UNIVERSAL MECHANISM 9

User's manual

UM COM Interfaces

2020

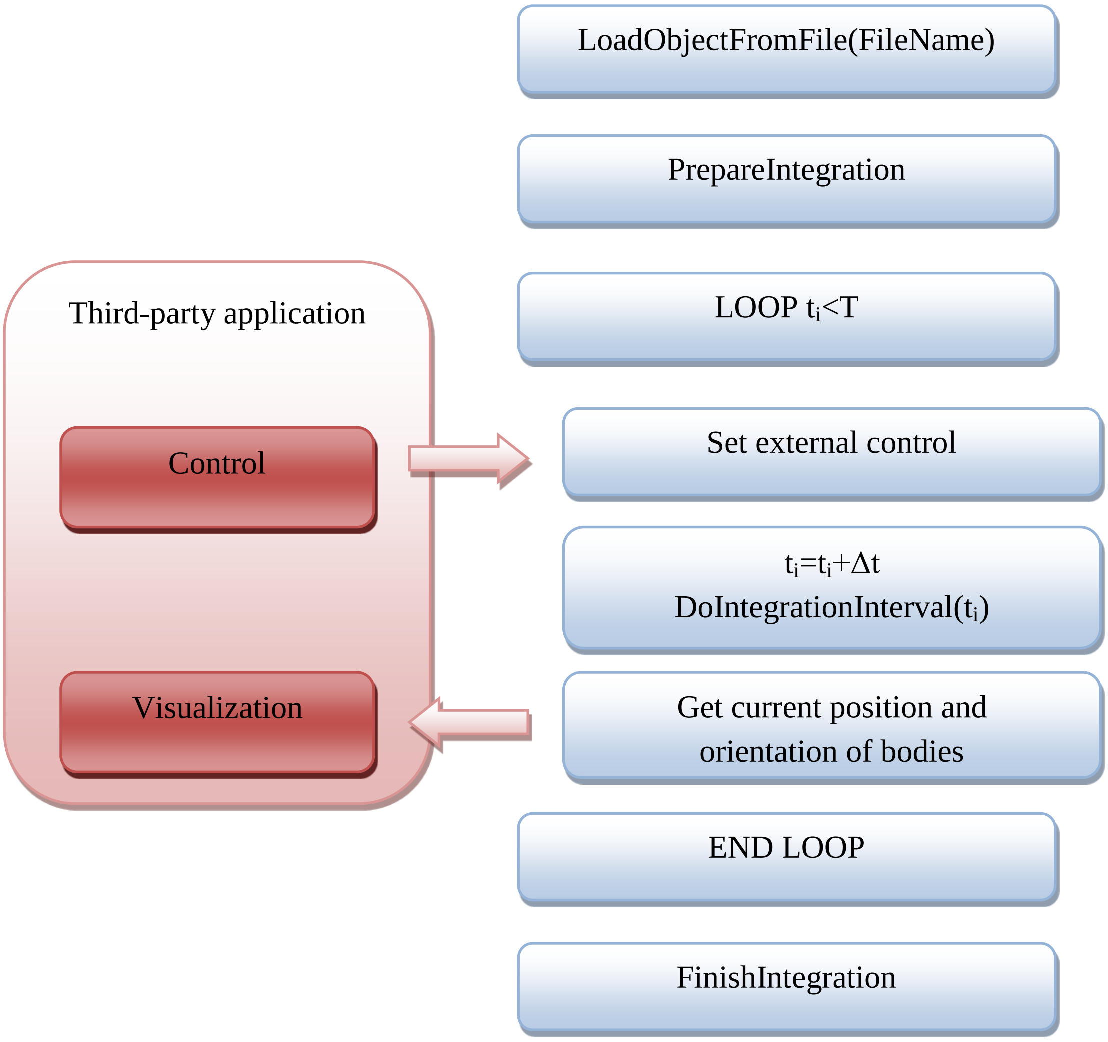
Contents

20. UM COM Interfaces

20.1. Introduction

Universal Mechanism software includes a COM server that can be used in third-party applications. Implemented COM interfaces allows a user to load UM models prepared in advance with the help of **UM Input** preprocessor or other preprocessor, change parameters of the model, set up all simulation environment, simulate dynamics of the model and get kinematical data for external visualization and control.

Typical way of usage of UM COM interfaces is given in the picture below. All mentioned there methods are related to *IUMObject* interface. Implemented interfaces are intended for supporting high-quality dynamics in applications of third-party developers. It is supposed that the third-party application provides processing of external control and visualization of the current model configuration.



Please note, COM object must be registered before exploiting. Use utility regsrv32.dll for the registration. Example of command line:

C:\Windows\System32\regsvr32.exe C:\Program Files\UM Software Lab\UM\9.0\bin\umcomsolver.dll

20.2. IUMObject interface

IUMObject is a basic interface. Work with UM COM interfaces starts exactly with creating and using object of *IUMObject* type. Let us consider methods of *IUMObject*.

Interface: *IUMObject*

Hierarchy: *IUnknown – IBasicElement – IUMObject*

Methods	Description
AddWindow	HRESULT _stdcall AddWindow([in] long WindowType, [in] void * Window); Adds new UM-style animation or graphical window depending on WindowType: 0 means animation window; 1 means graphical window. Window is a pointer to created window, should be considered as IComAnimationWindow or IUMGraphicWindow correspondingly.
DoIntegrationInterval	int _stdcall DoIntegrationInterval([in] double TFinish); Simulates dynamics of the model till TFinish seconds. After each time-step new positions and orientations of bodies and all other performances of mechanical system are available. For real-time simulation usually called on timer. To obtain smooth animation (25-80 Hz) time increment should be correspondent (0.04 – 0.0125 s). Output: 0 –simulation interval is correct; 1 – simulation interval is over; 2 – simulation was not prepared; otherwise – simulation interval fails, see *GetLastError* method for comments.
FinishIntegration	int _stdcall FinishIntegration(void); Frees memory and initializes data after calling DoIntegrationInterval.
GetTrain	HRESULT _stdcall GetTrain([out] void * aTrain); Returns IUMComTrain interface if any or NULL if there is no train model loaded.
GetCar	HRESULT _stdcall GetCar([out] void * aTrain); Returns IComCar interface if any or NULL if there is no road vehicle model loaded.
GetLastError	LPSTR _stdcall GetLastError(void); Returns comments to the last errors

LoadObjectFromFile
LoadObjectFromFileW
	int _stdcall LoadObjectFromFile([in] LPSTR FileName);
	int _stdcall LoadObjectFromFile([in] LPWSTR FileName);
	
