arraySize*	size of output array

3.3.21. ctiGetContactBodyForces

Get contact body forces and moments.

Prototype:

void ctiGetContactBodyForces (int th, int cbh, double fs0[], double tss0[]);

Parameters:

in	*th*	tire handle
in	*cbh*	contact body handle
out	*fs0*	force acting on road body reference point, expressed in global coordinates[N]
out	*tss0*	torque acting on road body reference point, expressed in global coordinates[Nm]

3.3.22. ctiGetCosinSoftwareVersion

Get cosin software version.

Prototype:

void ctiGetCosinSoftwareVersion (int∗rev);

Parameters:

out	*rev*	cosin software revision number

3.3.23. ctiGetFileName

Get filenames for current tire handle.

Prototype:

void ctiGetFileName (int th, int kind, char* buffer, int bufferlen, int* ier);

Parameters:

in	*th*	tire handle

`in`	*kind*	kind of file • 1: tire data in • 2: tire data out • 3: road data • 4: rim data • 5: suspension data • 6: plot output • 7: contact forces • 8: geometry output • 9: belt states • 10: regular grid ground pressure • 11: contact patch boundaries • 12: tread depth • 13: RGR road update data • 14: record file
`out`	*buffer*	output buffer to store the current filename (0-terminated string)
`in`	*bufferlen*	length of output buffer
`out`	*ier*	• 0: ok • 2: invalid kind flag • 3: buffer size to small to store filename

3.3.24. ctiGetInstallationInfo

Get details of cosin installation.

Prototype:

void ctiGetInstallationInfo (int item, char *buffer, int bufferlen);

Parameters:

in	*item*	kind of info item requested: 0: cosin major version string 1: cosin software revision number 2: computer platform 3: cosin installation path 4: path to cosin binaries 5: path to cosin libraries
out	*buffer*	queried installation info item (0-terminated string)
in	*bufferlen*	length of buffer provided by calling function, to store installation info item

3.3.25. ctiGetLTIMatrix

Get linearized system matrices.

Prototype:

void ctiGetLTIMatrix (int th, int kind, int ns, double mat[], int∗ier);

Parameters:

in	*th*	tire handle
in	*kind*	kind of LTI matrix • 1 A-matrix • 2 B-matrix • 3 C-matrix • 4 D-matrix
in	*ns*	number of states as assumed by the calling program
out	*mat*	matrix in full and dense storage mode
out	*ier*	• 0 ok • 1 matrix can not be used

3.3.26. ctiGetNodePositions

Get surface node positions.

Prototype:

void ctiGetNodePositions (int th, double sx, double sy, double rn0[], double rnr[], double rnc[], int∗ier);

Parameters:

in	*th*	tire handle
in	*sx*	relative circumferential path coordinate, between 0 and 1

in	*sy*	relative lateral path co-ordinate, between 0 and 1 (0=midpoint of left rim flange circle, 1=midpoint of right rim flange circle), or between 10 and 11 (10=left rim flange, 11=right rim flange)
out	*rn0*	surface node position in global coordinates
out	*rnr*	surface node position in rim-fixed frame
out	*rnc*	surface node position in TYDEX C frame
out	*ier*	• 0 ok • 1 surface node position could not be calculated

3.3.27. ctiGetNodePositionsWithAttributes

Get surface node positions and additional surface-related attributes.

Prototype:

void ctiGetNodePositionsWithAttributes (int th, double sx, double sy, double rn0[], double rnr[], double rnc[], int natt, double att[], int∗ier);

Parameters:

in	*th*	tire handle
in	*sx*	relative circumferential path coordinate, between 0 and 1
in	*sy*	relative lateral path co-ordinate, between 0 and 1 (0=midpoint of left rim flange circle, 1=midpoint of right rim flange circle), or between 10 and 11 (10=left rim flange, 11=right rim flange)
out	*rn0*	surface node position in global coordinates
out	*rnr*	surface node position in rim-fixed frame
out	*rnc*	surface node position in TYDEX C frame
in	*natt*	size (maximum number of components) of array*att* as provided by calling program
out	*att*	additional interpolated surface attribute values (provided as many as available, but not more than*natt*): 1. normal deflections of tread blocks [m] 2. longitudinal (=tangential) deflections of tread blocks [m] 3. lateral deflections of tread blocks [m] 4. tread height [m] 5. surface temperature [degC] 6. stick(0)/slip(1) states of tread blocks

out	*ier*	• 0 ok • 1 surface node position could not be calculated

3.3.28. ctiGetNumberContinuousStates

Get number of time-continuous states (not supported in client/server environment).

Prototype:

void ctiGetNumberContinuousStates (int th, int∗nsc);

Parameters:

in	*th*	tire handle
out	*nsc*	number of time-continuous state variables

3.3.29. ctiGetOutputSignal

Get single output signal value.

Prototype:

void ctiGetOutputSignal (int th, double∗os, char∗label);

Parameters:

in	*th*	tire handle
out	*os*	value of output signal. (>= 1e60, if label not found in signal list)
in	*label*	label or label-substring of output signal. Notes: • „curvature“ would be a valid label-substring for„footprint center line curvature, 1/m“. The substring may occur both at the**beginning** or somewhere**inside**the label • if label-substring is not unique, the first matching label in the list of labels (which is**not**lexicographically sorted like the table below) will be used • output signals availability depends on output request and active sub-model • the position of the labels in the table below cannot be used as a signal number inctiGetOutputSignalNumber • since this routine repeatedly scans a long list of labels of available output signals, it might slow down program execution. A more efficient and recommended way of retrieving output signal values is the combination of the two API functionsctiFindOutputSignalNumber andctiGetOutputSignalNumber • plot signals under consideration here are only available if an additional output (plot) file is written, containing the requested signal. This is in contrast to signals ordered byctiFindOutputSignalNumber andctiGetOutputSignalNumber, which are unconditionally available

List of supported labels:

actual pressure, bar	air velocity variation, m/s
air-vibr-induced rim force x, N	air-vibr-induced rim force y, N
air-vibr-induced rim force z, N	aligning torque (C-axis), Nm
aligning torque (H-axis), Nm	aligning torque (ISO), Nm
aligning torque (W-axis), Nm	aligning torque FP (W-axis), Nm
ambient temperature, degC	belt extension, %
belt structure forces RTF, -	brake force, N
brake slip, %	camber angle, deg
contact processor RTF, -	cornering force, N
drag force, N	footprint area, m^2
footprint center line curvature, 1/m	footprint center line torsion, deg
footprint length, mm	footprint width, mm
fore-aft force (C-axis), N	fore-aft force (H-axis), N
fore-aft force (ISO), N	fore-aft force (W-axis), N
fore-aft force FP (W-axis), N	geom. long. footprint shift, mm
global friction coefficient, -	global integration step size, ms
ground pressure RMS, MPa	gyroscopic align. torque, Nm
gyroscopic overt. torque, Nm	implicit integration solver RTF, -
inflation pressure, bar	interior volume, m^3
lat. displacement, mm	lat. footprint shift, mm

lat. road surface displ., mm	lat. shift of Fz point of attack, mm
loaded radius, mm	local integration step size, ms
long. displacement, mm	long. footprint shift, mm
long. road surface displ., mm	long. shift of Fz point of attack, mm
mass variation near footprint, %	max. ground pressure, MPa
max. lat. elast. left rim displ., mm	max. lat. elast. right rim displ., mm
max. lat. plast. left rim def., mm	max. lat. plast. right rim def., mm
max. rad. elast. left rim displ., mm	max. rad. elast. right rim displ., mm
max. rad. plast. left rim def., mm	max. rad. plast. right rim def., mm
max. radial belt displ., mm	max. rim/belt intrusion, mm
max. rim/road intrusion, mm	max. side-wall/road intrusion, mm
max. sliding velocity, m/s	max. tread deflection, mm
mean circumferential air vel., m/s	mean contact patch temperature, degC
mean ground pressure, MPa	mean sliding velocity, m/s
mean tire torsion about spin axis, deg	mean tread temperature, degC
mean tread wear at y= 4.7mm, mm	mean tread wear at y= -4.7mm, mm
mean tread wear at y= 14.1mm, mm	mean tread wear at y= 23.5mm, mm
mean tread wear at y= 32.9mm, mm	mean tread wear at y= 42.3mm, mm
mean tread wear at y= 51.7mm, mm	mean tread wear at y= 61.1mm, mm
mean tread wear at y= 70.5mm, mm	mean tread wear at y= 80.0mm, mm
mean tread wear at y= 89.4mm, mm	mean tread wear at y= -14.1mm, mm
mean tread wear at y= -23.5mm, mm	mean tread wear at y= -32.9mm, mm
mean tread wear at y= -42.3mm, mm	mean tread wear at y= -51.7mm, mm
mean tread wear at y= -61.1mm, mm	mean tread wear at y= -70.5mm, mm
mean tread wear at y= -80.0mm, mm	mean tread wear at y= -89.4mm, mm
mean tread wear rate, mm/s	mean wear rate at y= 4.7mm, mm/s
mean wear rate at y= -4.7mm, mm/s	mean wear rate at y= 14.1mm, mm/s
mean wear rate at y= 23.5mm, mm/s	mean wear rate at y= 32.9mm, mm/s
mean wear rate at y= 42.3mm, mm/s	mean wear rate at y= 51.7mm, mm/s
mean wear rate at y= 61.1mm, mm/s	mean wear rate at y= 70.5mm, mm/s
mean wear rate at y= 80.0mm, mm/s	mean wear rate at y= 89.4mm, mm/s
mean wear rate at y= -14.1mm, mm/s	mean wear rate at y= -23.5mm, mm/s
mean wear rate at y= -32.9mm, mm/s	mean wear rate at y= -42.3mm, mm/s
mean wear rate at y= -51.7mm, mm/s	mean wear rate at y= -61.1mm, mm/s
mean wear rate at y= -70.5mm, mm/s	mean wear rate at y= -80.0mm, mm/s
mean wear rate at y= -89.4mm, mm/s	model level, -
normalized wheel rotation angle, deg	number of elements in road contact, -
on cleat, -	overturning torque (C-axis), Nm
overturning torque (H-axis), Nm	overturning torque (ISO), Nm
overturning torque (W-axis), Nm	overturning torque FP (W-axis), Nm
perc. of contacts in itv. v0/p0, %	perc. of contacts in itv. v0/p1, %
perc. of contacts in itv. v0/p2, %	perc. of contacts in itv. v0/p3, %
perc. of contacts in itv. v1/p0, %	perc. of contacts in itv. v1/p1, %
perc. of contacts in itv. v1/p2, %	perc. of contacts in itv. v1/p3, %

perc. of contacts in itv. v2/p0, %	perc. of contacts in itv. v2/p1, %
perc. of contacts in itv. v2/p2, %	perc. of contacts in itv. v2/p3, %
perc. of contacts in itv. v3/p0, %	perc. of contacts in itv. v3/p1, %
perc. of contacts in itv. v3/p2, %	perc. of contacts in itv. v3/p3, %
perc. of contacts in itv. v4/p0, %	perc. of contacts in itv. v4/p1, %
perc. of contacts in itv. v4/p2, %	perc. of contacts in itv. v4/p3, %
ply-steer moment, Nm	pneumatic scrub, mm
pneumatic trail, mm	power loss in plies, kW
power loss in tread, kW	power loss through damping, kW
power loss through road friction, kW	pressure variation, bar
probe block 1 lat. defl., mm	probe block 1 long. defl., mm
probe block 1 normal defl., mm	probe block 1 temp., degC
probe block 1 tread depth, mm	probe block 2 lat. defl., mm
probe block 2 long. defl., mm	probe block 2 normal defl., mm
probe block 2 temp., degC	probe block 2 tread depth, mm
probe block 3 lat. defl., mm	probe block 3 long. defl., mm
probe block 3 normal defl., mm	probe block 3 temp., degC
probe block 3 tread depth, mm	probe block 4 lat. defl., mm
probe block 4 long. defl., mm	probe block 4 normal defl., mm
probe block 4 temp., degC	probe block 4 tread depth, mm
probe block 5 lat. defl., mm	probe block 5 long. defl., mm
probe block 5 normal defl., mm	probe block 5 temp., degC
probe block 5 tread depth, mm	probe block 6 lat. defl., mm
probe block 6 long. defl., mm	probe block 6 normal defl., mm
probe block 6 temp., degC	probe block 6 tread depth, mm
probe block frict. coeff., -	probe block sliding vel., m/s
probe segment inclination, deg	probe segment lat. defl., mm
probe segment lateral curvature, 1/m	probe segment left rim force x, N
probe segment left rim force y, N	probe segment left rim force z, N
probe segment mean tread temp., degC	probe segment radial defl., mm
probe segment right rim force x, N	probe segment right rim force y, N
probe segment right rim force z, N	probe segment tang. defl., mm
rel. size of rim flange to belt contact area, %	rel. size of sliding area, %
rim camber angle, deg	rim rotation angle, deg
rim toe angle, deg	rim/road contact force x, N
rim/road contact force y, N	rim/road contact force z, N
road height at footprint center, m	road surface temperature, degC
rolling loss, N	rolling resistance coefficient, -
rolling torque (C-axis), Nm	rolling torque (H-axis), Nm
rolling torque (ISO), Nm	rolling torque (W-axis), Nm
rolling torque FP (W-axis), Nm	side force (C-axis), N
side force (H-axis), N	side force (ISO), N
side force (W-axis), N	side force FP (W-axis), N
sliding velocity RMS, m/s	slip angle, deg

steering angle, deg	time, s
tire deflection (C-axis), mm	tire deflection (H-axis), mm
tire deflection (W-axis), mm	tire deflection velocity, m/s
tire deflection, mm	tire mass, kg
tire structure temperature, degC	tire/rim slip, deg
toe angle, deg	total RTF, -
total power loss, kW	traveled distance, m
tread depth, mm	tread wear rate non-uniformity, mm/s
velocity (C-axis), m/s	velocity (W-axis), m/s
velocity, m/s	weight first ground pressure value, -
weight second ground pressure value, -	weight third ground pressure value, -
wheel angular acceleration, rad/s^2	wheel angular speed, rad/s
wheel lateral velocity, m/s	wheel load (C-axis), N
wheel load (H-axis), N	wheel load (ISO), N
wheel load (W-axis), N	wheel load FP (W-axis), N
wheel load w/o weight (C-axis), N	wheel longitudinal velocity, m/s
wheel rotation angle, deg	wheel slip, %
wheel vertical velocity, m/s	x angular velocity rim, rad/s
x-position footprint center, m	x-position rim center, m
x-position rim-center, m	x-position road ref., m
x-shift center of gravity, mm	x-velocity rim center, m/s
x-velocity rim-center, m/s	x-velocity road, m/s
y angular velocity rim, rad/s	y-position footprint center, m
y-position rim center, m	y-position rim-center, m
y-position road ref., m	y-shift center of gravity, mm
y-velocity rim center, m/s	y-velocity rim-center, m/s
y-velocity road, m/s	z angular velocity rim, rad/s
z-position rim center, m	z-position rim-center, m
z-position road ref., m	z-shift center of gravity, mm
z-velocity rim center, m/s	z-velocity rim-center, m/s
z-velocity road, m/s	

3.3.30. ctiGetOutputSignalNumber

Get single output signal.

Prototype:

void ctiGetOutputSignalNumber (int th, double*os, int ns);

Parameters:

`in`	*th*	tire handle
`out`	*os*	value of output signal. (>= 1e60, if label not found in signal list)
`in`	*ns*	signal number (obtained byctiFindOutputSignalNumber)

3.3.31. ctiGetOutputSignals

Provide output signal arrays for postprocessing in calling environment

Prototype:

void ctiGetOutputSignals (int th, int no, double * o);

Parameters:

in	*th*	tire handle
in	*no*	size of o array, as dimensioned in calling program
out	*o*	tire model-specific output signal array. For a list of output signals seeftire_model_pdf. At most, the first no components will be written to o.

3.3.32. ctiGetRimForces

Get linear distributed rim flange forces.

Prototype:

void ctiGetRimForces (int th, int n, double rl0[][3], double fl0[][3], double rr0[][3], double fr0[][3]);

Parameters:

in	*th*	tire handle
in	*n*	number of equally spaced probe nodes
out	*rl0*	position of left rim flange probe nodes, expressed in global coordinates[m]
out	*fl0*	linear distr. rim flange forces in left probe nodes[N]
out	*rr0*	position of right rim flange probe nodes, expressed in global coordinates[m]
out	*fr0*	linear distr. rim flange forces in right probe nodes[N]

3.3.33. ctiGetRimProperties

Get some important rim properties (properties eventually needed by calling program).

Prototype:

void ctiGetRimProperties (int th, double*rr, double*iryy, int*ier);

Parameters:

in	*th*	tire handle
out	*rr*	rim flange radius[m] (= rim well radius + rim flange height)
out	*iryy*	rim moment of inertia about spin axis, including fixed tire share[kgm2]
out	*ier*	• 0 ok • 1 tire data can not be used

Document Revision:2018-4-r18799

3.3.34. ctiGetRimRotationStates

Get rim rotation states.

Prototype:

void ctiGetRimRotationStates (int th, double∗a, double∗an, double∗w);

Parameters:

in	*th*	tire handle
out	*a*	absolute rim rotation angle[rad]
out	*an*	rim rotation angle, normalized to[0,2pi][rad]
out	*w*	rim angular velocity[rad/s][rad/s]

3.3.35. ctiGetRoadForces

Get road forces and moments.

Prototype:

void ctiGetRoadForces (int th, double fs0[], double tss0[]);

Parameters:

in	*th*	tire handle
out	*fs0*	force acting on road reference point expressed in global coordinates[N]
out	*tss0*	torque acting on road reference point, expressed in global coordinates[Nm]

3.3.36. ctiGetRoadParameters

Get TYDEX road parameters.

Prototype:

void ctiGetRoadParameters (int th, int pril[], double prdl[]);

Parameters:

in	*th*	tire handle
out	*pril*	integer road parameters
out	*prdl*	double road parameters

3.3.37. ctiGetRoadSize

Get road size.

Prototype:

void ctiGetRoadSize (int th, double rs[]);

Parameters:

in	*th*	tire handle

out	*rs*	road sizes array • rs[0] = smin[m] • rs[1] = smax[m] • rs[2] = dmin[m] • rs[3] = dmax[m] • rs[4] = ds[m] • rs[5] = dd[m] • rs[6] = ns[-] • rs[7] = nd[-] • rs[8] = length center-line[m]

3.3.38. ctiGetStatus

Get CTI interface status.

Prototype:

void ctiGetStatus (int∗st);

Parameters:

out	*st*	current CTI interface status 0: CTI not active (not yet initialized or closed for all instances) 1: CTI initializing, calling solver not yet set 2: CTI initialized, but calling solver not yet set 3: CTI initialized, calling solver set 4: CTI closing

3.3.39. ctiGetStepSize

Get last integration step size.

Prototype:

void ctiGetStepSize (int th, double∗h, double∗t);

Parameters:

in	*th*	tire handle
out	*h*	last integration step size[s]
out	*t*	last simulation time[s]

3.3.40. ctiGetTireDimensionData

Get tire dimension (and more) double data.

Prototype:

void ctiGetTireDimensionData (int th, int item, double*value);

Parameters:

in	*th*	tire handle
in	*item*	item to be provided: 1: section width [mm] 2: aspect ratio [%] 3: rim diameter [inch] 4: rim width [inch] 5: load index [LI] 6: numerically coded speed symbol [SSY] 7: tire load class [TLC] 8: reference inflation pressure [bar] 9: 2nd inflation pressure [bar] 10: actual inflation pressure [bar] 11: LI load [kN] 12: rated maximum velocity [m/s] 13: estimated maximum translational in-plane K&C compliance in look-up-table-based vehicle models [mm/kN] 14: estimated maximum rotational in-plane K&C compliance in look-up-table-based vehicle models [deg/Nm] 15: estimated maximum translational out-of-plane K&C compliance in look-up-table-based vehicle models [mm/kN] 16: estimated maximum rotational out-of-plane K&C compliance in look-up-table-based vehicle models [deg/Nm]
out	*value*	item value (0 if not defined or unknown)

3.3.41. ctiGetTireDimensionStringData

Get tire dimension string data.

Prototype:

void ctiGetTireDimensionStringData (int th, int item, char*value, int size);

Parameters:

in	*th*	tire handle
in	*item*	item to be provided: 1: manufacturer 2: brand 3: ETRTO size string
out	*value*	item value ('unknown' if not defined or unknown)
in	*size*	Size, in bytes, of the array referenced by 'value'

3.3.42. ctiGetTireHandle

Get tire handle from tire instance.

Prototype:

void ctiGetTireHandle (int ti, int∗th);

Parameters:

in	*ti*	tire instance
out	*th*	tire handle. -1, if ti is not yet used.

3.3.43. ctiGetTireInstance

Get tire instance from tire handle.

Prototype:

void ctiGetTireInstance (int th, int∗ti);

Parameters:

in	*th*	tire handle
out	*ti*	tire instance. -1, if too many tire instances are defined.

3.3.44. ctiGetTireKeyData

Get key tire data (data needed by calling program).

Prototype:

void ctiGetTireKeyData (int th, double∗rm, double∗rd, double∗rl, double∗mfr, double∗ifryy, double∗ifrzz, double∗mf, double∗ifyy, double∗ifzz, double∗cr, double∗cr2, int∗ier);

Parameters:

in	*th*	tire handle
out	*rm*	unloaded tire radius at zero camber angle and zero speed[m]
out	*rd*	dynamic rolling radius at zero speed and half LI load[m]
out	*rl*	loaded radius at zero speed and half LI load[m]
out	*mfr*	share of tire mass which is considered 'fixed to the rim', and which needs to be taken into account in the rigid body 'rim'[kg]
out	*ifryy*	share of tire moment of inertia about spin axis which is considered 'fixed to the rim', and which needs to be taken into account in the rigid body 'rim'[kgm2]
out	*ifrzz*	share of tire moment of inertia about verticall axis which is considered 'fixed to the rim', and which needs to be taken into account in the rigid body 'rim' (for symmetry reasons, IFRXX = ifrzz is assumed)[kgm2]
out	*mf*	share of tire mass which is considered 'free'[kg]
out	*ifyy*	share of tire moment of inertia about spin axis which is considered 'free'[kgm2]
out	*ifzz*	share of tire moment of inertia about verticall axis which is considered 'free' (for symmetry reasons, IFXX = ifzz is assumed)[kgm2]

Document Revision:2018-4-r18799

out	*cr*	coefficients of a quadratic approximation of the tire radial stiffness characteristic at zero camber angle, zero wheel speed, flat surface, and nominal inflation pressure: Fz = cr∗d + dr2∗d^2 - m∗g (d global tire deflection in , Fz static wheel load in N). cr is given in[N/m], cr2 in[N/m^2][m]
out	*cr2*	coefficients of a quadratic approximation of the tire radial stiffness characteristic at zero camber angle, zero wheel speed, flat surface, and nominal inflation pressure: Fz = cr∗d + dr2∗d^2 - m∗g (d global tire deflection in , Fz static wheel load in N). cr is given in[N/m], cr2 in[N/m^2][m]
out	*ier*	• 0 ok • 1 tire data can not be used

3.3.45. ctiGetTireModelType

Get tire model type.

Prototype:

void ctiGetTireModelType (int th, int∗tm);

Parameters:

in	*th*	tire handle

Document Revision:2018-4-r18799

out	*tm*	tire model type: • -3 recognized, but data file not loadable • -2 recognized, but respective model not licensed • -1 not (yet) recognized • 0 HTIRE / Magic Formula, Version M. • 4 HTIRE / Magic Formula, Version S. • 6 HTIRE / Magic Formula, Version 96 • 7 HTIRE / Magic Formula, Version H. • 8 HTIRE / Magic Formula, Version 89 • 9 HTIRE / Magic Formula, Version 94 • 10 HTIRE / Magic Formula, Version M./94 • 11 HTIRE / Magic Formula, Version BS./96 • 12 HTIRE / Magic Formula, Version 02 • 19 HTIRE / User-defined (uhm) • 20 FETire • 21 FTire • 98 User-defined, called via STI (sti) • 99 User-defined (utm)

3.3.46. ctiGetTireProperties

Get some important tire properties needed by the calling program (reduced version of ctiGetTireKeyData).

Prototype:

void ctiGetTireProperties (int th, double∗rm, double∗rd, double∗mr, double∗iryy, double∗irzz, double∗cr, double∗cr2, int∗ier);

Parameters:

in	*th*	tire handle
out	*rm*	unloaded tire radius at zero camber angle and zero speed[m]
out	*rd*	dynamic rolling radius at zero speed and half LI load[m]
out	*mr*	share of tire mass which is considered 'fixed to the rim', and which needs to be taken into account in the rigid body 'rim'[kg]
out	*iryy*	share of tire moment of inertia about spin axis which is considered 'fixed to the rim', and which needs to be taken into account in the rigid body 'rim'[kgm2]

out	*irzz*	share of tire moment of inertia about verticall axis which is considered 'fixed to the rim', and which needs to be taken into account in the rigid body 'rim' (for symmetry reasons, IRXX = irzz is assumed)[kgm2]
out	*cr*	coefficients of a quadratic approximation of the tire radial stiffness characteristic at zero camber angle, zero wheel speed, flat surface, and nominal inflation pressure: Fz = cr∗d + dr2∗d^2 - m∗g (d global tire deflection in , Fz static wheel load in N). cr is given in[N/m], cr2 in[N/m^2][m]
out	*cr2*	coefficients of a quadratic approximation of the tire radial stiffness characteristic at zero camber angle, zero wheel speed, flat surface, and nominal inflation pressure: Fz = cr∗d + dr2∗d^2 - m∗g (d global tire deflection in , Fz static wheel load in N). cr is given in[N/m], cr2 in[N/m^2][m]
out	*ier*	• 0 ok • 1 tire data can not be used

3.3.47. ctiGetTreadStates

Get tread states in footprint.

Prototype:

void ctiGetTreadStates (int th, double xt, double yt, int type, double∗ft, int∗ier);

Parameters:

in	*th*	tire handle
in	*xt*	co-ordinates in contact frame of tread element where state is to be put out[mm]
in	*yt*	co-ordinates in contact frame of tread element where state is to be put out[mm]
in	*type*	type of state to be put out • 1 ground pressure distribution • 2 long. shear stress distribution • 3 lat. shear stress distribution • 4 friction coeff. distribution • 5 sliding velocity • 6 temperature distribution • 7 tread depth • 8 power loss density in tread • 9 tread wear rate
out	*ft*	interpolated value of state

out	*ier*	• 0 ok • 1 state could not be interpolated

3.3.48. ctiGetTydexSignals

CTI Output.

Prototype:

void ctiGetTydexSignals (int th, int not, double∗ot);

Parameters:

in	*th*	tire handle
in	*not*	number of output signals • 0xxxx Using TYDEX-conform output signals. xxxx is size of ot array, as dimensioned in calling program. E.g. Use 20 to store the first 20 TYDEX-conform output signals in the ot array. • 1xxxx Using TYDEX-subset output signals. xxxx is size of ot array as dimensioned in calling program. E.g. Use 10020 to store the first 20 TYDEX-subset output signals signals in the ot array.
out	*ot*	TYDEX output signal array. First 25 signals coincide with the standardized STI VARINF output signals. For a list of TYDEX-conform output signals seeftire_model_pdf. For a list of TYDEX-subset output signals seeA.2. At most, the first not components will be written to out.

3.3.49. ctiInit

Initialize CTI. Initializes the CTI interface and sets calling application specific attributes:

- message line length
- 3rd-party license acceptance
- required license feature
- program options source
- 'kill solver on escape' property

depending on application and/or calling solver environment.

Repeated calling of ctiInit is not effective until next call of ctiClose.

Prototype:

void ctiInit (CTIINIT∗param);

Parameters:

in	*param*	parameter structure of typedef CTIINIT Note: Use CTIINIT_INITIALIZER to initalize this structure.

Example Code:

```
#include "cti.h"
void myMessageFunc (int level, char* message) {
    printf ("%s", message);
}
int foo1 (void) {
    /* default initialization */
    CTIINIT p = CTIINIT_INITIALIZER;
    ctiInit (&p);
}
int foo2 (void) {
    /* initialization with own message function */
    CTIINIT p = CTIINIT_INITIALIZER;
    p.message_func = myMessageFunc;
    ctiInit (&p);
}
int foo3 (void) {
    /* general initialization */
    CTIINIT p = CTIINIT_INITIALIZER;
    p.message_func = myMessageFunc;
    snprintf (p.output_folder, sizeof(p.output_folder), "/tmp/myOutputFolder");
    snprintf (p.output_prefix, sizeof(p.output_prefix), "myPrefix");
    p.calling_solver = 99;
    p.mt_call_flag = 1;
    ctiInit (&p);
}
```

3.3.50. ctiKillSolverOnEsc

Kill solver on Esc. Abort calling solver, if escape key in cosins animation window is hit.

Prototype:

void ctiKillSolverOnEsc (void);

Parameters:

none

3.3.51. ctiLinearize

Linearize system.

Prototype:

void ctiLinearize (int th, double t, double r0r0[], double a0r[], double v0r0[], double w0r0[], double*frmax, int*n, int job, double sli[], double slidot[], double fr0[], double trr0[], int*ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time
in	*r0r0*	rigid-body state of rim, used as operating point for linearization in first call, and as input to linearized system in subsequent calls
in	*a0r*	rigid-body state of rim, used as operating point for linearization in first call, and as input to linearized system in subsequent calls
in	*v0r0*	rigid-body state of rim, used as operating point for linearization in first call, and as input to linearized system in subsequent calls
in	*w0r0*	rigid-body state of rim, used as operating point for linearization in first call, and as input to linearized system in subsequent calls
in,out	*frmax*	on input: maximum natural frequency taken into account for linearized system. Not used if less or equal to zero on output: actual maximum natural frequency of linearized system[Hz]
in,out	*n*	on input: desired size (number of state variables) of linearized system. Not used if less or equal to zero. Arrays sli and slidot must be dimensioned in calling program at least with size n on output: actual size of linearized system
in	*job*	job control • ...0 compute derivatives • ...1 compute output signals • ...2 compute both • ..0. refresh linearization only if system size has changed • ..1. unconditionally refresh linearization • .0.. angle input: Bryant angles • .1.. angle input: finite rotation about global x/y/z axis • 0... take damping into account • 1... neglect damping
in	*sli*	linearized state vector (to be integrated outside CTI)
out	*slidot*	time derivative of linearized state vector (to be integrated outside CTI)
out	*fr0*	linearized force acting on rim center (point of attack = wheel center) expressed in global coordinates[N]
out	*trr0*	linearized torque acting on rim center, expressed in global coordinates[Nm]

out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Linearized system should not be used

3.3.52. ctiLinearizeWheelCarrier

Alternative linearize system.

Prototype:

void ctiLinearizeWheelCarrier (int th, double t, double r0c0[], double a0c[], double v0c0[], double w0c0[], double∗frmax, int∗n, int job, double sli[], double slidot[], double fc0[], double tcc0[], int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time
in	*r0c0*	rigid-body state of wheel carrier, used as operating point for linearization in first call, and as input to linearized system in subsequent calls
in	*a0c*	rigid-body state of wheel carrier, used as operating point for linearization in first call, and as input to linearized system in subsequent calls
in	*v0c0*	rigid-body state of wheel carrier, used as operating point for linearization in first call, and as input to linearized system in subsequent calls
in	*w0c0*	rigid-body state of wheel carrier, used as operating point for linearization in first call, and as input to linearized system in subsequent calls
in,out	*frmax*	on input: maximum natural frequency taken into account for linearized system. Not used if less or equal to zero on output: actual maximum natural frequency of linearized system[Hz]
in,out	*n*	on input: desired size (number of state variables) of linearized system. Not used if less or equal to zero. Arrays sli and slidot must be dimensioned in calling program at least with size n on output: actual size of linearized system

in	*job*	job control • ...0 compute derivatives • ...1 compute output signals • ...2 compute both • ..0. refresh linearization only if system size has changed • ..1. unconditionally refresh linearization • .0.. angle input: Bryant angles • .1.. angle input: finite rotation about global x/y/z axis • 0... take damping into account • 1... neglect damping
in	*sli*	linearized state vector (to be integrated outside CTI)
out	*slidot*	time derivative of linearized state vector (to be integrated outside CTI)
out	*fc0*	linearized force acting on wheel carrier (point of attack = wheel center) expressed in global coordinates[N]
out	*tcc0*	linearized torque acting on wheel carrier, expressed in global coordinates[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Linearized system should not be used

3.3.53. ctiLoadRimData

Select and load rim data file.

Prototype:

void ctiLoadRimData (int th, int∗ier, char∗file);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading data file (error message was written to log). Simulation should be aborted • 2 no license available
in	*file*	rim data filename

3.3.54. ctiLoadRimModel

Select and load rim model library.

Prototype:

void ctiLoadRimModel (int th, int∗ier, char∗riml, char∗rimf);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading rim library (error message was written to log). Simulation should be aborted
in	*riml*	name of dynamic library, observing resp. OS conventions
in	*rimf*	name of urim.c-compatible module in dynamic library, observing resp. OS conventions

3.3.55. ctiLoadRoadData

Select and load road data file.

Prototype:

void ctiLoadRoadData (int th, int∗ier, char∗file);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading data file (error message was written to log). Simulation should be aborted • 2 no license available
in	*file*	road data filename. Use function-pointer-called user-defined road model, if file name is preceded by 'urm:'

Note:

If called for road evaluations (e.g. with ctiEvaluateRoadHeight) not related to a specific tire, please use a differnt tire handle to prevent problems with internal states of the tire handle.