	Loads UM model from specified input.dat file. Returns 0 in suc-

	cessful termination, non-zero result in case of non-successful termination.
PrepareIntegration	int _stdcall PrepareIntegration(void); Allocates memory and initializes data prior to calling DoIntegrationInterval. If railroad is used for the description of track geometry a pre-simulation of the start mode stage is fulfilled in background mode. Output: 0 –simulation interval is correct; 1 – internal error; otherwise – simulation preparing fails, see *GetLastError* method for comments.
GetRailRoad	HRESULT _stdcall GetRailRoad([out] void * aRailRoad); Returns IRailRoad interface if any or NULL if there is no train model loaded.
GetLastTime	HRESULT _stdcall GetLastTime([out] double * LastTime); Returns model time [seconds]
GetSystemID	LPSTR _stdcall GetSystemID(void); Returns SystemID of user PC. Used in licensing system.
IsLicensed	VARIANT_BOOL _stdcall IsLicensed(void); Check if UMCom.dll is registered or not. Used in licensing system.
SetLicense	HRESULT _stdcall SetLicense([in] LPSTR aLicenseData, [in] LPSTR aLicenseKey, [in] LPSTR aSystemID); Enter license data for UMCom.dll. Used in licensing system.

GetElementByNameEx
GetElementByNameExW
	HRESULT _stdcall GetElementByNameEx([in] long ElementType, [in] LPSTR ElementName, [out] void* Element);
	
	HRESULT _stdcall GetElementByNameEx([in] long ElementType, [in] LPWSTR ElementName, [out] void* Element);
	
	Input: ElementType – type of interface (body, joint, identifier etc.)
	eltBody = 1;
	eltJoint = 2;
	eltSubsystem =3;
	eltBFrc = 4;
	eltLFrc = 5;
	eltCFrc = 6;
	eltAFrc = 7;
	eltSFrc = 8;
	eltGO = 11;
	eltIdentifier = 12;
	ElementName : name of element

	Output: Element is interface to the element.

20.2.1. Run, pause and stop

Here we consider the simplest way to pause and stop simulation with UMCOMSolver.

Three buttons on the form of class TTrainComForm are used: btnRun, btnPause, btnStop. Below the service procedures are given for start, pause and stop simulation.

```
var
    SimulationRun : boolean;
    PauseMode : boolean;
    UMObject : IUMObject;

procedure TTrainComForm.btnRunClick(Sender: TObject);
var T : double;
    dT : double;
begin
  T:=0;
  dT:=0.025;
  SimulationRun:=true;
  PauseMode:=false;
  If UMObject.PrepareIntegration = 0 then begin
    GetLastTime(T);
// if railroad is used for track geometry description only
    while SimulationRun do begin
      T:=T+DT;
      If UMObject.DoIntegrationInterval(T) <> 0 then
         break;
      if PauseMode then DoPause;
    end;
    UMObject.FinishIntegration;
  end;
end;

procedure TTrainComForm.btnPauseClick(Sender: TObject);
begin
  PauseMode:=true;
end;

procedure TTrainComForm.btnStopClick(Sender: TObject);
begin
  SimulationRun:=false;
end;

procedure TTrainComForm.DoPause;
begin
  if MessageDlg('Pause. To continue click OK button. To stop click Cancel
button.',
    mtInformation, [mbOK, mbCancel],0) = mrCancel then SimulationRun:=false;
  PauseMode:=false;
end;
```

20.2.2. Recommended solver initialization

Before start train simulation, it is recommended to set the following parameters of solver:

```
UMObject–>SetSolver(5);
UMObject–>SetJacobianComputation(1);
```

```
UMObject–>SetBlockDiagonalJacobians(0);
UMObject–>SetSolverAccuracy(1.0e-6);
UMObject–>SetMinimalStep(0.005, 5, 20);
```

20.2.3. Getting model data

User can use GetElementByNameEx(W) function for getting data about bodies, joints and identificators included in the model, see description above.

20.3. Interface for generation of equations

This interface allows the user to generate equations of motion of an UM Object in symbolic form. Equations are generated in Pascal, and should be compiled by an external Delphi compiler. Symbolic equations requires less floating point operations for evaluations of mass matrix and inertia forces of MBS with many degrees of freedom, and makes simulation process faster than in case of numeric-iterative generation. Sometimes this acceleration could be critical for real-time simulations.

The following files are required for compiling equations:

- dcc32.exe (a stand along Borland Delphi compiler),
- standard Delphi *.dcu files from Lib directory (e.g. windows.dcu, comctrls.dcu end so on),
- some UM service *.pas files located in the COM directory.

Equations can be generated according to one of the algorithm: the direct or the composite body algorithms. The second one is more derives more efficient code for object with long kinematic chains. The following constants specify the method:

gmDirectMethod = 0;
gmCompositeMethod = 2;

If the user set the method index, which differs from the above ones, UM chooses the optimal method automatically.

Interface: *IUMEquations*

Hierarchy: *IUnknown – IUMEquations*

Methods	Description
LoadUMObjectFromFile	HRESULT _stdcall LoadUMObjectFromFile([in] LPSTR FileName); Reads an UM object for which the equations must be derived. Input: FileName is the full path to the input.dat file with model. Output: 0 is the object is loaded, 1 if the loading fails.
CanGenerateEquations	HRESULT _stdcall CanGenerateEquations(void); Verifies whether the description of the loaded UM object is full and correct. Output: 0 is the object is correct, 1 if the object is not correct, and equations cannot be generated.

SetDelphiCompilerPath	HRESULT _stdcall SetDelphiCompilerPath([in] LPSTR DCC32Path, [in] LPSTR DCULibPath, [in] LPSTR ComPasPath) Specifies full paths to files, which are necessary for the compiling process Input: DCC32Path is the path to dcc32.exe compiler DCULibPath is the path to the standard Delphi *.dcu files ComPasPath is the path to the UM service *.pas files located in the COM directory. COM directory should not be included in the path. Output : 0 of all the paths are correct, 1 otherwise
GenerateEquations	HRESULT _stdcall GenerateEquations([in] long Method, [in] VARIANT_BOOL Compile) Generates equations for a loaded correct UM object. Input: Method is the index of algorithms Compile: 1 if equations should be compiled, 0 otherwise Output: 0 if equations are generated (and compiled if Compile = 1) successfully and 1 otherwise.