3.3.56. ctiLoadRoadModel

Select and load road model library.

Prototype:

void ctiLoadRoadModel (int th, int∗ier, char∗roadl, char∗roadf);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading road library (error message was written to log). Simulation should be aborted
in	*roadl*	name of dynamic library, observing resp. OS conventions
in	*roadf*	name of urm.c-compatible module in dynamic library, observing resp. OS conventions

3.3.57. ctiLoadSTIRoadModel

Select and load STI road model library.

Prototype:

void ctiLoadSTIRoadModel (int th, int∗ier, char∗stil, char∗stif);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading STI library (error message was written to log). Simulation should be aborted
in	*stil*	undecorated name of dynamic library , containing STT-compatible road model with calling syntax: void stif (TIME,DIS, IFLAG,IDROAD, NROPAR,ROPAR, NCHRDS,CHRDST, NPRSUR,NROAD, Z,DZ,DDZ, DFLAG,PRSURF, IERR)
in	*stif*	undecorated name of road function in stil

3.3.58. ctiLoadSTITireModel

Select and load STI tire model library.

Prototype:

void ctiLoadSTITireModel (int th, int∗ns, int∗ier, char∗stil, char∗stif);

Parameters:

in	*th*	tire handle
out	*ns*	number of time-continuous state variables of STI-compatible tire model, to be integrated by calling solver
out	*ier*	error flag • 0 ok • 1 error occurred when loading STI libraries (error message was written to log). Simulation should be aborted
in	*stil*	undecorated name of dynamic library , containing STT-compatible tire model function with calling syntax: void stif (NDEV, ISWTCH, JOBFLG, IDTYRE, TIME, DIS, TRAMAT, ANGTWC, VEL, OMEGA, OMEGAR, NDEQVR, DEQVAR, NTYPAR, TYPARR, NCHTDS, CHTDST, ROAD, IDROAD, NROPAR, ROPAR, NCHRDS, CHRDST, FORCE, TORQUE, DEQINI, DEQDER, TYRMOD, NVARS, VARINF, NWORK, WRKARR, NIWORK, IWRKAR, IERR)
in	*stif*	undecorated name of tire function in stil

3.3.59. ctiLoadSoilModel

Select and load soil model library.

Prototype:

void ctiLoadSoilModel (int th, int∗ier, char∗soill, char∗soilf);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading soil library (error message was written to log). Simulation should be aborted
in	*soill*	name of dynamic library, observing resp. OS conventions
in	*soilf*	name of usm.c-compatible module in dynamic library, observing resp. OS conventions

3.3.60. ctiLoadSuspensionData

Select and load suspension data file.

This routine will load and pre-process a*cosin/mbs* suspension model for use within `ctiComputeForcesOnCarBody` (3.3.10). Suspension model dctiLoadSTIRoadModelata files should have file extension .cm. After installation ofcosin software, you will find the following example files in subdirectoryftire/param of your working directory:

- _default.cm (a steered double wishbone front susopension)
- double_wishbone_front.cm

- double_wishbone_rear.cm
- five_link_front.cm
- five_link_rear.cm
- mcpherson_front.cm
- mcpherson_rear.cm

All data files are parameterized with key data, and thus can serve as a starting point for user-defined suspension models. For more on the format and contents of these files, please refer to the*cosin/mbs* documentation.

Prototype:

void ctiLoadSuspensionData (int th, int*ier, char*file);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading data file (error message was written to log). Simulation should be aborted • 2 no license available
in	*file*	tire data-file name

3.3.61. ctiLoadTireData

Select and load tire data file.

Prototype:

void ctiLoadTireData (int th, int*ier, char*file);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading data file (error message was written to log). Simulation should be aborted • 2 no license available
in	*file*	tire data-file name. If data have to be mirrored, place prefix mirror: in front. That is, file-name mirror: c:\data\mydata.tir will mirror data in file c:\data\mydata.tir

3.3.62. ctiModifyFriction

Modify friction characteristic.

Prototype:

void ctiModifyFriction (int th, double muf);

Parameters:

in	*th*	tire handle
in	*muf*	friction characteristic modification factor[-]. Default value = 1.0

3.3.63. ctiOpenOutputFile

Open additional plot output file.

Prototype:

void ctiOpenOutputFile (int th, int∗ier, char∗file);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log)
in	*file*	output file name

3.3.64. ctiOpenRoadGui

Open road GUI with current road file

Prototype:

void ctiOpenRoadGui (int th);

Parameters:

in	*th*	tire handle

3.3.65. ctiOpenTireGui

Open tire GUI with current tire file

Prototype:

void ctiOpenTireGui (int th);

Parameters:

in	*th*	tire handle

3.3.66. ctiReadLTIMatrices

Read linearized A,B system matrices from file. Only needed for ADAMS GSE interface.

Prototype:

void ctiReadLTIMatrices (int th, int∗ier);

Parameters:

`in`	*th*	tire handle
`out`	*ier*	• 0 ok • 1 no success

3.3.67. ctiReadOperatingConditions

Read time- or location-dependent operating conditions from TeimOrbit file.

Prototype:

void ctiReadOperatingConditions (int th, double t, double r0r0[], int∗ier);

Parameters:

`in`	*th*	tire handle
`in`	*t*	simulation time
`in`	*r0r0*	rigid-body position of rim

out	*ier*	error flag • 0 ok • .....xx (xx>0) error opening or reading data file • .....1.. first operating condition type not supported • .....2.. first operating condition not specified • .....1... second operating condition type not supported • .....2... second operating condition not specified • ...1..... third operating condition type not supported • ...2..... third operating condition not specified • ..1..... fourth operating condition type not supported • ..2..... fourth operating condition not specified • .1..... fifth operating condition type not supported • .2..... fifth operating condition not specified • 1..... sixth operating condition type not supported • 2..... sixth operating condition not specified

3.3.68. ctiReadStates

State array in (read from file).

Prototype:

void ctiReadStates (int th, int∗ier, char∗file);

Parameters:

in	*th*	tire handle
out	*ier*	error code • 0 ok • 1 file containing state array not found • 2 file containing state array not valid • 3 state array size unknown since no tire data file loaded • 4 file containing state array created by different cosin version • 5 unspecified error reading state file (corrupted file?)
in	*file*	file to read state array from. note that each tire instance needs a separate file.

3.3.69. ctiReadStatesMemory

State array in (read from memory).

Prototype:

void ctiReadStatesMemory (int th, int mode, int memsize, double∗mem, int∗ier);

Parameters:

in	*th*	tire handle
in	*mode*	mode flag • 0 write all states and output values • 1 write all states
in	*memsize*	size of memory block 'mem'. Use ctiGetArraySize(flag = 10 + mode) to obtain the CTI state array size for declaration or allocation of the memory block 'mem'.
in	*mem*	memory block containing the states
out	*ier*	error code • 0 ok • 1 provided array size larger than CTI state array size. Incompatibility likely • 2 provided array size too small. Incompatible

3.3.70. ctiRecorder

Set/unset record output flag.

Prototype:

void ctiRecorder (int th, int rec);

Parameters:

in	*th*	tire handle
in	*rec*	record flag • 0 stop record output for this tire instance • 1 start record output for this tire instance, use common record file (does not work with multi-threaded call), if speed mode == 0 and not in diagnosis mode • 2 start record output for this tire instance, create and use individual record file, if speed mode == 0 and not in diagnosis mode

3.3.71. ctiReset

Reset CTI.

Prototype:

void ctiReset (void);

Parameters:

none

3.3.72. ctiSaveRecordedForcesMoments

Save recorded forces and torques.

Prototype:

void ctiSaveRecordedForcesMoments (int th, double f[], double t[]);

Parameters:

in	*th*	tire handle
in	*f*	recorded forces[N]
in	*t*	recorded torques[Nm]

3.3.73. ctiSetAffinity

Set tire (thread) affinity.

Prototype:

void ctiSetAffinity (int th, int mode, int cpu_id);

Parameters:

in	*th*	tire handle
in	*mode*	set mode • 0: set tire thread affinity to cpu_id • 1: set tire thread affinity to ti_0 modulo N where ti_0 is the belonging tire instance (0-based) and N is the number of processors Note: cpu_id is not used in this case
in	*cpu_id*	cpu id (0-based)

3.3.74. ctiSetAmbientTemperature

Set tire (or ambient) temperature.

Prototype:

void ctiSetAmbientTemperature (int th, double ttemp);

Parameters:

in	*th*	tire handle
in	*ttemp*	tire (or ambient) temperature[degC]

3.3.75. ctiSetAnimationStepSize

Set animation step-size h.

Prototype:

void ctiSetAnimationStepSize (int th, double h);

Parameters:

in	*th*	tire handle
in	*h*	animation step-size[ms]

3.3.76. ctiSetCompatVersion

Set compatibility version.

Prototype:

void ctiSetCompatVersion (int th, int mode, int value);

Parameters:

in	*th*	tire handle
in	*mode*	mode flag • 0: value is a version, format = YYYYQ, default is 0 (latest version) • 1: value is a date, format = YYYYDDMM, default is 0 (latest date)
in	*value*	compatibility version (mode = 0) OR compatibility date (mode = 1)

Note: this function is only effective if called before ctiLoadTireData.

The compatibility date or version can be influenced in several ways:

1. with the environmental variable COSIN_OPTIONS
2. by setting the command-line option -cosin_compvers
3. by selecting a 'mimicking library version' in cosin's GUI
4. by specifying a compatibility date or version in the tire data file, using cosin/tools for tires
5. by calling ctiSetCompatVersion

The compatibility date or version actually in effect will be the earliest of all dates specified in any of the ways listed above. CTI will write a message stating the date in effect and the source of this date.

Only the newest cosin software version lets you take advantage of all the latest model enhancements and bug fixes. So please have in mind that cosin does **not** recommend making use of the compatibility date. Only reason to do so anyway is being urged to exactly reproduce results obtained with an older cosin version.

3.3.77. ctiSetContactBodyMotionData

Set contact body motion data.

Prototype:

void ctiSetContactBodyMotionData (int th, int cbh, double r0s0[], double a0s[], double v0s0[], double w0s0[]);

Parameters:

in	*th*	tire handle
in	*cbh*	contact body handle, refers to body handle in accompanying triangulation-based road data file
in	*r0s0*	rigid-body states of contact body: displacement of reference point in global coordinates
in	*a0s*	rigid-body states of contact body: transformation matrix from body-fixed coordinates to global coordinates, cf. ctiComputeForces
in	*v0s0*	rigid-body states of contact body: displacement velocity vector in global coordinates
in	*w0s0*	rigid-body states of contact body: angular velocity vector in global coordinates

3.3.78. ctiSetDesignParameter

Set design parameter.

Prototype:

void ctiSetDesignParameter (int th, int kp, double vp);

Parameters:

in	*th*	tire handle
in	*kp*	index of parameter to be set
in	*vp*	parameter value to be set

3.3.79. ctiSetDiagMode

Set design parameter.

Prototype:

void ctiSetDiagMode (int th, int diag);

Parameters:

in	*th*	tire handle

in	*diag*	set diagnosis level • 0 diagnosis for tire hande off • 1 diagnosis for tire hande, level 1: force mtl output • 2.. diagnosis for tire hande, level 2: force mtl output, force anim.

3.3.80. ctiSetDrumTorque

Set the drive/brake torque acting on a drum testrig. This torque, which is positive for driving and negative for braking, does *not* contain the reaction torques due to tire longitudinal forces, but only the external drive/brake torque of the testrig. If several tires are running on the same drum, make sure that this torque is set (by calling `ctiSetDrumTorque`) for only *one* of these tires. Tire reaction torques will be accumulated automatically by CTI for all wheels runing on the same drum. Drums will be considered the same if their respective x/y location (as optionally set in the 2D road model type 'drum', see *cosin/roads* docu) is the same. Resulting drum angular speed will be computed and set by CTI on basis of the tire reaction torques, the applied torque as set by `ctiSetDrumTorque`, and the drum's moment of inertia as set in the road data file.

Prototype:

void ctiSeDrumTorque (int th, double tdrum);

Parameters:

in	*th*	tire handle
in	*tdrum*	drum drive/brake torque[Nm]

3.3.81. ctiSetInflationPressure

Set the current 'cold' inflation pressure.

Prototype:

void ctiSetInflationPressure (int th, double press);

Parameters:

in	*th*	tire handle
in	*press*	inflation pressure[bar]

3.3.82. ctiSetInitialRimAngle

Set initial rim rotation angle (effective only together with the alternative interface). This function is to be called prior to the first call to ctiComputeForcesOnWheelCarrierxxx() for the respective tire instance.

Prototype:

void ctiSetInitialRimAngle (int th, double arim0);

Parameters:

in	*th*	tire handle
in	*arim0*	initial rim rotation angle[deg]

3.3.83. ctiSetInitialTemperature

Set initial tire temperatures of filling gas and mean tread surface temperature. This function is to be called prior to the first call to ctiComputeForcesxxx() for the respective tire instance, and is effective only if the thermal model is activated.

Prototype:

void ctiSetInitialTemperature (int th, double ttemp0);

Parameters:

in	*th*	tire handle
in	*ttemp0*	initial tire temperature[degC]

3.3.84. ctiSetInitialTireTemperatures

Set initial tire temperatures of filling gas and mean tread surface temperature. This function is to be called prior to the first call to ctiComputeForcesxxx() for the respective tire instance, and is effective only if the thermal model is activated.

Prototype:

void ctiSetInitialTireTemperatures (int th, double tg0, double tt0);

Parameters:

in	*th*	tire handle
in	*tg0*	initial filling gas temperature[degC]
in	*tt0*	initial mean tread surface temperature[degC]

3.3.85. ctiSetIntegerRoadParameter

Set single integer road parameter.

Prototype:

void ctiSetIntegerRoadParameter (int th, int irp, int virp);

Parameters:

in	*th*	tire handle
in	*irp*	index of parameter to be set
in	*virp*	parameter value to be set

3.3.86. ctiSetMultiThreadedCallFlag

Set multi-threaded call flag.

Prototype:

void ctiSetMultiThreadedCallFlag (void);

Parameters:

none

3.3.87. ctiSetNotify

Register a notify callback function.

Prototype:

void ctiSetNotify (int th, int type, void*handle, void*retbuf, int*ier);

Parameters:

in	*th*	tire handle
in	*type*	notify type • 0 notify if worker thread created • 1 notify if time step started • 2 notify if time step stopped • 3 notify if time step started and return current time stamp • 4 notify if time step stopped and return current time stamp
in	*handle*	notify function pointer oftypedef CTINOTIFY
in	*retbuf*	notify return buffer, passed to handle. Ensure the buffer matches with the notify type. Buffer typedefs are defined in ctinotify.h header file.
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

Example Code:

```
#include "cti.h"
#include "ctinotify.h"

int handle (void* retval) {
    ctinotify0_t* r = (ctinotify0_t*) retval;
    if (r) {
        printf("Worker thread created (tid=%p, th=%i)\n", r->tid, r->th);
    }
    return(0);
```

```
}
ctinotify0_t retval; /* working array for notify callback */

ctiSetNotify (th, 0, &handle, &retval, &ier);
```

3.3.88. ctiSetOption

Set program option for use within other CTI or FTire function

Prototype:

void ctiSetOption (char* option, char* value);

Parameters:

in	*option*	option name (a list of recognized option names will be added as soon as available)
in	*value*	option value

3.3.89. ctiSetOutputFilePrefix

Set folder and prefix of output files.

Prototype:

void ctiSetOutputFilePrefx (char* folder, char* prefix);

Parameters:

in	*folder*	folder to save output files to (using respective operating system's naming conventions)
in	*prefix*	prefix of output file names. Suffixes will be set automatically, using tire index. File extensions indicate data types

3.3.90. ctiSetOutputStepSize

Set output step-size h.

Prototype:

void ctiSetOutputStepSize (int th, double h);

Parameters:

in	*th*	tire handle
in	*h*	step-size for output of additional data in files[ms]

3.3.91. ctiSetPPTireDataFilename

Set pre-processed tire-data filename.

Prototype:

void ctiSetPPTireDataFilename (int th, char∗filepp);

Document Revision:2018-4-r18799

Parameters:

in	*th*	tire handle
in	*filepp*	pre-processed tire-data filename

3.3.92. ctiSetPrmHandle

Set PRM handle.

Prototype:

void ctiSetPrmHandle (void∗prmHandle);

Parameters:

in	*prmHandle*	external PRM handle

3.3.93. ctiSetRoadEvalPreference

Set road evaluation preference.