Example

```
procedure TTrainComForm.btnGenerateEquationsClick(Sender: TObject);
var UMEquations : IUMEquations;
begin
  UMEquations:=CreateComObject(CLASS_UMEquations) as IUMEquations;
  if UMEquations.SetDelphiCompilerPath(PChar(TrainSimPath+'\bin\dcc'),
           PChar(TrainSimPath+'\bin\dcc'),PChar(TrainSimPath+'\bin')) = 0
  then begin
    OpenDialog.InitialDir:= TrainSimPath + '\models\';
    OpenDialog.Filter:= 'Dat files|*.dat';
    if OpenDialog.Execute then
     if UMEquations.LoadUMObjectFromFile(PChar(OpenDialog.FileName)) = 0 then
      begin
        if (UMEquations.CanGenerateEquations=0) then
          UMEquations.GenerateEquations(-1, true);
      end;
  end;
  UMEquations._Release;
end;
```

20.4. IComIdentifier Interface

This interface is used for work with variables in current model.

Interface: *IComIdentifier*

Hierarchy: *IUnknown – IComIdentifier*

Methods	Description
GetValue	double _stdcall GetValue(void); Output: value of the variable.
SetValue	HRESULT _stdcall SetValue([in] double Value); Input: value of the variable.
SetAssignToAll	HRESULT _stdcall SetAssignToAll([in] long Value); This function sets flag which indicates to assign a new value of the variable to others same ones in subsystems or not. Input: 0 – false, 1 – true.
ShouldRefreshElements	HRESULT _stdcall ShouldRefreshElements([in] long Value); This function sets flag which indicates to refresh elements (joints, forces, etc.) where the identifier is used. Input: 0 – false, 1 – true.

Example

```
…
P: Pointer;
Omega: IComIdentifier;
UMObject: IUMObject;
…
// Name of variable is a long name. It is assembled from owner-subsystem name
// and short (simple) name of the variable divided by dot.
UMObject.GetElementByNameEx(eltIdentifier,
        PChar('Subsystem1Name.omega_rotor'), P);
Omega := IUnknown(P) as IComIdentifier;
Omega.SetAssignToAll(1);
Omega.ShouldRefreshElements(1);
…
Omega.SetValue(1);
…
lOmega := Omega.GetValue(1);
…
```

20.5. Interfaces for development of train models

20.5.1. IInpTrain interface

The interface is used for development of train models consisting of any number of 1D and 3D vehicle models.

Interface: *IInpTrain*

Hierarchy: *IUnknown – IBasicElement – IUMObject – IInpTrain*

Methods	Description
Add3DVehicle	HRESULT _stdcall Add3DVehicle([in] LPSTR Path, [in] long PositionInTrain, [out] void * Car); Adds a 3D rail vehicle to the train model. Input: Path – full path to input.dat file of a 3D model; in our case: to the 3D locomotive model/ Example: …\rw\train\3Dmodels\ LocoE43000_for_train\input.dat PositionInTrain: position of the vehicle in train (starts with 1). Output: Car: interface IComCar3D.
AddCarByIndex	HRESULT _stdcall AddCarByIndex([in] long Index, [out] void * Car); Adds a 1D car to the train model. Input: Index: index of a component 0.. ComponentCount-1 Output: Car: interface IComCar1D.
AddCarByName	HRESULT _stdcall AddCarByName([in] LPSTR CarName, [out] void * Car); Adds a 1D car to the train model. Input: CarName – name of a car model from Cars directory of database. Recommended value: ‘car’ Output: Car: interface IComCar1D.
AddLocomotiveByIndex	HRESULT _stdcall AddLocomotiveByIndex([in] long Index, [out] void * Loco); Adds a 1D locomotive to the train model. Input: Index = 0.. ComponentCount-1– index of a car model from Locomotives directory of database Output: Loco: interface IComLoco1D.
AddLocomotiveByName	HRESULT _stdcall AddLocomotiveByName([in] LPSTR LocoName, [out] void * Loco); Adds a 1D locomotive to the train model. Input: LocoName – name of a car model from Locomotives directory of database. Output: Loco: interface IComLoco1D.
ComponentCount	long _stdcall ComponentCount([in] long ComponentType); Input: ComponentType – type of a train component, Sect. 20.5.1.1. *"Types of train components"*, p. 20-13.

	Output: Number of components in database. 0 if TrainDataPath is incorrect.
ComponentName	LPSTR _stdcall ComponentName([in] long ComponentType, [in] long Index); Input: ComponentType – type of a train component Index: index of a component 0.. ComponentCount-1 Output: name of the component in database.
GetComponentIndexByName	long _stdcall GetComponentIndexByName([in] long ComponentType, [in] LPSTR ComponentName); Input: ComponentType – type of a train component ComponentName: name of a component; Output: index of a component 0..ComponentCount-1 if succeed, -1 if fails.
GetTrainDataPath	LPSTR _stdcall GetTrainDataPath(void); Output: current value of path to Train database ..\rw\train
SaveAs	HRESULT _stdcall SaveAs([in] LPSTR Path); Saves the ready model according to the Path. Example: d:\TrainSimulator\models\Train30 The train model will be saved in Train30 directory. Models of 3D vehicles are copied in this directory as well.
SetCurveResistanceFactor	HRESULT _stdcall SetCurveResistanceFactor([in] double Value); Value of a parameter A, characterizing resistance forces in curves, Sect. 20.5.1.2. *"Model of resistance force in curve"*, p. 20-14.
SetTrainDataPath	HRESULT _stdcall SetTrainDataPath([in] LPSTR Path); Input: path to train database ..\rw\train; is used if the default path is incorrect.
SetHoldingBrakeParameters	HRESULT _stdcall SetHoldingBrakeParameters([in] double aHoldingBrakeDemand, [in] double aHoldingVelocity, [in] VARIANT_BOOL aHoldingBrakeEnabled, [in] VARIANT_BOOL aHoldingBrakeControlsBP); Sets holding brake parameters. Input: aHoldingBrakeDemand – minimal brake demand for holding brake; aHoldingVelocity – maximal velocity for holding brake, m/s; aHoldingBrakeEnabled – enables holding brakes; aHoldingBrakeControlsBP – enables brake pipe changing for holding brake demand according to the brake system characteristics.

20.5.1.1. Types of train components

The following constants specify types of train components in database:

1. **tcCar** = 0;

   1D car model;

   Path: {TrainDataPath}\Cars
2. **tcLocomotive** = 1;

   1D locomotive model

   Path: {TrainDataPath}\Locomotives
3. **tcDraftGear** = 2;

   Model of draft gear

   Path: {TrainDataPath}\Draftgears
4. **tcBrakeCoefFriction** = 3;

   Model of brake coefficient of friction

   Path: {TrainDataPath}\Brakes\Coefs
5. **tcResistance** = 4;

   Model of vehicle resistance by run in tangent sections.