Prototype:

void ctiSetRoadEvalPreference (int th, int pref);

Parameters:

in	*th*	tire handle
in	*pref*	road evaluation preference indicator • 0 use default evaluation method • 1 if available, prefer cosin evaluation method • 2 if available, prefer 3rd-party evaluation method

3.3.94. ctiSetRoadMotionData

Set road motion data.

Prototype:

void ctiSetRoadMotionData (int th, double r0s0[], double a0s[], double v0s0[], double w0s0[]);

Parameters:

in	*th*	tire handle
in	*r0s0*	rigid-body state of road-supporting body
in	*a0s*	rigid-body state of road-supporting body
in	*v0s0*	rigid-body state of road-supporting body
in	*w0s0*	rigid-body state of road-supporting body

3.3.95. ctiSetRoadParameters

Set TYDEX road parameters.

Prototype:

void ctiSetRoadParameters (int th, int pril[], double prdl[]);

Parameters:

in	*th*	tire handle
in	*pril*	integer road parameters
in	*prdl*	double road parameters. Do not use if prdl(1)=NAN

3.3.96. ctiSetRoadTemperature

Set road surface temperature.

Prototype:

void ctiSetRoadTemperature (int th, double troad);

Parameters:

in	*th*	tire handle
in	*troad*	road surface temperature[degC]

3.3.97. ctiSetRunTimeMode

Set or limit run-time mode and enable/disable step-size control of calling solver. Note: the 4 levels of 'accelerated execution mode' and the 5 levels of 'real-time mode' are composed of specific values of the run-time mode set here, and of additional specifications of numerical settings like force extrapolation and so on. These numerical settings are specified either in one of the `ctiComputeForces..()` calls, or directly in the tire data file (see *cosin/tools for tires* for more). If an accelerated execution level or real-time level is set in the tir-file, `ctiSetRunTimeMode()` doesn't need to be called.

Prototype:

void ctiSetRunTimeMode (int rm);

Parameters:

in	*rm*	number with up to two decimal digits with following meaning: <0 exit without any changes -1 do not accept step-size control of calling solver (default for run-time mode modes 4 and 5) -2 accept step-size control of calling solver (default for run-time modes 0 to 3) .0 set standard run-time mode .1 set run-time mode 1: no model extensions .2 set run-time mode 2: no model extensions, no output .3 set run-time mode 3: no model extensions, no output, coarse mesh .4 set run-time mode 4: no model extensions, no output, coarse mesh, reduced extra signal output .5 set run-time mode 5: no model extensions, no output, coarse mesh, minimum extra signal output 0. **unconditionally** set run-time mode to value as specified in last digit 1. **downward** restrict run-time mode by value as specified in last digit. Effective only if called after tire data files have been loaded 2. **upward** restrict run-time mode by value as specified in last digit. Effective only if called after tire data files have been loaded

3.3.98. ctiSetServer

Specify remote server.

Prototype:

void ctiSetServer (int portl, int∗ier, char∗host);

Parameters:

in	*portl*	TCP port used for connection. Default port used if negative
out	*ier*	Error return code from CTICLI
in	*host*	server host name. No remote server used if blank

3.3.99. ctiSetStatesMemory

Specify state array out (write to memory). Writing is triggered by job flag in ctiComputeForcesList.

Prototype:

void ctiSetStatesMemory (int th, int mode, int memsize, double∗mem, int∗ier);

Parameters:

in	*th*	tire handle
in	*mode*	mode flag • 0 write all states and output values • 1 write all states
in	*memsize*	size of memory block 'mem'. Use ctiGetArraySize(flag = 10 + mode) to obtain the CTI state array size for declaration or allocation of the memory block 'mem'.
out	*mem*	memory block to store the states
out	*ier*	error code • 0 ok • 1 provided array size too small. Incompatible

3.3.100. ctiSetTimeConstantForces

Set time constant of force low-pass filter.

Prototype:

void ctiSetTimeConstantForces (int th, double tconst);

Parameters:

in	*th*	tire handle
in	*tconst*	time constant, may be zero or negative[s]

3.3.101. ctiSetTireSide

Set tire side. ctiSetTireSide must be called prior to loading tire data with ctiLoadTireData!

Prototype:

void ctiSetTireSide (int th, int side);

Parameters:

in	*th*	tire handle
in	*side*	mounted tire side • 0 automatic • 1 left • 2 right • otherwise: side unchanged

3.3.102. ctiSetTreadDepth

Set tread depth.

Prototype:

void ctiSetTreadDepth (int th, double tdepth);

Parameters:

in	*th*	tire handle
in	*tdepth*	tread depth[mm]

3.3.103. ctiSetURIM

Register user-defined road model function of typedef URIM as callback function.

Prototype:

void ctiSetURIM (int th, URIM func);

Parameters:

in	*th*	tire handle
in	*func*	user-provided callback function.

3.3.104. ctiSetURM

Register user-defined road model function of typedef URM as callback function. FTire will call this function to evaluate road height instead of its internal road evaluation function. For each individual tire handle, ctiSetURM-FuncHandle may define an individual callback function, and is used mutually exclusively to ctiLoadRoadData.

Prototype:

void ctiSetURM (int th, int∗ier, URM func, char∗road_file);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 error occurred when loading data file of user-defined road model. Simulation should be aborted
in	*func*	user-provided road model callback function.
in	*road_file*	road data filename of user-defined road model. This file name is passed over to the URM function.

3.3.105. ctiSetUSM

Register user-defined road model function of typedef USM as callback function.

Prototype:

void ctiSetUSM (int th, USM func);

Parameters:

in	*th*	tire handle
in	*func*	user-provided callback function.

3.3.106. ctiSetVehicleStates

Set vehicle states (for use in animation and certain 2D roads).

Prototype:

void ctiSetVehicleStates (double tv, double r0v0[], double a0v[], double v0v0[], double w0v0[]);

Parameters:

in	*tv*	current simulation time
in	*r0v0*	rigid-body state of vehicle body
in	*a0v*	rigid-body state of vehicle body
in	*v0v0*	rigid-body state of vehicle body
in	*w0v0*	rigid-body state of vehicle body

3.3.107. ctiSetWheelCenterRefPosition

Set wheel center reference position.

Prototype:

void ctiSetWheelCenterRefPosition (int th, double r0r0l[], double a0rl[]);

Parameters:

in	*th*	tire handle
in	*r0r0l*	wheel center reference position in global coordinates
in	*a0rl*	wheel reference transformation matrix

3.3.108. ctiUpdateRoadData

Update RGR road data file no action, if no RGR road or no update file available.

Prototype:

void ctiUpdateRoadData (int th, double t, int∗ier, char∗file);

Parameters:

in	*th*	tire handle
in	*t*	simulation time

out	*ier*	error flag • 0 ok • 1 error occurred when loading data file (error message was written to log). Simulation should be aborted • 2 no license available
in	*file*	filename with RGR update data, as created by evrgru

3.3.109. ctiUpdateWheelEnvelope

Update wheel envelope.

Prototype:

void ctiUpdateWheelEnvelope (int th);

Parameters:

in	*th*	tire handle

3.3.110. ctiVerbose

Set/unset verbosity.

Prototype:

void ctiVerbose (int th, int v);

Parameters:

in	*th*	tire handle
in	*v*	verbosity flag • < 0 exit without any chages • 0 verbosity for tire th off • 1 verbosity for tire th on

3.3.111. ctiWriteAdditionalOutput

Set/unset additional plot output in mtl (ascii) or mtb (binary) file

Prototype:

void ctiWriteAdditionalOutput (int th, int ao);

Parameters:

in	*th*	tire handle

in	*ao*	flag to request / set kind of additional plot output in mtl (ascii) or mtb (binary) file • 0 off • 1 on (only standard plot signals, ascii) • 2 on (more plot signals, ascii) • 3 on (all available plot signals, ascii) • 4 on (only standard plot signals, binary) • 5 on (more plot signals, binary) • 6 on (all available plot signals, binary)

3.3.112. ctiWriteLTIMatrices

Write linearized A,B system matrices to file. Only needed for ADAMS GSE interface.

Prototype:

void ctiWriteLTIMatrices (int th, int∗ier);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 files could not be written

3.3.113. ctiWriteRoadData

Write road data of x/y region swept during previous simulation, in terms of an rgr file. Size and resolution of the file can be set with cosin/tools for tires, 'output' tab.

Prototype:

void ctiWriteRoadData (int th, int*ier);

Parameters:

in	*th*	tire handle
out	*ier*	error flag • 0 ok • 1 file could not be written

3.3.114. ctiWriteStates

State array out (write to file).

Prototype:

void ctiWriteStates (int th, int∗ier, char∗file);

Parameters:

in	*th*	tire handle
out	*ier*	error code • 0 ok • 1 file containing state array could not be written; too many other files open • 3 no state array available for writing since no tire data file loaded • 5 unspecified error writing file containing state array (device full?)
in	*file*	file to save state array to. note that each tire instance needs a separate file.

3.3.115. ctiWriteStatesMemory

State array out (write to memory).

Prototype:

void ctiWriteStatesMemory (int th, int mode, int memsize, double∗mem, int∗ier);

Parameters:

in	*th*	tire handle
in	*mode*	mode flag • 0 write all states and output values • 1 write all states
in	*memsize*	size of memory block 'mem'. Use ctiGetArraySize(flag = 10 + mode) to obtain the CTI state array size for declaration or allocation of the memory block 'mem'.
out	*mem*	memory block to store the states
out	*ier*	error code • 0 ok • 1 provided array size too small. Incompatible

3.3.116. ctiWriteWheelEnvelope

Write wheel envelope.

Prototype:

void ctiWriteWheelEnvelope (int th);

Parameters:

in	*th*	tire handle

3.4. API Type Definition Reference

3.4.1. typedef CTIINIT

Parameter structure for ctiInit:

typedef struct { UMSGF message_func; char output_folder[256]; char output_prefix[64]; int calling_solver; int mt_call_flag; int nr_options; char* options[64]; int major_version; } CTIINIT;

Note: Use CTIINIT_INITIALIZER macro to initialize this structure.

Members:

message_func	message output callback function of typedef UMSGF. If specified, FTire will call this function to pass messages to the calling application.
output_folder	directory to save output files to
output_prefix	output file prefix
calling_solver	calling solver environment • 0: unknown (default)
mt_call_flag	multi-threaded call flag • ..0: single-threaded (default) • ..1: multi-threaded, mandatory if a function from **Cosin Tire Interface Multi-Threading Extension (CTIMT)** is called • .0.: default affinity handling is enabled • .1.: default affinity handling is disabled • 0..: off-line application • 1..: hardware-in-the-loop application
nr_options	number of options stored in options array
options	options array
major_version	cosin major version. Note: Do not change this value!

3.4.2. typedef CTINOTIFY

User-defined notify function:

typedef int(* CTINOTIFY) (void *retbuf);

Parameters:

in, out	*retbuf*	notify return buffer

3.4.3. typedef UMSGF

User-defined message function callback:

typedef void(∗ UMSGF) (int level, char∗message);

Parameters:

in	*level*	severity level • 0: Info • 1: Warning • 2: Error • 3: Fatal error
in	*message*	the message string

3.4.4. typedef URIM

User-defined rim model callback:

typedef void(∗ URIM) (int th, int nseg, double rrim, double wrim, double t, double fl[ ][3], double fr[ ][3], double del[ ][3], double der[ ][3], double dpl[ ][3], double dpr[ ][3], int∗ier, char∗rim_file);

Parameters:

in	*th*	tire handle (tire instance if using deprecated function ctiSetURIMFunc)
in	*nseg*	number of equally distributed nodes on one rim flange
in	*rrim*	rim bead radius[m]
in	*wrim*	axial rim flanges distance[m]
in	*t*	simulation time, terminate model if t>= 1e60.[s]
in	*fl*	force array on left rim flange nodes, in cylinder coordinates.[N]
in	*fr*	force array on right rim flange nodes, in cylinder coordinates.[N]
out	*del*	elastic displacements of left rim flange nodes, in cylinder coordinates.[m]
out	*der*	elastic displacements of right rim flange nodes, in cylinder coordinates.[m]
in,out	*dpl*	plastic deformation of left rim flange nodes, in cylinder coordinates.[m]
in,out	*dpr*	plastic deformation of right rim flange nodes, in cylinder coordinates.[m]
out	*ier*	error code • 0: ok
in	*rim_file*	data file name

3.4.5. typedef URM

User-defined road model callback:

typedef void(∗ URM) (int th, double t, double x, double y, double∗z, double∗vx, double∗vy, double∗vz, double∗mu, int∗ier, char∗road_file);

Parameters:

in	*th*	tire handle (tire instance if using deprecated function ctiSetURMFunc)
in	*t*	simulation time, terminate model if t>= 1e60.[s]
in	*x*	x-coordinate of point for road evaluation, in global coordinates[m]. If road is moved with ctiSetRoadMotionData or similar, position is relative to this moving coordinate system
in	*y*	y-coordinate of point for road evaluation, in global coordinates[m]. If road is moved with ctiSetRoadMotionData or similar, position is relative to this moving coordinate system
out	*z*	evaluated road height, in global coordinates[m]. If road is moved with ctiSetRoadMotionData or similar, height is relative to this moving coordinate system
out	*vx*	evaluated x-component of road surface velocity, in global coordinates[m/s]. If road is moved with ctiSetRoadMotionData or similar, velocity is relative to this moving coordinate system
out	*vy*	evaluated y-component of road surface velocity, in global coordinates[m/s]. If road is moved with ctiSetRoadMotionData or similar, velocity is relative to this moving coordinate system
out	*vz*	evaluated z-component of road surface velocity, in global coordinates[m/s]. If road is moved with ctiSetRoadMotionData or similar, velocit is relative to this moving coordinate system
out	*mu*	evaluated road/tread friction coefficient correction factor (typically=1.0)
out	*ier*	error code • 0: ok
in	*road_file*	data file name

3.4.6. typedef USM

User-defined soil model callback (See also SAE_2008_M20_Paper_Gipser.pdf):

typedef void(∗ USM) (double x0, double dx, int nx, double y0, double dy, int ny, double phi, double fx[], double fy[], double fz[], double z[], double vx[], double vy[], double vz[], double mu[], int th, int mode, double dt, char∗soil_file);

Parameters:

in	*x0*	grid origin in global coordinates, provided by the tire model.[m]
in	*dx*	grid spacing in x/y-direction, provided by the tire model.[m]
in	*nx*	number of grid points in x/y- direction, provided by the tire model.
in	*y0*	grid origin in global coordinates, provided by the tire model.[m]

in	*dy*	grid spacing in x/y-direction, provided by the tire model.[m]
in	*ny*	number of grid points in x/y- direction, provided by the tire model.
in	*phi*	counter-clockwise grid rotation angle about z-axis, provided by the tire model.[deg]
in	*fx*	x-components of contact forces in global coordinates, provided by the tire model.[N]
in	*fy*	y-components of contact forces in global coordinates, provided by the tire model.[N]
in	*fz*	z-components of contact forces in global coordinates, provided by the tire model.[N]
out	*z*	z-elevations of grid points, returned by the soil model.[m]
out	*vx*	x-components of grid point velocities in global coordinates, returned by the soil model.[m/s]
out	*vy*	y-components of grid point velocities in global coordinates, returned by the soil model.[m/s]
out	*vz*	z-components of grid point velocities in global coordinates, returned by the soil model.[m/s]
out	*mu*	sliding friction modification factors in grid points, returned by the soil model.
in	*th*	tire handle (tire instance if using deprecated function ctiSetUSMFunc) of calling tire, to be used in soil model to select the respective tire's parameter and state arrays; provided by the tire model.
in	*mode*	discriminates between initialization (0), normal mode (1), and termination (99), provided by the tire model. • 0: initialize the soil model instance given by tire instance, using data file soil_model_data_file • 1: call the soil model which applies the contact forces and advances its state variables according to time-step dt; compute and return new grid elevations and velocities for soil-model instance given by tire instance • 99: terminate the soil model instance given by tire instance
in	*dt*	current simulation time step[s]
in	*soil_file*	name of the file that contains the data of the soil model. Passed through from the calling simulation program via the tire model to the soil model.

3.5. Deprecated API Function Reference

Deprecated CTI Function	Recommended CTI Function	Remarks
ctiCallingSolver	ctiInit	set calling_solver intypedef CTIINIT
ctiGetOperatingConditions	ctiReadOperatingConditions	
ctiGetStates	ctiWriteStates	
ctiInitialize	ctiInit	set calling_solver, output_folder and output_prefix intypedef CTIINIT
ctiPutContactBodyForces	ctiGetContactBodyForces	
ctiPutLTIMatrix	ctiGetLTIMatrix	
ctiPutNodePositions	ctiGetNodePositionsWithAttributes	
ctiPutOutputSignal	ctiGetOutputSignal	
ctiPutOutputSignalNumber	ctiGetOutputSignalNumber	
ctiPutRimForces	ctiGetRimForces	
ctiPutRimProperties	ctiGetRimProperties	
ctiPutRimRotationStates	ctiGetRimRotationStates	
ctiPutRoadForces	ctiGetRoadForces	
ctiPutStates	ctiReadStatesMemory	
ctiPutTireKeyData	ctiGetTireKeyData	
ctiPutTireProperties	ctiGetTireProperties	
ctiPutTreadStates	ctiGetTreadStates	
ctiPutTydexSignals	ctiGetTydexSignals	
ctiSetMessageFunc	ctiInit	set message_func intypedef CTIINIT
ctiSetModelLevel		
ctiSetURIMFunc	ctiSetURIM	
ctiSetURMFunc	ctiSetURM	
ctiSetUSMFunc	ctiSetUSM	

4. Cosin Tire Interface Multi-Threading Extension (CTIMT)

FTire and HTire solver allows to run the time integration of multiple tire instances in parallel. This can save a significant amount of computing time on multi-CPU systems.

The multi-threading extension of **CTI** are an extension to the API functions, providing multi-threaded evaluation calls.

In the multi-threading extension of **CTI**, an extra program thread is assigned to every tire instance. Passing information to and from these threads is organized within **CTI**. Typically, in every time step of the calling integrator, every thread receives actual values of its input signals (like the respective wheel position and velocity values). Then, it will advance one step in time and return the resulting output signals (like the respective wheel forces and moments) in commonly known variables. After completion of the time step, the thread will wait until it is triggered by **CTI** to perform the next step.

Obviously, all threads may (and should) run in parallel. They are not directly interfering in any way with each other. Somehow simplified, this property is called 'thread-safe'. Best performance will be reached if, in a single time step, all threads receive their input signals at the beginning of the step and then start computing simultaneously. The calling program starts waiting for the results only after **all** threads have been made busy.

Some parts of the initialization of **CTI** inevitably must run sequentially, for several software architectural reasons. Anyway, this is not relevant with respect to computing time. The only potentially time-consuming computation during initialization is the FTire pre-processing in case its data have changed. But this pre-processing, in most cases, is required only for one tire instance. The others share the same pre-processed data. Because of this, pre-processing cannot be parallelized anyway.

4.1. Program Structure of CTI/MT Applications

Typical multi-threaded **CTI** applications will be structured as follows:

1. Call `ctiInit` to initialize the interface.
2. Loop over all tire instances to be computed, loading tire and road data with routines `ctiLoadTireData` and `ctiLoadRoadData`, as usual.
3. Enter the time loop. In every time step,
   a) loop over all tire instances. Update wheel position and velocity variables and tell the respective thread to perform one step, using the **CTI** functions `ctiComputeForcesMT` or `ctiComputeForcesOnWCarrierMT`
   b) in a second loop, only after having triggered **all** threads by completing the first loop, wait for all threads to complete the current step and return the resulting wheel forces and moments. Both is performed by the **CTI** function `ctiGetForcesMT`;
4. Terminate all threads and **CTI** functions by calling `ctiClose`, as usual.

The complete time-loop part in step (2) of this algorithm is implemented in the two routines `ctiComputeForcesList` and `ctiComputeForcesOnWCarrierList`. These routines act as if they were simultaneous calls to routine `ctiComputeForces` or `ctiComputeForcesOnWheelCarrier`, respectively, but in full multi-threaded mode.

Note that, when using the latter two user-friendly routines, neither `ctiComputeForcesMT`, `ctiComputeForcesOnWCarrierMT` nor `ctiGetForcesMT` is required. Those routines might be necessary if the tire instances are treated by the calling solver at different times or in different places. However, users

are strongly encouraged to use `ctiComputeForcesList` and `ctiComputeForcesOnWCarrierList` instead of `ctiComputeForcesMT`, `ctiComputeForcesOnWCarrierMT`, or`ctiGetForcesMT`.

Simple main programs demonstrating multi-threaded **CTI** application are contained in the ctiMtDemo.c.

4.2. API Function Reference

4.2.1. ctiComputeForcesList

Compute forces for list of tire instances in multi-threaded mode.

Prototype:

void ctiComputeForcesList (int nthl, int thl[], double t, double r0r0[][3], double a0r[][9], double v0r0[][3], double w0r0[][3], int mode, double fr0[][3], double trr0[][3], int∗ier);

Parameters:

`in`	*nthl*	number of tire instances
`in`	*thl*	list of tire handles
`in`	*t*	simulation time[s]
`in`	*r0r0*	rigid-body states of rims
`in`	*a0r*	rigid-body states of rims
`in`	*v0r0*	rigid-body states of rims
`in`	*w0r0*	rigid-body states of rims

in	*mode*	job control • ...0 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have not yet been accepted by external integrator • ...1 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have been accepted by external integrator • ...2 unconditionally (re-)calculate tire forces • ...3 calculate steady-state tire forces • ...4 calculate static tire forces, avgd. road • ...5 calculate static tire forces, enhanced accuracy, avgd. road • ...6 calculate static tire forces, time-/loc-dependent road • ...9 reset (prepare for next time loop w/o closing tire handle) • ..k. compute steady states first, if not yet done (only in dynamic case); repeat this in the first k>0 dynamic steps • .1.. extrapolate forces/moments to next time step (only in dynamic case), for use if calling solver is to run in parallel with CTI • 0... enable multi-threading • 1... disable multi-threading
out	*fr0*	forces acting on rim centers[N]
out	*trr0*	torques acting on rim centers[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted • 2 no license

4.2.2. ctiComputeForcesListMT

Compute forces for list of tire instances in multi-threaded mode.

Prototype:

void ctiComputeForcesListMT (int nthl, int thl[], double t, double r0r0[][3], double a0r[][9], double v0r0[][3], double w0r0[][3], int mode, int∗ier);

Parameters:

in	*nthl*	number of tire instances
in	*thl*	list of tire handles
in	*t*	simulation time[s]
in	*r0r0*	rigid-body states of rims
in	*a0r*	rigid-body states of rims
in	*v0r0*	rigid-body states of rims
in	*w0r0*	rigid-body states of rims
in	*mode*	job control • ...0 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have not yet been accepted by external integrator • ...1 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have been accepted by external integrator • ...2 unconditionally (re-)calculate tire forces • ...3 calculate steady-state tire forces • ...4 calculate static tire forces, avgd. road • ...5 calculate static tire forces, enhanced accuracy, avgd. road • ...6 calculate static tire forces, time-/loc-dependent road • ..1. compute steady states first, if not yet done (only in dynamic case) • .1.. extrapolate forces/moments to next time step (only in dynamic case), for use if calling solver is to run in parallel with CTI • 0... enable multi-threading • 1... disable multi-threading
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted • 2 no license

4.2.3. ctiComputeForcesMT

Compute forces in multi-threading mode.

Prototype:

void ctiComputeForcesMT (int th, double t, double r[], double a[], double v[], double w[], int mode, int∗ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time[s]
in	*r*	rigid-body state of rim r = position of rim center in global coordinates[m]
in	*a*	rigid-body state of rim a = 3x3 orthogonal transformation matrix from rim-fixed frame to global coordinates. Vectors in rim-fixed frame are to be multiplied by 'a' to result in the representation in global coordinates. Example: a∗ (0 1 0)' = 2nd column of a0r = direction vector of wheel spinning axis in global coordinates. Note: a is stored column-wise (like matrices are stored in FORTRAN, but unlike it is done in C), a = (a_11, a_21, a_31, a_12, ...)
in	*v*	rigid-body state of rim v = translational velocity of rim center in global coordinates: v = d/dt(r)[m/s][m/s]
in	*w*	rigid-body state of rim w = angular velocity vector of rim relative to global coordinates, represented in global coordinates[rad/s][rad/s]
in	*mode*	job control • 0 calculate (if not yet available) or return (if available) dynamic tire forces. simulation time t have not yet been accepted by external integrator • 1 calculate (if not yet available) or return (if available) dynamic tire forces. simulation time T have been accepted by external integrator • 2 (re-)calculate tire forces regardless on previous success of external integrator • 3 calculate steady-state tire forces • 4 calculate static tire forces • 10 like 0, compute steady states first, if not yet done • 11 like 1, compute steady states first, if not yet done
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log)

4.2.4. ctiComputeForcesOnWCarrierList

Alternative main routine, coupling the tire model to wheel carrier instead of rim. Compute forces for list of tire instances in multi-threaded mode.

Prototype:

void ctiComputeForcesOnWCarrierList (int nthl, int thl[], double t, double r0r0[][3], double a0r[][9], double v0r0[][3], double w0r0[][3], double tdr[], double tbr[], int mode, double fr0[][3], double trr0[][3], int∗ier);

Parameters:

in	*nthl*	number of tire instances
in	*thl*	list of tire handles
in	*t*	simulation time[s]
in	*r0r0*	rigid-body states of rims
in	*a0r*	rigid-body states of rims
in	*v0r0*	rigid-body states of rims
in	*w0r0*	rigid-body states of rims
in	*tdr*	drive/brake torque
in	*tbr*	drive/brake torque
in	*mode*	job control • ...0 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have not yet been accepted by external integrator • ...1 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have been accepted by external integrator • ...2 unconditionally (re-)calculate tire forces • ...3 calculate steady-state tire forces • ...4 calculate static tire forces, avgd. road • ...5 calculate static tire forces, enhanced accuracy, avgd. road • ...6 calculate static tire forces, time-/loc-dependent road • ..1. compute steady states first, if not yet done (only in dynamic case) • .1.. extrapolate forces/moments to next time step (only in dynamic case), for use if calling solver is to run in parallel with CTI • 0... enable multi-threading • 1... disable multi-threading
out	*fr0*	forces acting on rim centers[N]
out	*trr0*	torques acting on rim centers[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted • 2 no license

4.2.5. ctiComputeForcesOnWCarrierMT

Alternative routine, coupling the tire model to wheel carrier instead of rim. Compute forces in multi-threaded mode.

Prototype:

void ctiComputeForcesOnWCarrierMT (int th, double t, double r[], double a[], double v[], double w[], double tdr, double tbr, int mode, int*ier);

Parameters:

in	*th*	tire handle
in	*t*	simulation time
in	*r*	rigid-body state of wheel carrier r = position of wheel carrier in global coordinates[m]
in	*a*	rigid-body state of wheel carrier a = 3x3 orthogonal transformation matrix from wheel-carrier- fixed frame to global coordinates. Vectors in wheel-carrier- fixed frame are to be multiplied by 'a' to result in the representation in global coordinates. Example: a* (0 1 0)' = 2nd column of a0r = direction vector of wheel-spin axis in global coordinates. Note: a is stored column-wise (like matrices are stored in FORTRAN, but unlike it is done in C), a = (a_11, a_21, a_31, a_12, ...)
in	*v*	rigid-body state of wheel carrier v = translational velocity of wheel carrier in global coordinates: v = d/dt(r)[m/s][m/s]
in	*w*	rigid-body state of wheel carrier w = angular velocity vector of wheel carrier relative to global coordinates, represented in global coordinates[rad/s][rad/s]
in	*tdr*	drive torque as put out by the drive-train model. Only scalar component in direction of spindle. The calling program will have to take care that the reaction torque of tdr is applied to the appropriate part of the drive-train model[Nm]
in	*tbr*	brake torque as put out by the brake model. Only scalar component in direction of spindle. tbr is understood to be the maximum absolute brake torque which is in effect when the wheel is rolling. The tire model will compute and apply the effective brake torque. This will be negative when the wheel is rolling backward, and have smaller absolute value when the wheel rotation is locked. The calling program does not need to compute any reaction torque. In contrast to tdr, CTI treats tbr as an inner torque, acting between rim and wheel carrier[Nm]

in	*mode*	job control • 0 calculate (if not yet available) or return (if available) dynamic tire forces. simulation time t have not yet been accepted by external integrator • 1 calculate (if not yet available) or return (if available) dynamic tire forces. simulation time T have been accepted by external integrator • 2 (re-)calculate tire forces regardless on previous success of external integrator • 3 calculate steady-state tire forces • 4 calculate static tire forces • 10 like 0, compute steady states first, if not yet done • 11 like 1, compute steady states first, if not yet done
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

4.2.6. ctiComputeForcesWithExtRoadList

Main routine with external road function with external road function; list of tire instances to be computed in parallel.

Prototype:

void ctiComputeForcesWithExtRoadList (int nthl, int thl[], double t, double r0r0[][3], double a0r[][9], double v0r0[][3], double w0r0[][3], void∗road, int mode, double fr0[][3], double trr0[][3], int∗ier);

Parameters:

in	*nthl*	number of tire instances
in	*thl*	list of tire handles
in	*t*	simulation time
in	*r0r0*	rigid-body state of rim r0r0 = position of rim center in global coordinates[m]
in	*a0r*	rigid-body state of rim a0r = 3x3 orthogonal transformation matrix from rim-fixed frame to global coordinates. Vectors in rim-fixed frame are to be multiplied by a0r to result in the representation in global coordinates.
in	*v0r0*	rigid-body state of rim v0r0 = translational velocity of rim center in global coordinates: v0r0 = d/dt(r0r0)[m/s][m/s]
in	*w0r0*	rigid-body state of rim w0r0 = angular velocity vector of rim relative to global coordinates, represented in global coordinates[rad/s][rad/s]
in	*road*	name of road model subroutine used in EVRMR

in	*mode*	job control • ...0 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have not yet been accepted by external integrator • ...1 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have been accepted by external integrator • ...2 unconditionally (re-)calculate tire forces • ...3 calculate steady-state tire forces • ...4 calculate static tire forces, avgd. road • ...5 calculate static tire forces, enhanced accuracy, avgd. road • ...6 calculate static tire forces, time-/loc-dependent road • ..1. compute steady states first, if not yet done (only in dynamic case) • .1.. extrapolate forces/moments to next time step (only in dynamic case), for use if calling solver is to run in parallel with CTI • 0... enable multi-threading • 1... disable multi-threading
out	*fr0*	force acting on rim center (point of attack = wheel center) expressed in global coordinates[N]
out	*trr0*	torque acting on rim center, expressed in global coordinates[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted • 999 no license

4.2.7. ctiComputeForcesWithExtRoadMT

Compute forces in multi-\/threading mode with external road function.

Prototype:

void ctiComputeForcesWithExtRoadMT (int th, double t, double r[], double a[], double v[], double w[], void∗road, int mode, int∗ier);

Parameters:

in	*th*	tire handle

in	*t*	simulation time
in	*r*	rigid-body state of rim r = position of rim center in global coordinates[m]
in	*a*	rigid-body state of rim a = 3x3 orthogonal transformation matrix from rim-fixed frame to global coordinates. Vectors in rim-fixed frame are to be multiplied by 'a' to result in the representation in global coordinates. Example: a∗ (0 1 0)' = 2nd column of a0r = direction vector of wheel-spin axis in global coordinates. Note: a is stored column-wise (like matrices are stored in Fortran and Matlab, but unlike it is done in C and C++), a = (a_11, a_21, a_31, a_12, ...)
in	*v*	rigid-body state of rim v = translational velocity of rim center in global coordinates: v = d/dt(r)[m/s][m/s]
in	*w*	rigid-body state of rim w = angular velocity vector of rim relative to global coordinates, represented in global coordinates[rad/s][rad/s]
in	*road*	name of road model subroutine used in EVRMR
in	*mode*	job control • ...0 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have not yet been accepted by external integrator • ...1 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have been accepted by external integrator • ...2 unconditionally (re-)calculate tire forces • ...3 calculate steady-state tire forces • ...4 calculate static tire forces, avgd. road • ...5 calculate static tire forces, enhanced accuracy, avgd. road • ...6 calculate static tire forces, time-/loc-dependent road • ..1. compute steady states first, if not yet done (only in dynamic case) • .1.. extrapolate forces/moments to next time step (only in dynamic case), for use if calling solver is to run in parallel with CTI • 0... enable multi-threading • 1... disable multi-threading
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted • 999 no license

4.2.8. ctiGetForcesListMT

Get forces for list of tire instances in multi-threading mode.

Prototype:

void ctiGetForcesListMT (int nthl, int thl[], double fr0[][3], double trr0[][3], int*ier);

Parameters:

in	*nthl*	number of tire instances
in	*thl*	list of tire handles
out	*fr0*	forces acting on rim centers[N]
out	*trr0*	torques acting on rim centers[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

4.2.9. ctiGetForcesMT

Get forces in multi-threading mode.

Prototype:

void ctiGetForcesMT (int th, double force[], double torque[], int*ier);

Parameters:

in	*th*	tire handle
out	*force*	force acting on wheel carrier (point of attack = wheel center), expressed in global coordinates[N]
out	*torque*	torque acting on wheel carrier, expressed in global coordinates[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

4.2.10. ctiWriteStatesMemoryList

State array out (write to memory) for list of tire instances in multi-threaded mode.