   Path: {TrainDataPath}\Resistance
6. **tcAcceleratingChamber** = 5;

   Model of accelerating chamber

   Path: {TrainDataPath}\ Brakes\Accelerating Chambers
7. **tcAuxiliaryReservoir** = 6;

   Model of auxiliary reservoir

   Path: {TrainDataPath}\ Brakes\Auxiliary Reservoirs
8. **tcBrakeCylinder** = 7;

   Model of brake cylinder

   Path: {TrainDataPath}\ Brakes\Brake Cylinders
9. **tcBrakeValve** = 8;

   Model of brake valve

   Path: {TrainDataPath}\ Brakes\Brake Valves
10. **tcCompressor** = 9;

    Model of compressor

    Path: {TrainDataPath}\ Brakes\Compressors
11. **tcControlValve** = 10;

    Model of control valve

    Path: {TrainDataPath}\ Brakes\Control Valves
12. **tcTractionMotor** = 11;

    Model of traction motor

    Path: {TrainDataPath}\ TractionMotors
13. **tcBrakeEquipment**= 12;

    Model of brake equipment

    Path: {TrainDataPath}\ Brakes\Forces
14. **tcAuxBrakeValve**= 13;

    Model of locomotive brake valves

Path: {TrainDataPath}\ Brakes\Loco brake valves

20.5.1.2. Model of resistance force in curve

The following model of resistance force in curves for 1 ton of vehicle mass is implemented:

$$F_r = \frac{A}{R}$$

where R is the curve radius, and A (Nm) is an empirical parameter.

Examples

Russian standards: A=7000 Nm;

Handbook of Railway Vehicle Dynamics: A=6116 Nm.

20.5.2. IComCar1D interface

IComCar1D is used for setting vehicle parameters and subsystems by development of a train model.

Interface: *IComCar1D*

Hierarchy: *IUnknown – IComCar1D*

Methods	Description
GetCouplingBase	double _stdcall GetCouplingBase(void); Output: coupling base of a car (m)
GetCGZPosition	double _stdcall GetCGZPosition(void); Ouput: height of the center of mass of the vehicle above rail head (m). This value is used for calculation of the overturning factor, see IUMComTrainVehicle.GetOverturningFactor method, Sect. 20.6.5. *"IUMComTrainVehicle interface"*, p. 20-38.
GetMass	double _stdcall GetMass(void); Output: mass of a car (kg)
GetPivotBase	double _stdcall GetPivotBase(void); Output: pivot base a car (m)
SetAcceleratingChamberByIndex	HRESULT _stdcall SetAcceleratingChamberByIndex([in] int Index); Input: Index=0..IInpTrain–>ComponentCount-1 – model of Accelerating Chamber
SetAcceleratingChamberByName	HRESULT _stdcall SetAcceleratingChamberByName([in] LPSTR Name); Input: Name of file with model of Accelerating Chamber
SetAuxiliaryReservoirByIndex	HRESULT _stdcall SetAuxiliaryReservoirByIndex([in] int Index); Input: Index=0..IInpTrain–>ComponentCount-1 – model of Auxiliary Reservoir
SetAuxiliaryReservoirByName	HRESULT _stdcall SetAuxiliaryReservoirByName([in] LPSTR Name); Input: Name of file with model of Auxiliary Reservoir
SetBrakeCoefFrictionByName	HRESULT _stdcall SetBrakeCoefFrictionByName([in] LPSTR Name); Input: Name of file with model of brake coefficient of friction
SetBrakeCoefFrictionByIndex	HRESULT _stdcall SetBrakeCoefFrictionByIndex([in] int Index); Input: Index=0..IInpTrain–>ComponentCount-1 – model of brake coefficient of friction
SetBrakeCylinderByIndex	HRESULT _stdcall SetBrakeCylinderByIndex([in] int Index

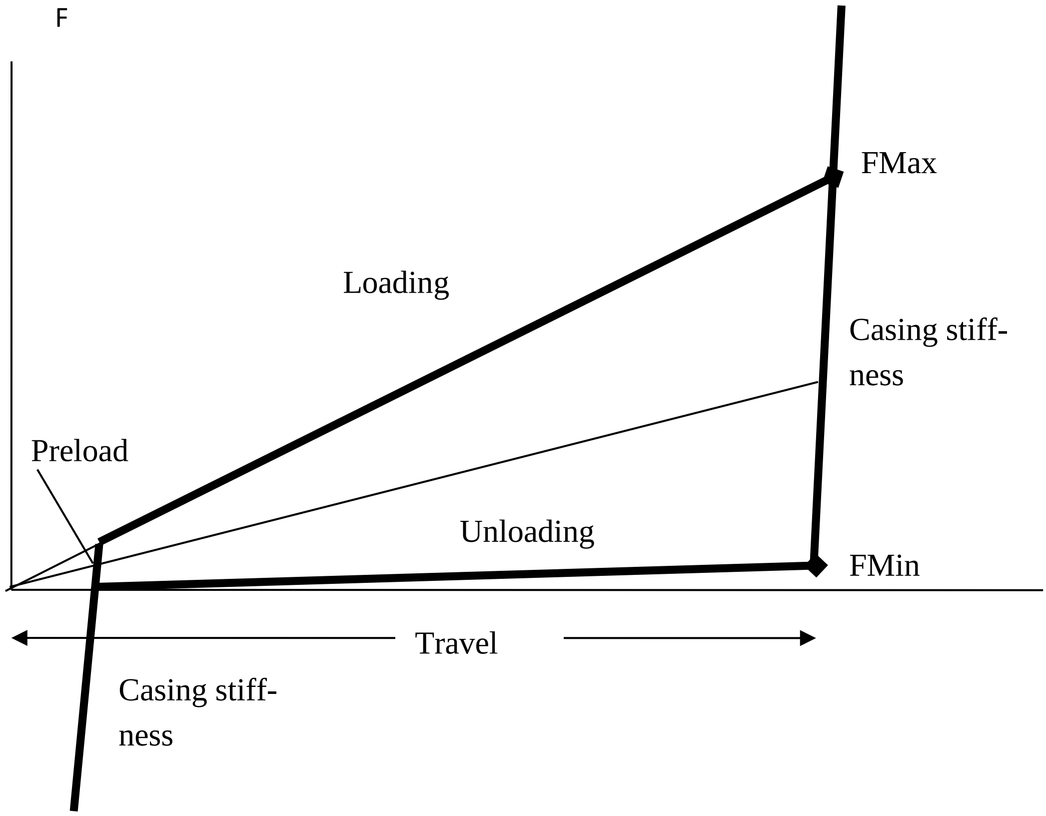
	); Input: Index=0..IInpTrain–>ComponentCount-1 – model of Brake Cylinder
SetBrakeCylinderByName	HRESULT _stdcall SetBrakeCylinderByName([in] LPSTR Name); Input: Name of file with model of Brake Cylinder
SetBrakeLeverageByIndex	HRESULT _stdcall SetBrakeLeverageByIndex([in] int Index); Assigns car brake leverage parameters from database. Brake leverage is stored in *.bf files in directory {TrainDataPath}\Brakes\Forces Input: Index=0..IInpTrain–>ComponentCount-1
SetBrakeLeverageByName	HRESULT _stdcall SetBrakeLeverageByName([in] LPSTR Name); Assigns car brake leverage parameters from database. Brake leverage is stored in *.bf files in directory {TrainDataPath}\Brakes\Forces Input: Name – name of car brake leverage in database.
SetBrakeValveByIndex	HRESULT _stdcall SetBrakeValveByIndex([in] int Index); Input: Index=0..IInpTrain–>ComponentCount-1 – model of Brake Valve
SetBrakeValveByName	HRESULT _stdcall SetBrakeValveByName([in] LPSTR Name); Input: Name of file with model of Brake Valve
SetBrakeSystemByIndex	HRESULT _stdcall SetBrakeSystemByIndex([in] int Index, [in] WordBool CheckPaths, [out] SYSINT IResult); Input: Index=0..IInpTrain–>ComponentCount-1 – model of BrakeSystem CheckPaths – if True, all internal paths in VP file are checked, IResult – bit-by-bit result of internal files checking. IResult bits: 0 bit – vehicle parameter file, 1 bit – resistance force model file (parameter “resistance”), 2 bit – auxiliary reservoir model file (parameter “auxreservoir”), 3 bit – brake cylinder model file (parameter “brakecylinder”), 4 bit – brake valve model file (parameter “brakevalve”), 5 bit – auxiliary (loco) brake valve model file (parameter