Prototype:

void ctiWriteStatesMemoryList (int nthl, int thl[], int mode, int memsize[], double* mem[], int*ier);

Parameters:

in	*nthl*	number of tire instances
in	*thl*	list of tire handles

in	*mode*	mode flag • 0 write all states and output values • 1 write all states
in	*memsize*	list of sizes of memory blocks 'mem'. Use ctiGetArraySize (flag = 10 + mode) to obtain the CTI state array size for declaration or allocation of the memory block 'mem'. Note: if memsize == NULL, 'memsize' info from ctiSetStatesMemory is used
out	*mem*	list of memory blocks to store the states Note: if mem == NULL, 'mem' info from ctiSetStatesMemory is used
out	*ier*	error flag • 0 ok

5. CTI Dynamic Library Wrapper

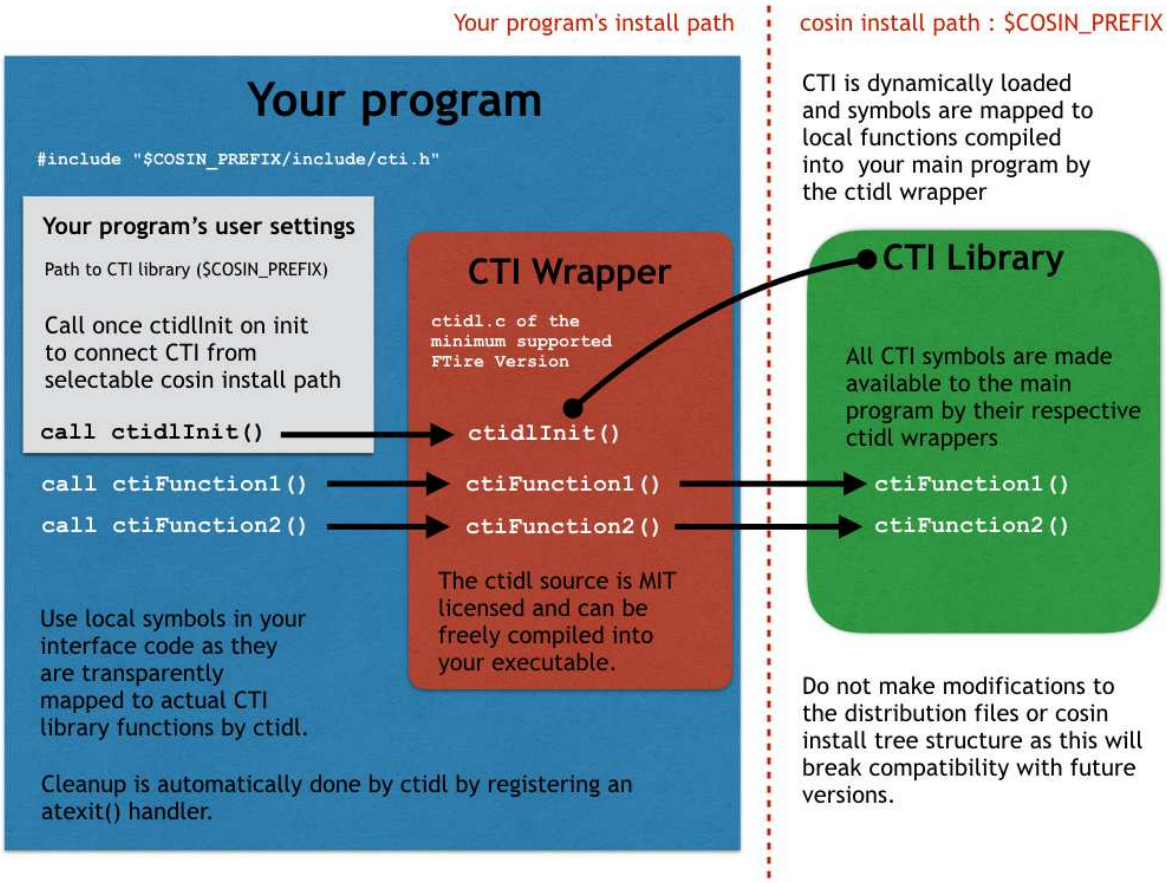

Figure 1: Using CTI wrapper in external program to load CTI functions at runtime

The CTI dynamic library wrappers are stored in a single C file

<cosin-install-folder>/interface/cti/ctidl.c

licensed under the MIT License to freely compile it to your program. It contains for every CTI function a wrapper coupling the corresponding CTI shared library function. To use the wrapper an initialization function ctidlInit needs to be called first in your program. The main information required by the wrapper initialization is the current cosin installation folder, the easiest way to get this folder is to call ctidlGetCosinInstallFolder. The finalization function ctidlClose is per default called via atexit but can also be called be the user if intended. The above folder <cosin-install-folder>/interface/cti/ contains a few demo files where all of them using the ctidl.c wrappers [1].

[1] E.g. gcc -o ctiDemo -I../../include ctiDemo.c ctidl.c OR cl.exe -FectiDemo.exe -I../../include ctiDemo.c ctidl.c

5.1. API Function Reference

5.1.1. ctidlClose

Finalize dynamic library wrappers.

Prototype:

void ctidlClose (void);

Parameters:

none

5.1.2. ctidlGetCosinGuiPath

Get cosin/tools exectuable path.

Prototype:

void ctidlGetCosinGuiPath (char* buf, int size, int* ier);

Parameters:

in	*buf*	Buffer to store the cosin/tools path
in	*size*	Size, in bytes, of the array referenced by 'buf'
out	*ier*	error flag • 0 ok • 1 could not get cosin installation folder

5.1.3. ctidlGetCosinInstallFolder

Get cosin installation folder.

Prototype:

void ctidlGetCosinInstallFolder (char* buf, int size, int* ier);

Parameters:

in	*buf*	Buffer to store the cosin installation folder
in	*size*	Size, in bytes, of the array referenced by 'buf'
out	*ier*	error flag • 0 ok • 1 could not found HOME folder • 2 could not found/open any ini file • 3 install folder tag not found in ini file

Document Revision:2018-4-r18799

Note: A valid COSIN_PREFIX environment variable has priority over the setting specified in the cosin ini file!

5.1.4. ctidlInit

Initialize dynamic library wrappers.

Prototype:

void ctidlInit (CTIDLINIT*param, int*ier);

Parameters:

in	*param*	parameter structure of typedef CTIDLINIT Note: Use CTIDLINIT_INITIALIZER to initalize this structure.
out	*ier*	error flag • 0 ok • 1 could not open cti shared library • 2 at least one symbol is mssing in the CTI shared library. Notes: – A list with all missing symbols will be printed if `param->message_func` is specified. – Using missing symbols can lead to undefined behavior.

5.1.5. ctidlOpenCosinGui

Open cosin/tools.

Prototype:

void ctidlOpenCosinGui (void);

Parameters:

none

5.1.6. ctidlOpenRoadGui

Open the cosin road GUI with the given road file.

Prototype:

void ctidlOpenRoadGui (char* road_file);

Parameters:

in	*road_file*	full path to road file

5.1.7. ctidlOpenTireGui

Open the cosin tire GUI with the given tire file.

Prototype:

void ctidlOpenTireGui (char* tire_file);

Parameters:

in	*tire_file*	full path to tire file

5.1.8. typedef CTIDLINIT

Parameter structure for ctidlInit:

typedef struct { UMSGF message_func; int init_mode; int close_mode; char cosin_install_folder[256];} CTIDLINIT;

Note: Use CTIDLINIT_INITIALIZER macro to initialize this structure.

Members:

message_func	message output callback function of typedef UMSGF. If specified, FTire will call this function to pass messages to the calling application.
close_mode	close mode • 0: ctidlClose call will be registered via atexit, no manual call of ctidlClose necessary (default) • 1: user has to call ctidlClose manually
init_mode	init mode • 0: load ALL cti shared library functions (default)
cosin_install_folder	cosin installation folder • cosin_install_folder[0] == '\0': the current working directory is used (default)

Document Revision:2018-4-r18799

6. CTI/server Client Interface (CTICLI)

6.1. Program Structure of CTICLI Applications

The CTI client API (CTICLI) provides command redirection to a remote CTI/server. No computation is done on the local machine, all requests are forwarded to the server.

CTICLI aims to provide equivalent calls for every CTI function call. However, not every CTI call can map to a CTICLI function by concept (e.g. animation). on the other hand, CTICLI provides additional calls, having no equivalents in CTI, for CTI/server administration. A comparison table of CTI and CTICLI is given in6.3.

CTICLI can be used in two different ways:

- by calling the standard CTI functions and an addtional call to ctiSetServerin the CTI initialization phase (see6.2)
- by replacing the CTI functions with their respective CTICLI equivalents in the user code.

Fig.2 shows an application using a CTICLI library to forward CTI calls to a remote CTI/server.

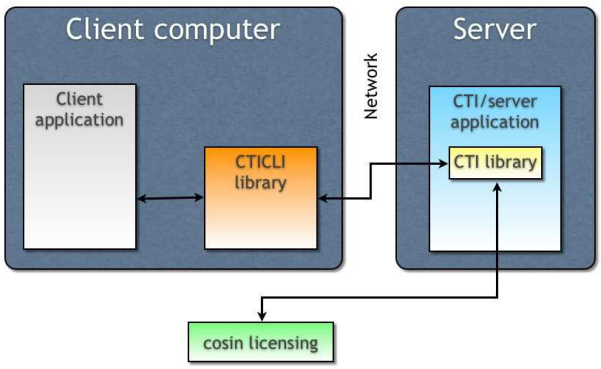

Figure 2: Call to CTI/server using dynamically linked CTICLI library

The CTICLI library can be compiled and linked statically with the calling application (Fig.3), e.g. in a HIL environment. CTICLI is available as source code on request.

Document Revision:2018-4-r18799

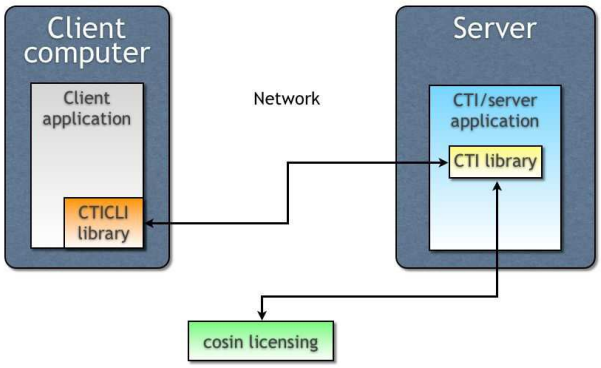

Figure 3: Call to CTI/server using statically linked CTICLI library

6.2. CTI / CTICLI gateway

CTICLI functionality is included with the CTI library. The calling application can connect to a CTI/server using the ctiSetServer function call (see3.3.98). All subsequent calls to CTI functions will be redirected to the CTI/server.

Note The calling application has to check error return codes very carefully. If the server connection fails, all subsequent CTI calls will we handled by the local CTI library.

Fig.4 shows an application calling a local CTI library.

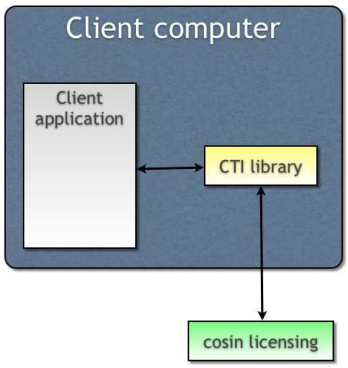

Figure 4: Call to local CTI library

After connecting to the target CTI/server, all computation is done on the remote host.

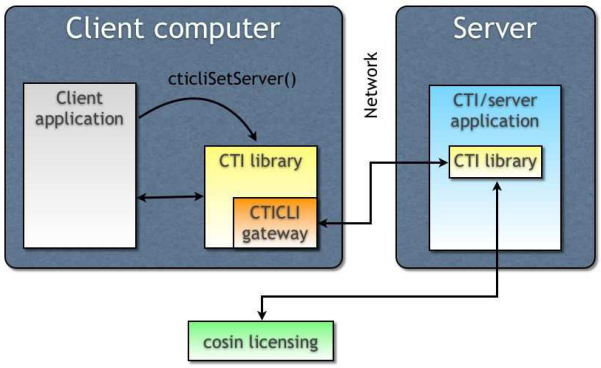

Figure 5: Call to a CTI/server using the CTI gateway

6.3. CTICLI Function Coverage

The CTICLI functions are classified into:

6.3.1. Client handles

Client handles are unique identifiers of the current client. The client handles are freely definable by the user (also negative values are allowed).

6.3.2. CTICLI functions with a corresponding CTI function and identical parameter list

The parameter lists of the CTICLI functions in this class are identical to the CTI function, e.g.

CTICLI Prototype	CTI Prototype
int cticliCloseTire (int ch, int th);	void ctiCloseTire (int th);
int cticliVerbose (int ch, int th, int v);	void ctiVerbose (int th, int v);

except the leading client handle 'int ch' parameter and the error code return for the CTICLI function:

CTICLI Function	TCP Support	UDP Support	Associated CTI Function	Remarks
cticliAdjustTwinTireWheelSpeed	x		ctiAdjustTwinTireWheelSpeed	
cticliClose	x	x	ctiClose	
cticliCloseTire	x		ctiCloseTire	
cticliComputeForces	x	x	ctiComputeForces	
cticliComputeForcesList	x	x	ctiComputeForcesList	UDP support is restricted to 4 tires because of UDP packet size

cticliComputeForcesListMT	x	x	ctiComputeForcesListMT	UDP support is restricted to 4 tires because of UDP packet size
cticliComputeForcesOnCarBody	x		ctiComputeForcesOnCarBody	
cticliComputeForcesOnWCarrierList	x		ctiComputeForcesOnWCarrierList	
cticliComputeForcesOnWheelCarrier	x		ctiComputeForcesOnWheelCarrier	
cticliEvaluateRoadCourse	x		ctiEvaluateRoadCourse	
cticliEvaluateRoadHeight	x		ctiEvaluateRoadHeight	
cticliFindOutputSignalNumber	x		ctiFindOutputSignalNumber	
cticliGetContactBodyForces	x		ctiGetContactBodyForces	
cticliGetNodePositions	x		ctiGetNodePositionsWithAttributes	
cticliGetOutputSignal	x		ctiGetOutputSignal	
cticliGetOutputSignalNumber	x		ctiGetOutputSignalNumber	
cticliGetRimForces	x		ctiGetRimForces	
cticliGetRimProperties	x		ctiGetRimProperties	
cticliGetRimRotationStates	x		ctiGetRimRotationStates	
cticliGetRoadForces	x		ctiGetRoadForces	
cticliGetRoadParameters	x		ctiGetRoadParameters	
cticliGetRoadSize	x		ctiGetRoadSize	
cticliGetStepSize	x		ctiGetStatus	
cticliGetTireDimensionData	x		ctiGetTireDimensionData	
cticliGetTireDimensionStringData	x		ctiGetTireDimensionStringData	
cticliGetTireHandle	x		ctiGetTireHandle	
cticliGetTireInstance	x		ctiGetTireInstance	
cticliGetTireKeyData	x	x	ctiGetTireKeyData	
cticliGetTireModelType	x		ctiGetTireModelType	
cticliGetTireProperties	x		ctiGetTireProperties	
cticliGetTreadStates	x		ctiGetTreadStates	
cticliGetTydexSignals	x	x	ctiGetTydexSignals	UDP support is restricted to 60 values because of UDP packet size.
cticliLinearize	x		ctiLinearize	
cticliLinearizeWheelCarrier	x		ctiLinearizeWheelCarrier	
cticliModifyFriction	x		ctiModifyFriction	
cticliOpenOutputFile	x		ctiOpenOutputFile	
cticliReadOperatingConditions	x		ctiReadOperatingConditions	
cticliReadStates	x		ctiReadStates	
cticliRecorder	x		ctiRecorder	Output file has to be downloaded at simulation end
cticliSaveRecordedForcesMoments	x		ctiSaveRecordedForcesMoments	Output file has to be downloaded at simulation end

cticliSetAffinity	x	x	ctiSetAffinity	
cticliSetAmbientTemperature	x		ctiSetAmbientTemperature	
cticliSetAnimationStepSize	x		ctiSetAnimationStepSize	
cticliSetContactBodyMotionData	x		ctiSetContactBodyMotionData	
cticliSetDesignParameter	x		ctiSetDesignParameter	
cticliSetInflationPressure	x		ctiSetDrumTorque	
cticliSetInitialTemperature	x		ctiSetInitialRimAngle	
cticliSetInitialTireTemperatures	x		ctiSetInitialTireTemperatures	
cticliSetIntegerRoadParameter	x		ctiSetIntegerRoadParameter	
cticliSetMultiThreadedCallFlag	x	x	ctiSetMultiThreadedCallFlag	
cticliSetOutputStepSize	x		ctiSetOutputFilePrefix	
cticliSetRoadMotionData	x		ctiSetRoadMotionData	
cticliSetRoadParameters	x		ctiSetRoadParameters	
cticliSetRoadTemperature	x		ctiSetRoadTemperature	
cticliSetRunTimeMode	x	x	ctiSetRunTimeMode	
cticliSetTimeConstantForces	x		ctiSetTimeConstantForces	
cticliSetTireSide	x		ctiSetTireSide	
cticliSetTreadDepth	x		ctiSetTreadDepth	
cticliSetWheelCenterRefPosition	x		ctiSetWheelCenterRefPosition	
cticliUpdateWheelEnvelope	x		ctiUpdateWheelEnvelope	
cticliVerbose	x		ctiVerbose	
cticliWriteAdditionalOutput	x		ctiWriteAdditionalOutput	Output file has to be downloaded at simulation end
cticliWriteStates	x		ctiWriteStates	
cticliWriteWheelEnvelope	x		ctiWriteWheelEnvelope	Output file has to be downloaded at simulation end