	“auxbrakevalve”), 6 bit – control valve model file (parameter “controlvalve”), 7 bit – acceleration chamber model file (parameter “acceleratingchamber”), 8 bit – compressor system model file (parameter “compressorsystem”), 9 bit – brake leverage model file (parameter “brakeleverage”), 10 bit – friction coefficient model file (parameter “coef”), other bits – always 0. For every bit: 0 – file exists, 1 – file does not exist.
SetBrakeSystemByName	HRESULT _stdcall SetBrakeSystemByName([in] LPSTR Name, [in] WordBool CheckPaths, [out] SYSINT IResult); Input: Name of file with model of Brake System CheckPaths – if True, all internal paths in VP file are checked, IResult – bit-by-bit result of internal files checking. IResult bits: 0 bit – vehicle parameter file, 1 bit – resistance force model file (parameter “resistance”), 2 bit – auxiliary reservoir model file (parameter “auxreservoir”), 3 bit – brake cylinder model file (parameter “brakecylinder”), 4 bit – brake valve model file (parameter “brakevalve”), 5 bit – auxiliary (loco) brake valve model file (parameter “auxbrakevalve”), 6 bit – control valve model file (parameter “controlvalve”), 7 bit – acceleration chamber model file (parameter “acceleratingchamber”), 8 bit – compressor system model file (parameter “compressorsystem”), 9 bit – brake leverage model file (parameter “brakeleverage”), 10 bit – friction coefficient model file (parameter “coef”), other bits – always 0. For every bit: 0 – file exists, 1 – file does not exist.
SetCGZPosition	HRESULT _stdcall SetCGZPosition([in] double Value); Input: Height of the center of mass of the vehicle above rail head (m). This value is used for calculation of the overturning factor, see IUMComTrainVehicle.GetOverturningFactor method, Sect. 20.6.5. *"IUMComTrainVehicle interface"*, p. 20-38.

SetCompressorByIndex	HRESULT _stdcall SetCompressorByIndex([in] int Index); Input: Index=0..IInpTrain–>ComponentCount-1 – model of Compressor
SetCompressorByName	HRESULT _stdcall SetCompressorByName([in] LPSTR Name); Input: Name of file with model of compressor
SetControlValveByIndex	HRESULT _stdcall SetControlValveByIndex([in] int Index, [in] WordBool CheckPaths, [out] SYSINT IResult); Input: Index=0..IInpTrain–>ComponentCount-1 – model of Control Valve CheckPaths – if True, all internal paths in CV file are checked, IResult – bit-by-bit result of internal files checking. IResult bits: 0 bit – control valve file, 1 bit – delta file (parameter “deltapath”), 2 bit – brake limiting curve file (parameter “brakelimit-path”), 3 bit – release limiting curve file (parameter “releaselimit-path”), 4–7 bits – always 0. For every bit: 0 – file exists, 1 – file does not exist.
SetControlValveByName	HRESULT _stdcall SetControlValveByName([in] LPSTR Name, [in] WordBool CheckPaths, [out] SYSINT IResult); Input: Name of file with model of Control Valve CheckPaths – if True, all internal paths in CV file are checked, IResult – bit-by-bit result of internal files checking. IResult bits: 0 bit – control valve file, 1 bit – delta file (parameter “deltapath”), 2 bit – brake limiting curve file (parameter “brakelimit-path”), 3 bit – release limiting curve file (parameter “releaselimit-path”), other bits – always 0. For every bit: 0 – file exists, 1 – file does not exist.
SetCouplingBase	HRESULT _stdcall SetCouplingBase([in] double Value); Input: coupling base of a car (m)
SetFrictionBufferGear	HRESULT _stdcall SetFrictionBufferGear([in] double Travel, [in] double FMin, [in] double FMax, [in] double Preload, [in] double CasingStiffness, [in] double CasingDamping, [in] double Damping, [in] long NGears);

	Assigns a frictional draft gear and its parameters, Sect. 20.5.2.1. *"Buffer gear parameters"*, p. 20-20.
SetGearByName	HRESULT _stdcall SetGearByName([in] LPSTR Name); Assigns a draft gear from database. Input: Name – name of a draft dear in database.
SetGearByIndex	HRESULT _stdcall SetGearByIndex([in] int Index); Assigns a draft gear from database. Input: Index 0.. IInpTrain–>ComponentCount-1 – index of a draft gear in database
SetMass	HRESULT _stdcall SetMass([in] double Value); Input: mass of a car (kg)
SetPivotBase	HRESULT _stdcall SetPivotBase([in] double Value); Input: pivot base a car (approximately distance between centers of bogies) (m)
SetResistanceByName	HRESULT _stdcall SetResistanceByName([in] LPSTR Name); Input: Name of file with model of resistance in tangent section
SetResistanceByIndex	HRESULT _stdcall SetResistanceByIndex([in] long Index); Input: Index=0..IInpTrain–>ComponentCount-1 – model of resistance in tangent section
SetSymmetricDraftGear	HRESULT _stdcall SetSymmetricDraftGear([in] double Travel, [in] double FMin, [in] double FMax, [in] double Gap, [in] double CasingStiffness, [in] double CasingDamping, [in] double Damping, [in] long NGears); Assigns a symmetric draft gear and its parameters, Sect. 20.5.2.2. *"Symmetric draft gear parameters"*, p. 20-21.
SetTractionMotorByIndex	HRESULT _stdcall SetTractionMotorByIndex([in] int Index); Assigns traction motor characteristics from database. Input: Index=0..IInpTrain–>ComponentCount-1
SetTractionMotorByName	HRESULT _stdcall SetTractionMotorByName([in] LPSTR Name); Assigns traction motor characteristics from database. Input: Name – name of a traction motor characteristics in database.

20.5.2.1. Buffer gear parameters

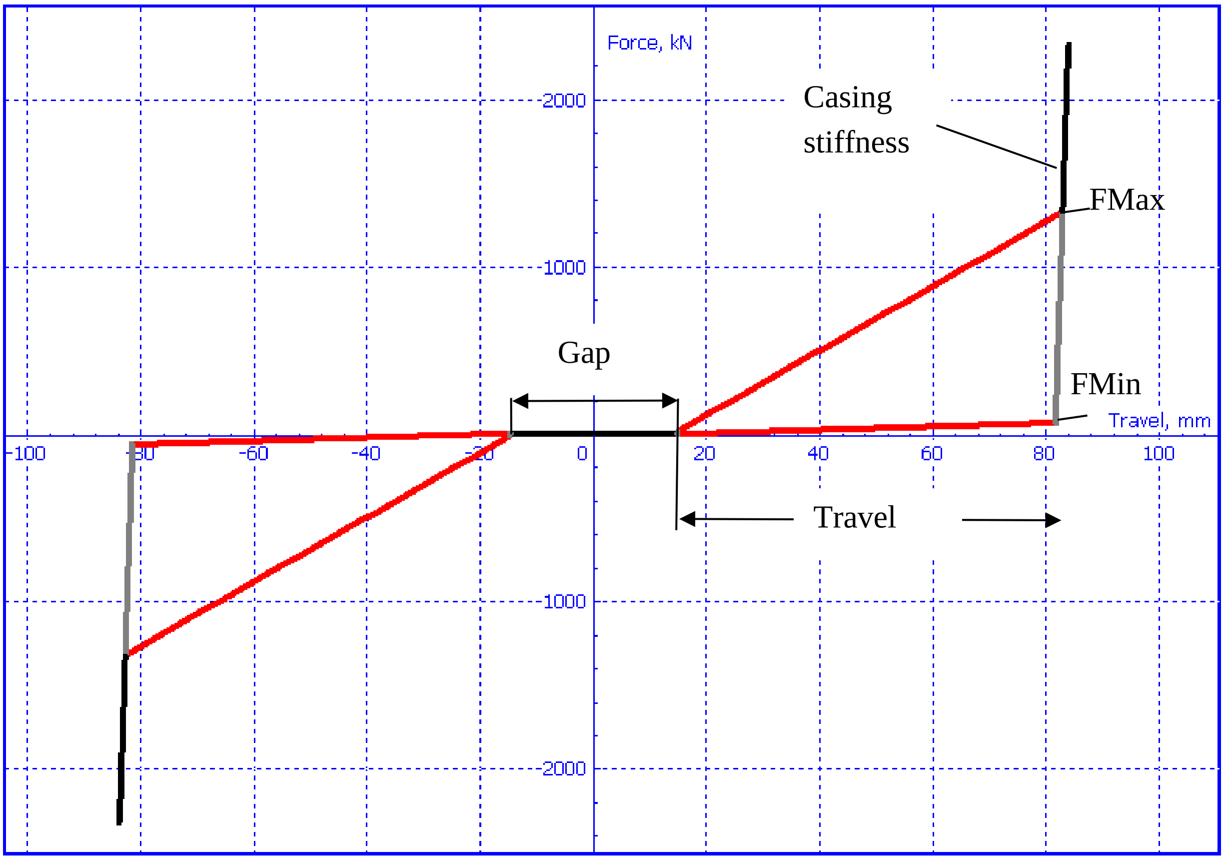


Buffer draft gear characteristic

Input: Travel (mm) –maximal travel of gear;
FMin (kN) – see figure,
FMax (kN) – see figure
Preload (kN) – preload stretching force in coupling;
CasingStiffness (N/m) – stiffness constant by reaching the maximal compression of gear;
CasingDamping (Ns/m) – damping constant by reaching the maximal compression of gear;
Damping (Ns/m) – damping constant in parallel with the draft gear characteristics.
NGears – number of gears in parallel (1 or 2)

Example (Mark-50 draft gear):
SetFrictionBufferGear(83, 66, 1330, 25, 2.0e9, 4.0e6, 1.0e4, 2).

20.5.2.2. Symmetric draft gear parameters



Symmetric draft gear characteristic

Input: Travel (mm) –maximal travel of gear;
FMin (kN) – see figure,
FMax (kN) – see figure
Gap (mm) – see figure;
CasingStiffness (N/m) – stiffness constant by reaching the maximal compression of gear;
CasingDamping (Ns/m) – damping constant by reaching the maximal compression of gear;
Damping (Ns/m) – damping constant in parallel with the draft gear characteristics.
NGears – number of gears in parallel (1 or 2)

Example:

SetSymmetricDraftGear (83, 66, 1330, 30, 2.0e9, 4.0e6, 1.0e4, 2).

20.5.3. IComLoco1D interface

IComLoco1D is used for setting 1D locomotive parameters and subsystems by development of a train model.

Interface: *IComCar1D*

Hierarchy: *IUnknown – IComCar1D- IComLoco1D*

Interface methods are inherited from *IComCar1D*, Sect. 20.5.2. *"IComCar1D interface"*, p. 20-15.

20.5.4. IComCar3D interface

IComCar3D is used for setting 3D vehicle parameters and subsystems by development of a train model.

Interface: *IComCar3D*

Hierarchy: *IUnknown – IComCar1D- IComLoco1D*

Most of methods are inherited from *IComCar1D*, Sect. 20.5.2. *"IComCar1D interface"*, p. 20-15.

Here is the list of additional methods.

Methods	**Description**
GetWheelsetCount	long _stdcall GetWheelsetCount(void); Output: number of wheelsets in the vehicle model
GetWheelsetRadius	double _stdcall GetWheelsetRadius(void); Output: radius of wheels(m)
SetWheelsetRadius	HRESULT _stdcall SetWheelsetRadius([in] double Value); Sets new wheel radius (m)

20.6. Interfaces for simulation of trains

20.6.1. IUMComTrain interface

IUMComTrain should be used in case of simulation of train dynamics. Interface allows access to 1D and 3D models included into a train model. Let us consider methods of *IUMComTrain*.