6.3.3. CTICLI functions with a corresponding CTI function and different parameter list

CTICLI Function	TCP Support	UDP Support	Associated CTI Function	Remarks
cticliInit	x	x	ctiInit	
cticliLoadRimData	x	x	ctiLoadRimData	
cticliLoadRoadData	x	x	ctiLoadRoadData	
cticliLoadSuspensionData	x	x	ctiLoadSuspensionData	
cticliLoadTireData	x	x	ctiLoadTireData	
cticliGetForcesListMT	x	x	ctiGetForcesListMT	UDP version is restricted to 4 tires because of UDP packet size

6.3.4. CTICLI functions without a corresponding CTI function

CTICLI Function	TCP Support	UDP Support	Associated CTI Function	Remarks
cticliDownloadFile	x		N/A	
cticliGetServerStats	x		N/A	
cticliListFiles	x		N/A	
cticliLoadCtiLibrary	x		N/A	
cticliUploadFile	x		N/A	
cticliComputeForcesWithOutputArrayList	x	x	N/A	
cticliGetForcesWithOutputArrayListMT	x	x	N/A	

6.3.5. CTI functions without a corresponding CTICLI function

CTI Function	Remarks
ctiAnimate	Server side animation not applicable
ctiAnimateOnly	
ctiAnimateScene	
ctiAnimateSceneWithExtRoad	
ctiCheckLicense	
ctiComputeForcesMT	
ctiComputeForcesOnWCarrierMT	
ctiComputeForcesTimeContinuous	
ctiComputeForcesTimeContinuousWithExtRoad	Passing local function pointer is not supported on server side
ctiComputeForcesWithExtRoad	
ctiComputeForcesWithExtRoadList	
ctiComputeForcesWithExtRoadMT	
ctiEnableTimeContinuous	
ctiGetArraySize	
ctiGetCosinSoftwareVersion	
ctiGetFileName	
ctiGetForcesMT	
ctiGetInstallationInfo	
ctiGetLTIMatrix	
ctiGetNumberContinuousStates	
ctiGetOutputSignals	
ctiKillSolverOnEsc	Client disconnection will always end CTI session
ctiLoadRimModel	Access to dynamic libraries is not applicable on server side
ctiLoadRoadModel	
ctiLoadSTIRoadModel	
ctiLoadSTITireModel	
ctiLoadSoilModel	
ctiOpenRoadGui	
ctiOpenTireGui	

ctiReadLTIMatrices	Not supported on server side
ctiReadStatesMemory	Server has no access to client memory
ctiReset	
ctiSetCompatVersion	
ctiSetDiagMode	
ctiSetNotify	
ctiSetOption	
ctiSetPPTireDataFilename	Preprocessing is always written to separate file on server side
ctiSetRoadEvalPreference	
ctiSetStatesMemory	Server has no access to client memory
ctiSetServer	
ctiSetURIM	
ctiSetURM	
ctiSetUSM	
ctiSetVehicleStates	
ctiUpdateRoadData	
ctiWriteLTIMatrices	
ctiWriteStatesMemory	Server has no access to client memory
ctiWriteStatesMemoryList	Server has no access to client memory

6.4. API Function Reference

Parameters of the CTICLI functions are corresponding to the parameters of the respective CTI function. See the CTI API documentation (3.3) for a detailed description. CTICLI functions have an additional first parameter client handle (returned by**cticliInit**) and an client error code return value.

6.4.1. cticliDownloadFile

Download a file from the server working directory.

Prototype:

int cticliDownloadFile (int ch, char∗file, char∗outfile);

Parameters:

`in`	*ch*	client handle
`in`	*file*	file to be downloaded from the server working directory
`in`	*outfile*	local output file name

6.4.2. cticliGetServerStats

Get server statistics.

Prototype:

int cticliGetServerStats (int ch, int∗ntir, int∗nrdf, int∗nctilib, int∗nconnect, int∗nmaxclient, int∗ncticli, long

int∗nhtml);

Parameters:

in	*ch*	client handle
out	*ntir*	number of files in the server tire parameter file database
out	*nrdf*	number of files in the server road definition database
out	*nctilib*	number of files in the server CTI library database
out	*nconnect*	number of connections accepted since server startup
out	*nmaxclient*	peak value of clients connected at the same time
out	*ncticli*	total number of executed CTI commands
out	*nhtml*	total number of delivered administration pages

6.4.3. cticliInit

Initialize CTICLI.

Prototype:

int cticliInit (int∗ch, CTICLIINIT∗param);

Parameters:

out	*ch*	client handle
in	*param*	parameter structure oftypedef CTICLIINIT Note: Use CTICLIINIT_INITIALIZER to initalize this structure.

6.4.4. cticliListFiles

List files.

Prototype:

int cticliListFiles (int ch, int∗nfile, char∗∗file, int targetdir);

Parameters:

in	*ch*	client handle
out	*nfile*	number of list items returned
out	*file*	file list Note: Memory for file list is allocated internally. Caller must free memory when not needed any longer.

in	*targetdir*	targetdir value • CTICLI_TARGETDIR_WORKDIR: list files from working directory • CTICLI_TARGETDIR_DBTIR: list files from tire parameter database • CTICLI_TARGETDIR_DBRDF: list files from road parameter database • CTICLI_TARGETDIR_DBCTI: list files from cti parameter database • CTICLI_TARGETDIR_DBSUSP: list files from suspension parameter database • CTICLI_TARGETDIR_DBRIM: list files from rim parameter database

6.4.5. cticliLoadCtiLibrary

Load CTI library.

Prototype:

int cticliLoadCtiLibrary (int ch, int∗ier, char∗file, int targetdir);

Parameters:

in	*ch*	client handle
out	*ier*	exit status
in	*file*	File name. Meaning depends of*targetdir*. See*targetdir* parameter description for details.
in	*targetdir*	targetdir value • CTICLI_TARGETDIR_WORKDIR: Upload*filename* to working directory. Server database is not searched • CTICLI_TARGETDIR_DBCTI: search file in cti library database. The file is searched by the basename of the*filename* passed

6.4.6. cticliLoadRimData

Load rim data.

Prototype:

int cticliLoadRimData (int ch, int∗ier, char∗file, int targetdir);

Parameters:

in	*ch*	client handle
out	*ier*	exit status
in	*file*	File name. Meaning depends of*targetdir*. See*targetdir* parameter description for details.

in	*targetdir*	targetdir value • CTICLI_TARGETDIR_WORKDIR: Upload*filename* to working directory. Server database is not searched • CTICLI_TARGETDIR_DBRIM: search file in rim parameter database. The file is searched by the basename of the*filename* passed

6.4.7. cticliLoadRoadData

Load road data.

Prototype:

int cticliLoadRoadData (int ch, int∗ier, char∗file, int targetdir);

Parameters:

in	*ch*	client handle
out	*ier*	exit status
in	*file*	File name. Meaning depends of*targetdir*. See*targetdir* parameter description for details.
in	*targetdir*	targetdir value • CTICLI_TARGETDIR_WORKDIR: Upload*filename* to working directory. Server database is not searched • CTICLI_TARGETDIR_DBRDF: search file in road parameter database. The file is searched by the basename of the*filename* passed

6.4.8. cticliLoadSuspensionData

Load suspension data.

Prototype:

int cticliLoadSuspensionData (int ch, int∗ier, char∗file, int targetdir);

Parameters:

in	*ch*	client handle
out	*ier*	exit status
in	*file*	File name. Meaning depends of*targetdir*. See*targetdir* parameter description for details.

in	*targetdir*	targetdir value • CTICLI_TARGETDIR_WORKDIR: Upload*filename* to working directory. Server database is not searched • CTICLI_TARGETDIR_DBSUSP: search file in suspension parameter database. The file is searched by the basename of the*filename* passed

6.4.9. cticliLoadTireData

Load tire data.

Prototype:

int cticliLoadTireData (int ch, int∗ier, char∗file, int targetdir);

Parameters:

in	*ch*	client handle
out	*ier*	exit status
in	*file*	File name. Meaning depends of*targetdir*. See*targetdir* parameter description for details.
in	*targetdir*	targetdir value • CTICLI_TARGETDIR_WORKDIR: Upload*filename* to working directory. Server database is not searched • CTICLI_TARGETDIR_DBTIR: search file in tire parameter database. The file is searched by the basename of the*filename* passed

6.4.10. cticliUploadFile

Load CTI library.

Prototype:

int cticliUploadFile (int ch, char∗file, char∗outfile);

Parameters:

in	*ch*	client handle
in	*file*	file to be uploaded to the server working directory
in	*outfile*	remote output file name

6.4.11. cticliGetForcesListMT

Get forces for list of tire instances in multi-threading mode.

Prototype:

int cticliGetForcesListMT (int nthl, int thl[], int timeout, double fr0[][3], double trr0[][3], int∗ier);

Parameters:

in	*nthl*	number of tire instances
in	*thl*	list of tire handles
in	*timeout*	UDP only: receive timeout[µs]
out	*fr0*	forces acting on rim centers[N]
out	*trr0*	torques acting on rim centers[Nm]
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

6.4.12. cticliComputeForcesWithOutputArrayList

Compute forces and outputs for list of tire instances in multi-threaded mode.

Prototype:

void cticliComputeForcesWithOutputArrayList (int nthl, int thl[], double t, double r0r0[][3], double a0r[][9], double v0r0[][3], double w0r0[][3], int mode, double fr0[][3], double trr0[][3], int outmode, int outdim, double∗ out[], int∗ier);

Parameters:

in	*nthl*	number of tire instances
in	*thl*	list of tire handles
in	*t*	simulation time[s]
in	*r0r0*	rigid-body states of rims
in	*a0r*	rigid-body states of rims
in	*v0r0*	rigid-body states of rims
in	*w0r0*	rigid-body states of rims

in	*mode*	job control • ...0 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have not yet been accepted by external integrator • ...1 calculate (if not yet available) or return (if available) dynamic tire forces. Values of system states at actual simulation time t have been accepted by external integrator • ...2 unconditionally (re-)calculate tire forces • ...3 calculate steady-state tire forces • ...4 calculate static tire forces, avgd. road • ...5 calculate static tire forces, enhanced accuracy, avgd. road • ...6 calculate static tire forces, time-/loc-dependent road • ..1. compute steady states first, if not yet done (only in dynamic case) • .1.. extrapolate forces/moments to next time step (only in dynamic case), for use if calling solver is to run in parallel with CTI • 0... enable multi-threading • 1... disable multi-threading
out	*fr0*	forces acting on rim centers[N]
out	*trr0*	torques acting on rim centers[Nm]
in	*outmode*	output mode • 0 Using TYDEX-conform output signals. • 1 Using TYDEX-subset output signals. • 2 Using dSPACE output signals.
in	*outdim*	output dimension for every vector in out[]
out	*out*	output values (every vector in out[] needs memory for at least outdim values)
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted • 2 no license

6.4.13. cticliGetForcesWithOutputArrayListMT

Get forces and outputs for list of tire instances in multi-threading mode.

Prototype:

int cticliGetForcesWithOutputArrayListMT (int nthl, int thl[], int timeout, double fr0[][3], double trr0[][3], int outmode, int outdim, double* out[], int*ier);

Parameters:

in	*nthl*	number of tire instances
in	*thl*	list of tire handles
in	*timeout*	UDP only: receive timeout[µs]
out	*fr0*	forces acting on rim centers[N]
out	*trr0*	torques acting on rim centers[Nm]
in	*outmode*	output mode • 0 Using TYDEX-conform output signals. • 1 Using TYDEX-subset output signals. • 2 Using dSPACE output signals.
in	*outdim*	output dimension for every vector in out[]
out	*out*	output values (every vector in out[] needs memory for at least outdim values)
out	*ier*	error flag • 0 ok • 1 error occurred (error message was written to log). Simulation should be aborted

6.5. API Type Definition Reference

6.5.1. typedef CTICLICOMMTYPE

Communication enumeration fortypedef CTICLIINIT:

typedef enum cticli_commtype { CTICLI_COMMTYPE_TCP, CTICLI_COMMTYPE_UDP, } CTICLICOMMTYPE;

Enumerators:

CTICLI_COMMTYPE_TCP	Using TCP communication (default)
CTICLI_COMMTYPE_UDP	Using UDP communication

6.5.2. typedef CTICLIINIT

Parameter structure forcticliInit:

typedef struct { int major_version; CTICLIMSGFUNC message_func; char output_folder[256]; char output_prefix[64]; int calling_solver; int mt_call_flag; CTICLICOMMTYPE comm_type; int srv_port; char* srv_hostname; int tcp_time_out; int tcp_srv_sock; int udp_cli_port; int udp_std_time_out; int udp_rt_time_out; int udp_rt_time_out_overrun; int udp_rt_max_overrun;} CTICLIINIT;

Note: Use CTICLIINIT_INITIALIZER macro to initialize this structure.

Members:

major_version	cosin major version. Note: Do not change this value!
message_func	message output callback function oftypedef UMSGF. If specified, FTire will call this function to pass messages to the calling application.
output_folder	directory to save output files to
output_prefix	output file prefix
calling_solver	calling solver environment • 0: unknown (default)
mt_call_flag	multi-threaded call flag • 0: single-threaded (default) • 1: multi-threaded, mandatory if a function from(4) is called
comm_type	communication type oftypedef CTICLICOMMTYPE.
srv_port	server port
srv_hostname	server hostname
tcp_time_out	TCP only: socket time out
udp_cli_port	UDP only: client port
udp_std_time_out	UDP only: standard time-out (in micro seconds) for cticliUtilUdpRecv calls
udp_rt_time_out	UDP only: realtime time-out (in micro seconds) for cticliUtilUdpRecv calls
udp_rt_time_out_overrun	UDP only: realtime time-out (in micro seconds) for cticliUtilUdpRecv calls until udp_rt_max_overrun is reached
udp_rt_max_overrun	UDP only: maximum number of allowed overruns during realtime

A. Additional Tables

A.1. Supported Labels forctiFindOutputSignalNumber

```
"error code (0:last integration step is ok)"
"longitudinal slip [%]"
"slip angle [deg]"
"tire camber angle [deg]"
"forces in footprint, inertial frame [N] (1)"
"forces in footprint, inertial frame [N] (2)"
"forces in footprint, inertial frame [N] (3)"
"torques in footprint, inertial frame [Nm] (1)"
"torques in footprint, inertial frame [Nm] (2)"
"torques in footprint, inertial frame [Nm] (3)"
"forces in footprint, TYDEX C [N] (1)"
"forces in footprint, TYDEX C [N] (2)"
"forces in footprint, TYDEX C [N] (3)"
"torques in footprint, TYDEX C [Nm] (1)"
"torques in footprint, TYDEX C [Nm] (2)"
"torques in footprint, TYDEX C [Nm] (3)"
"forces in footprint, TYDEX W [N] (1)"
"forces in footprint, TYDEX W [N] (2)"
"forces in footprint, TYDEX W [N] (3)"
"torques in footprint, TYDEX W [Nm] (1)"
"torques in footprint, TYDEX W [Nm] (2)"
"torques in footprint, TYDEX W [Nm] (3)"
"forces in footprint, ISO contact frame [N] (1)"
"forces in footprint, ISO contact frame [N] (2)"
"forces in footprint, ISO contact frame [N] (3)"
"torques in footprint, ISO contact frame [Nm] (1)"
"torques in footprint, ISO contact frame [Nm] (2)"
"torques in footprint, ISO contact frame [Nm] (3)"
"forces on rim, TYDEX C [N] (1)"
"forces on rim, TYDEX C [N] (2)"
"forces on rim, TYDEX C [N] (3)"
"torques on rim, TYDEX C [Nm] (1)"
"torques on rim, TYDEX C [Nm] (2)"
"torques on rim, TYDEX C [Nm] (3)"
"forces on rim, TYDEX W [N] (1)"
"forces on rim, TYDEX W [N] (2)"
"forces on rim, TYDEX W [N] (3)"
"torques on rim, TYDEX W [Nm] (1)"
"torques on rim, TYDEX W [Nm] (2)"
"torques on rim, TYDEX W [Nm] (3)"
"forces on rim, TYDEX H [N] (1)"
"forces on rim, TYDEX H [N] (2)"
"forces on rim, TYDEX H [N] (3)"
"torques on rim, TYDEX H [Nm] (1)"
"torques on rim, TYDEX H [Nm] (2)"
"torques on rim, TYDEX H [Nm] (3)"
"forces on rim, ISO contact frame [N] (1)"
"forces on rim, ISO contact frame [N] (2)"
"forces on rim, ISO contact frame [N] (3)"
"torques on rim, ISO contact frame [Nm] (1)"
"torques on rim, ISO contact frame [Nm] (2)"
"torques on rim, ISO contact frame [Nm] (3)"
"forces on rim, inertial frame [Nm] (1)"
"forces on rim, inertial frame [Nm] (2)"
```