Interface: *IUMComTrain*

Hierarchy: *IUnknown – IUMComTrain*

Methods	Description
CoefFrictionMode	HRESULT _stdcall CoefFrictionMode([in] VARIANT_BOOL SetNumeric, [in] double Value, [in] double ValueWithSand); Input: If SetNumeric=true (1): Value is used as the current coefficient of friction and ValueWithSand as coefficient for sanding. Otherwise, if SetNumeric=false (0) the coefficient of friction is obtained from the macrogeometry file and the empirical model of friction with sanding is applied. Use SetEmpiricalSandingModel(false) for use of ValueWithSand. Data on friction coefficients see in Sect. 20.6.3. *"Coefficient of contact friction for different state of rail"*, p. 20-36.
Distance	double _stdcall Distance(void); Output: distance in meters from train start. It is calculated as and Σvdt, where v is the current speed of the locomotive, dt is the current time step size. Such definition leads to the following feature. If you firstly run the train in the positive direction, then stop and run it back to the start point the Distance value in the start point will be 0 again. To get the unsigned travelled distance as odometer value use GetOdometerValue instead, see below.
GetInTrainForce	HRESULT _stdcall GetInTrainForce([in] int VehicleIndex, [out] double * Value); Returns current Value of in-train force after the vehicle specified by VehicleIndex, N. First vehicle has index 0. The last VehicleIndex that has sense is VehicleCount-2, because there are no in-train forces after the last vehicle with the index VehicleCount-1.
GetLocomotiveByIndex	HRESULT _stdcall GetLocomotiveByIndex([in] int

	Index, [out] void * Locomotive); Returns via Locomotive parameter interface to the locomotive (IUMComLocomotive) specified by Index. Index starts from 0 up to LocomotiveCount-1.
GetOdometerValue	double _stdcall GetOdometerValue (void); Output: unsigned true travelled distance in meters from train start. Both positive and negative running directions increase the result.
GetVehicle3DByIndex	HRESULT _stdcall GetVehicle3DbyIndex([in] int Index, [out] void * Vehicle); Returns via Vehicle parameter interface to the vehicle (IUMComTrainVehicle) specified by Index. Index starts from 0 up to Vehicle3Dcount-1.
GetVehicleByIndex	HRESULT _stdcall GetVehicleByIndex([in] int Index, [out] void * Vehicle); Returns via Vehicle parameter interface to the vehicle (IUMComTrainVehicle) specified by Index. Index starts from 0 up to VehicleCount-1.
LocomotiveCount	int _stdcall LocomotiveCount(void); Returns count of locomotives in the train.
Vehicle3Dcount	int _stdcall Vehicle3Dcount(void); Returns count of 3D vehicles in the train.
VehicleCount	int _stdcall VehicleCount(void); Returns total count of vehicles in the train, including locomotives and all 1D and 3D vehicles.
Speed	float _stdcall Speed(void); Returns current speed of the first vehicle of the train, in fact, train speed, km/h.
StartSpeed	HRESULT _stdcall StartSpeed([in] float Value); Sets initial speed of the train, km/h.
SetEmpiricalSandingModel	HRESULT _stdcall SetEmpiricalSandingModel([in] VARIANT_BOOL Active); Input: Active = true (1) if the empirical model of adhesion with sanding is used (ValueWithSanding in method CoefFrictionMode is ignored in this case). Use Active = false(0) to apply coefficient of friction with sand ValueWithSanding in method CoefFrictionMode, see above.
SetFrictionCoefficientVsSliding	HRESULT _stdcall SetFrictionCoefficientVsSliding([in] double FactorA, [in] double FactorB);

	Input: parameters A, B characterizing decrease of adhesion with the growth of sliding velocity, Sect. 20.6.4. *"Decrease of adhesion with sliding"*, p. 20-37.
SetSanding	HRESULT _stdcall SetSanding([in] VARIANT_BOOL Active); Input: Active = true (1) corresponds to down position of sanding button whereas Active = false (0) corresponds to upper position of the button.
SetTrackType	HRESULT _stdcall SetTrackType([in] long TrackTypeIndex); Input: TrackTypeIndex = 0 corresponds to macrogeometry type of track whereas TrackTypeIndex = 1 corresponds to railroad track type.
GetTrackType	long _stdcall GetTrackType(void); Returns index of current track type (see SetTrackType method description).
SetBrakeThread	HRESULT _stdcall SetBrakeThread([in] long Value); Input: Value = 1 pneumatic brake system is simulated in the tread parallel to the multibody dynamics solver; Value = 0 – all calculations are realized in a single thread (default value). Note. SetBrakeThread method can not be used during integration. It should be called before PrepareIntegration method or after FinishIntegration method.
SetGauge	HRESULT _stdcall SetGauge([in] double Value); Sets the used railway track gauge in meters. Default value is 1.435 m. Track gauge is used for calculation of overturning factor only.
GetGauge	double _stdcall GetGauge(void); Returns the currently used track gauge. Track gauge is used for calculation of overturning factor only.
SetBlendBrakeDemand	HRESULT _stdcall CalcBlendBrakeDemand([in] double aDemand); Set the brake demand value for blending brake. Input: aDemand – brake demand (0 – no braking, 1 – maximal braking). Maximal braking corresponds to sitation when maximal brake force is applied to every vehicle of a train. Thus aDemand is ratio of maximal train brake force. Maximal force is calculated separetly for every vehicle as brake force with filled brake cylinders. Blending brake works as follows. When the demand is low and the ED brake power is enough, only ED brake

is activated. When the demand is high and ED brake is not enough, the ED brake is complemented with the pneumatic brake (blending), in motor and trailer bogies independently, according to the brake demand. On low speed, when ED brake power very small or zero, ED brake is fully substituted by pneumatic brake.

In UM one-dimensional vehicle models, the total number of wheelsets and the number of motor wheelsets are set by identifiers wheelset_count and motor_count respectively. By default, if a model has no such identifiers, wheelset count is equal to 4 and motor wheelset count is zero that corresponds to 4-axle car.

Note, that additionally the indices of motor wheelsets and the number of wheelsets are set in TMC-file (traction motor parameter files in folder ..\rw\Train\TractionMotors) by using the motoraxle parameter. For example the line in TMC-file:

motoraxle = (0, 1, 1, 0);

means that a locomotive model has 4 axles and the motor axles are the second and third ones: 0 – trailer axle (wheelset), 1 – motor axle.

So by using TMC-file, a user can define the location of motor wheelsets. If TMC-file is not assigned for a locomotive model, then motor wheelsets are supposed to be the last ones in the model. The total number of wheelsets and number of motor wheelsets defined in a model and in an assigned TMC-file must be equal. So, if in a TMC-file it is set that

motoraxle = (0, 1, 1, 0);

in a model it must be wheelset_count=4 and motor_count=2.