"forces on rim, inertial frame [Nm] (3)"
"torques on rim, inertial frame [Nm] (1)"
"torques on rim, inertial frame [Nm] (2)"
"torques on rim, inertial frame [Nm] (3)"
"forces on cleat, inertial frame [Nm] (1)"
"forces on cleat, inertial frame [Nm] (2)"
"forces on cleat, inertial frame [Nm] (3)"
"forces in cont. bodies, inertial frame [N] (1)"
"forces in cont. bodies, inertial frame [N] (2)"
"forces in cont. bodies, inertial frame [N] (3)"
"forces in cont. bodies, inertial frame [N] (4)"
"forces in cont. bodies, inertial frame [N] (5)"
"forces in cont. bodies, inertial frame [N] (6)"
"forces in cont. bodies, inertial frame [N] (7)"
"forces in cont. bodies, inertial frame [N] (8)"
"forces in cont. bodies, inertial frame [N] (9)"
"forces in cont. bodies, inertial frame [N] (10)"
"forces in cont. bodies, inertial frame [N] (11)"
"forces in cont. bodies, inertial frame [N] (12)"
"forces in cont. bodies, inertial frame [N] (13)"
"forces in cont. bodies, inertial frame [N] (14)"
"forces in cont. bodies, inertial frame [N] (15)"
"forces in cont. bodies, inertial frame [N] (16)"
"forces in cont. bodies, inertial frame [N] (17)"
"forces in cont. bodies, inertial frame [N] (18)"
"forces in cont. bodies, inertial frame [N] (19)"
"forces in cont. bodies, inertial frame [N] (20)"
"forces in cont. bodies, inertial frame [N] (21)"
"forces in cont. bodies, inertial frame [N] (22)"
"forces in cont. bodies, inertial frame [N] (23)"
"forces in cont. bodies, inertial frame [N] (24)"
"forces in cont. bodies, inertial frame [N] (25)"
"forces in cont. bodies, inertial frame [N] (26)"
"forces in cont. bodies, inertial frame [N] (27)"
"forces in cont. bodies, inertial frame [N] (28)"
"forces in cont. bodies, inertial frame [N] (29)"
"forces in cont. bodies, inertial frame [N] (30)"
"torques in cont. bodies, inertial frame [Nm] (1)"
"torques in cont. bodies, inertial frame [Nm] (2)"
"torques in cont. bodies, inertial frame [Nm] (3)"
"torques in cont. bodies, inertial frame [Nm] (4)"
"torques in cont. bodies, inertial frame [Nm] (5)"
"torques in cont. bodies, inertial frame [Nm] (6)"
"torques in cont. bodies, inertial frame [Nm] (7)"
"torques in cont. bodies, inertial frame [Nm] (8)"
"torques in cont. bodies, inertial frame [Nm] (9)"
"torques in cont. bodies, inertial frame [Nm] (10)"
"torques in cont. bodies, inertial frame [Nm] (11)"
"torques in cont. bodies, inertial frame [Nm] (12)"
"torques in cont. bodies, inertial frame [Nm] (13)"
"torques in cont. bodies, inertial frame [Nm] (14)"
"torques in cont. bodies, inertial frame [Nm] (15)"
"torques in cont. bodies, inertial frame [Nm] (16)"
"torques in cont. bodies, inertial frame [Nm] (17)"
"torques in cont. bodies, inertial frame [Nm] (18)"
"torques in cont. bodies, inertial frame [Nm] (19)"
"torques in cont. bodies, inertial frame [Nm] (20)"
"torques in cont. bodies, inertial frame [Nm] (21)"
"torques in cont. bodies, inertial frame [Nm] (22)"

"torques in cont. bodies, inertial frame [Nm] (23)"
"torques in cont. bodies, inertial frame [Nm] (24)"
"torques in cont. bodies, inertial frame [Nm] (25)"
"torques in cont. bodies, inertial frame [Nm] (26)"
"torques in cont. bodies, inertial frame [Nm] (27)"
"torques in cont. bodies, inertial frame [Nm] (28)"
"torques in cont. bodies, inertial frame [Nm] (29)"
"torques in cont. bodies, inertial frame [Nm] (30)"
"mean distance rim center - road [mm]"
"mean absolute height footprint [m]"
"approximate footprint area [m^2]"
"accurate length of footprint [mm]"
"accurate width of footprint [mm]"
"minimum interior cross section area [m^2]"
"air volume [m^3]"
"air pressure [bar]"
"air temperature [degC]"
"mean tread temperature [degC]"
"mean contact patch temperature [degC]"
"mean contact patch temperature near zenith [degC]"
"actual relative belt extension [%]"
"maximum ground pressure in footprint [MPa]"
"mean ground pressure in footprint [MPa]"
"ground pressure std .dev. in footprint [MPa]"
"mean longit. long-waved curv. of road prof. [1/m]"
"global tire deflection [mm]"
"global tire deflection velocity [m/s]"
"loaded radius [mm]"
"maximum radial belt element displacement [mm]"
"belt-to-rim torsion about wheel spin axis [deg]"
"actual pneumatic trail [mm]"
"actual pneumatic scrub [mm]"
"approx. footprint center in global coord. [m] (1)"
"approx. footprint center in global coord. [m] (2)"
"approx. footprint center in global coord. [m] (3)"
"assumed footprint center in global coord. [m] (1)"
"assumed footprint center in global coord. [m] (2)"
"assumed footprint center in global coord. [m] (3)"
"approx. fp center in road-fixed coord. [m] (1)"
"approx. fp center in road-fixed coord. [m] (2)"
"approx. fp center in road-fixed coord. [m] (3)"
"road friction factor in approx. fp. center [-]"
"gyroscopic overturning moment [Nm]"
"gyroscopic aligning moment [Nm]"
"longitudinal rim center velocity [m/s]"
"lateral rim center velocity [m/s]"
"vertical rim center velocity [m/s]"
"longitudinal slip velocity at contact point [m/s]"
"lateral slip velocity at contact point [m/s]"
"contact patch bore velocity [rad/s]"
"footprint area cg, expr. in contact frame [mm] (1)"
"footprint area cg, expr. in contact frame [mm] (2)"
"footprint area cg, expr. in contact frame [mm] (3)"
"x-shift of belt above footprint [mm]"
"y-shift of belt above footprint [mm]"
"torsion of belt above footprint [deg]"
"bending of belt above footprint [1/m]"
"long. geometrical shift of footprint [mm]"
"flag whether rim contacts / penetrates road [-]"

"transf. m. to wheel frame (TYDEX C) (1)"
"transf. m. to wheel frame (TYDEX C) (2)"
"transf. m. to wheel frame (TYDEX C) (3)"
"transf. m. to wheel frame (TYDEX C) (4)"
"transf. m. to wheel frame (TYDEX C) (5)"
"transf. m. to wheel frame (TYDEX C) (6)"
"transf. m. to wheel frame (TYDEX C) (7)"
"transf. m. to wheel frame (TYDEX C) (8)"
"transf. m. to wheel frame (TYDEX C) (9)"
"transf. m. to road contact frame (TYDEX W) (1)"
"transf. m. to road contact frame (TYDEX W) (2)"
"transf. m. to road contact frame (TYDEX W) (3)"
"transf. m. to road contact frame (TYDEX W) (4)"
"transf. m. to road contact frame (TYDEX W) (5)"
"transf. m. to road contact frame (TYDEX W) (6)"
"transf. m. to road contact frame (TYDEX W) (7)"
"transf. m. to road contact frame (TYDEX W) (8)"
"transf. m. to road contact frame (TYDEX W) (9)"
"transf. m. to ISO 8855 axis system (1)"
"transf. m. to ISO 8855 axis system (2)"
"transf. m. to ISO 8855 axis system (3)"
"transf. m. to ISO 8855 axis system (4)"
"transf. m. to ISO 8855 axis system (5)"
"transf. m. to ISO 8855 axis system (6)"
"transf. m. to ISO 8855 axis system (7)"
"transf. m. to ISO 8855 axis system (8)"
"transf. m. to ISO 8855 axis system (9)"
"longitudinal rim displacement [mm]"
"lateral rim displacement [mm]"
"rim toe angle [deg]"
"rim angular accel. about spin axis [deg/s^2]"
"rim transl. acceleration [m/s^2] (1)"
"rim transl. acceleration [m/s^2] (2)"
"rim transl. acceleration [m/s^2] (3)"
"rim angular acceleration [rad/s^2] (1)"
"rim angular acceleration [rad/s^2] (2)"
"rim angular acceleration [rad/s^2] (3)"
"road motion states [-] (1)"
"road motion states [-] (2)"
"road motion states [-] (3)"
"road motion states [-] (4)"
"road motion states [-] (5)"
"road motion states [-] (6)"
"road motion states [-] (7)"
"road motion states [-] (8)"
"road motion states [-] (9)"
"road motion states [-] (10)"
"road motion states [-] (11)"
"road motion states [-] (12)"
"force/moment standard deviations [N],[Nm] (1)"
"force/moment standard deviations [N],[Nm] (2)"
"force/moment standard deviations [N],[Nm] (3)"
"force/moment standard deviations [N],[Nm] (4)"
"force/moment standard deviations [N],[Nm] (5)"
"force/moment standard deviations [N],[Nm] (6)"
"max. normal tread block deflection [mm]"
"total power loss in tread [kW]"
"total power loss in plies [kW]"
"total power loss by friction [kW]"

"total power loss [kW]"
"rolling loss [N]"
"actual estimated dynamic rolling radius [mm]"
"total ply-steer moment [Nm]"
"maximum sliding velocity in footprint [m/s]"
"mean sliding velocity in footprint [m/s]"
"sliding velocity std. dev. in footprint [m/s]"
"maximum belt-to-rim contact intrusion [mm]"
"maximum side-wall-to-road intrusion [mm]"
"maximum rim-to-road intrusion [mm]"
"maximum side-wall stretching defl. left side [mm]"
"maximum side-wall stretching defl. right side [mm]"
"rim-flange-to-road contact force [N] (1)"
"rim-flange-to-road contact force [N] (2)"
"rim-flange-to-road contact force [N] (3)"
"max. elastic left rim flange rad. displ. [mm]"
"max. elastic right rim flange rad. displ. [mm]"
"max. plastic left rim flange rad. def. [mm]"
"max. plastic right rim flange rad. def. [mm]"
"max. elastic left rim flange lat. displ. [mm]"
"max. elastic right rim flange lat. displ. [mm]"
"max. plastic left rim flange lat. def. [mm]"
"max. plastic right rim flange lat. def. [mm]"
"gas-vibration-induced rim force (p) [N] (1)"
"gas-vibration-induced rim force (p) [N] (2)"
"gas-vibration-induced rim force (p) [N] (3)"
"gas-vibration-induced belt force (p) [N] (1)"
"gas-vibration-induced belt force (p) [N] (2)"
"gas-vibration-induced belt force (p) [N] (3)"
"gas-vibration-induced rim force (v) [N] (1)"
"gas-vibration-induced rim force (v) [N] (2)"
"gas-vibration-induced rim force (v) [N] (3)"
"mean circumferential filling gas velocity [m/s]"
"circumferential filling gas vel. variation [m/s]"
"filling gas pressure variation [bar]"
"TPMS sensor signal pressure [bar]"
"TPMS sensor signal temperature [degC]"
"TPMS sensor-fixed transl. accel. [m/s^2] (1)"
"TPMS sensor-fixed transl. accel. [m/s^2] (2)"
"TPMS sensor-fixed transl. accel. [m/s^2] (3)"
"TPMS sensor-fixed rot. accel. [rad/s^2] (1)"
"TPMS sensor-fixed rot. accel. [rad/s^2] (2)"
"TPMS sensor-fixed rot. accel. [rad/s^2] (3)"
"TPMS location curvature radius longitudinal [mm]"
"TPMS location curvature radius lateral [mm]"
"TPMS sensor distance to wheel center [mm]"
"min z component [m]"
"min required contact processor bound [%]"
"relative circumf. coord. of lowest segment [-]"
"cleat contact phase [-]"
"act. total tire mass geom. [m],[kg],[kgm^2] (1)"
"act. total tire mass geom. [m],[kg],[kgm^2] (2)"
"act. total tire mass geom. [m],[kg],[kgm^2] (3)"
"act. total tire mass geom. [m],[kg],[kgm^2] (4)"
"act. total tire mass geom. [m],[kg],[kgm^2] (5)"
"act. total tire mass geom. [m],[kg],[kgm^2] (6)"
"act. total tire mass geom. [m],[kg],[kgm^2] (7)"
"act. total tire mass geom. [m],[kg],[kgm^2] (8)"
"act. total tire mass geom. [m],[kg],[kgm^2] (9)"

```
"act. total tire mass geom. [m],[kg],[kgm^2] (10)"
"perc. belt mass variation at lowest segm. [%]"
"weight 1st press. value in friction char. [-]"
"weight 2nd press. value in friction char. [-]"
"weight 3rd press. value in friction char. [-]"
"weight sticking velocity in friction char. [-]"
"weight max. frict. velocity in friction char. [-]"
"weight sliding velocity in friction char. [-]"
"weight blocking velocity in friction char. [-]"
"total RTF factor, w/o communication [-]"
"first partial RTF factor [-]"
"second partial RTF factor [-]"
"third partial RTF factor [-]"
"communication overhead  RTF factor [-]"
"relative size of sliding area [%]"
"tire squealing amplitude factor [-]"
"actual rim-to-road contact vertical stiffness [N/m]"
"total contact patch shear stiffness [N/m]"
"total contact patch shear damping [Ns/m]"
"number of contact patch boundary nodes [-]"
"number of tread blocks with contact [-]"
"number of tread blocks close to contact [-]"
"tread pattern loc. of first cleat cont. [-] (1)"
"tread pattern loc. of first cleat cont. [-] (2)"
"road type [-]"
"road-type specific extra output [div] (1)"
"road-type specific extra output [div] (2)"
"road-type specific extra output [div] (3)"
"road-type specific extra output [div] (4)"
"road-type specific extra output [div] (5)"
"road-type specific extra output [div] (6)"
"road-type specific extra output [div] (7)"
"road-type specific extra output [div] (8)"
"road-type specific extra output [div] (9)"
"road-type specific extra output [div] (10)"
```

A.2. Subset of TYDEX Output Signals

Following a subset of TYDEX output signals is listed, containing all TYDEX signals defined in ftire_model_pdf (chapter 10) without the signals marked either „not used“ or „do not use“.

Note: This subset is numbered consecutively and uses, due to compression, different numbers as the TYDEX-conform output signals. For example, tire deflection is signal 28 in this subset and 44 in the TYDEX-conform complete list. Moreover, certain signals like slip values and ISO forces appear more than once, since the TYDEX list of signals is not cleanly standardized, and different applications expect these signals in different locations of the output array.

signal	***description***	***unit***
1-6	tire forces and torques, expressed in TYDEX W-axis	*N, Nm*
7	slip angle	*rad*
8	longitudinal slip	*m*
9	camber angle	*rad*
10-12	geometrical center of contact patch in global co-ordinates	*m*
13-21	3x3 transformation matrix from contact tangential plane to global coordinates (storage column-wise)	-

Document Revision:2018-4-r18799

22-27	tire forces and torques, expressed in ISO axis system	*N,Nm*
28	tire deflection	*m*
29	vertical rim center velocity	*m/s*
30	longitudinal slip velocity at contact point	*m/s*
31	lateral slip velocity at contact point	*m/s*
32	longitudinal rim center velocity	*m/s*
33	maximum belt radius after inflation	*m*
34	rim angular velocity relative to wheel carrier (ABS signal)	*rad/s*
35	mean road friction factor in contact patch	-
36	tire deflection	*mm*
37	slip angle	*deg*
38	longitudinal slip	*%*
39-44	tire forces and torques, expressed in TYDEX C-axis	*N,Nm*
45	rolling loss	*N*
46	maximum belt-to-rim contact intrusion	*m*
47	dynamic rolling radius	*m*
48-53	tire forces and torques, expressed in ISO axis	*N,Nm*

Index