In case of necessity to get brake force for every wheelset of a vehicle, for example to watch how the brake blending works for trailer and motor wheelsets, use the GetBCBrakeForce function, see 20.6.5. *"IUMComTrainVehicle interface"*, p. 20-38 for details.

For example, if a locomotive has 2 trailer wheelsets and 2 motor wheelsets and 8 brake cylinders (2 brake cylinders per wheelset) and TMC-file not assigned (the motor wheelsets are the lsat ones), to get brake forces for every wheelset use the following code:

```
var
  aVehicle : IUMComTrainVehicle;
```

```
begin
…
// Trailer wheelsets (first ones)
  BrakeForce1WS := Locomotive.GetBCBrakeForce(0)
+   Locomotive.GetBCBrakeForce(1);
  BrakeForce2WS := Locomotive.GetBCBrakeForce(2)
+ Locomotive.GetBCBrakeForce(3);
// Motor wheelsets (last ones)
  BrakeForce3WS := Locomotive.GetBCBrakeForce(4)
+   Locomotive.GetBCBrakeForce(5);
  BrakeForce4WS := Locomotive.GetBCBrakeForce(6)
+ Locomotive.GetBCBrakeForce(7);
…
```

SetInitialBrakeSystemPressures	HRESULT _stdcall SetInitialBrakeSystemPressures(void) Set pressure values in brake devices, brake pipes and feed pipes as they were after loading train model.
SetHoldingBrakeRelease	HRESULT _stdcall SetHoldingBrakeRelease([in] VARIANT_BOOL aState); Sets holding brake release.
GetHoldingBrakeRelease	VARIANT_BOOL _stdcall GetHoldingBrakeRelease(void); Returns if holding brake released or not.
SetElectronicBrakeControl	HRESULT _stdcall SetElectronicBrakeControl([in] VARIANT_BOOL aState); Sets the state of the electronic brake control. If aState = true, then the electronic brake control brake control is used; if aState = false – not used. The electronic brake control is provided in UM brake system models by using brake control units (BCU) installed on vehicles. Files with parameters of BCUs are store in folder ../Train/Brakes/BCU. BCU supports the following features: direct electropneumatic brake, indirect electropneumatic brake, electronic control of brake pipe pressure for direct and indirect brakes, assimilation of brake pipe.
GetElectronicBrakeControl	VARIANT_BOOL _stdcall GetElectronicBrakeControl(void); Return the state of the electronic brake control.
GetHoldingBrakeManualMode	VARIANT_BOOL _stdcall GetHoldingBrakeManualMode(void); Returns if holding brake is in manual mode.

SetHoldingBrakeManualMode	HRESULT _stdcall SetHoldingBrakeManualMode([in] VARIANT_BOOL aMode); Sets holding brake in manual mode. Input: true – set manual mode for holding brake, false -– cancel manual mode for holding brake.
The group of methods related to work with railroad track type.	
GetCurrentElementID	int _stdcall GetCurrentElementID(void); Returns GlobalID of current element under the first point of train. Global ID's of railroad elements are defined from reading of railroad XML file, Sect. 20.6.8. *"IRailRoad interface"*, p. 20-70.
GetCurrentSectionID	int _stdcall GetCurrentSectionID(void); Returns ID of active section of current element (see method above). 1- first section; 2 – second one, etc.
GetLocalSectionPosition	double _stdcall GetLocalSectionPosition(void); Returns current local position in meters of the front point of the train on the active section of current element (see methods above). Returning value is limited by zero and limit length defined for the section in railroad XML file, Sect. 20.6.8. *"IRailRoad interface"*, p. 20-70.
GetLocalSectionPositionRatio	double _stdcall GetLocalSectionPositionRaio (void); Returns current local position of the front point of the train on the active section of current element (see methods above) as ratio within [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. In comparision with the GetLocalSectionPosition presented above it provides more smooth train visualization for the external graphical engine.
GetFrontPointSlope	double _stdcall GetFrontPointSlope (void); Returns slope of railroad track under the first point of train in ppm.
GetFrontPointCurveRadii	double _stdcall GetFrontPointCurveRadii (void); Returns radii of the railroad track curve under the first point of train in meters: 1e10 for tangetn track, negative values for right curve, positive value for left curve.
GetFrontPointRRPosition	HRESULT _stdcall GetFrontPointRRPosition([out] int * ElementID, [out] int * SectionID, [out] double * LocalPosition, [out] int * ElementType); Returns current position of the front point of the train

	on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalSectionPosition is local position in meters on the active section of current element, ElementType is a flag of positioning of the front point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).
GetLastWheelSetRRPosition	HRESULT _stdcall GetLastWheelSetRRPosition([out] int * ElementID, [out] int * SectionID, [out] double * LocalPosition, [out] int * ElementType); Returns current position of the last wheelset, if the last vehicle is 3D one, or the last wheelset point, if the last vehicle is 1D one, of the train. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPosition is local position in meters on the active section of current element, ElementType is a flag of positioning of the wheelset on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).
GetFrontPointRRPositionRatio	HRESULT _stdcall GetFrontPointRRPosition ([out] int * ElementID, [out] int * SectionID, [out] double * LocalPositionRatio, [out] int * ElementType); Returns current position of the front point of the train on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPositionRatio is local position ratio in [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. ElementType is a flag of positioning of the wheelset on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).
GetLastPointRRPosition	HRESULT _stdcall GetLastPointRRPosition([out] int * ElementID, [out] int * SectionID, [out] double * LocalPosition, [out] int * ElementType);

	Returns current position of the last point of the train on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalSectionPosition is local position in meters on the active section of current element, ElementType is a flag of positioning of the last point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).
GetLastPointRRPositionRatio	HRESULT _stdcall GetLastPointRRPosition ([out] int * ElementID, [out] int * SectionID, [out] double * LocalPositionRatio, [out] int * ElementType); Returns current position of the last point of the train on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPositionRatio is local position ratio in [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. ElementType is a flag of positioning of the last point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).
GetMaxMinAvailablePositions	HRESULT _stdcall GetMaxMinAvailablePositions([out] double* aMaxPosition, [out] double* aMinPosition, [out] int* aMaxObstacleIndex, [out] int* aMinObstacleIndex); Returns current available track length for the positive-speed (aMaxPosition) and negative-speed (aMinPosition) motion of the train. Returns S_Ok if the current train position is admissible (aMinPosition, aMaxPosition values are positive), S_False in oposite case. aMaxObstacleIndex, aMinObstacleIndex parameters returns the type of limitation: 0 – end of railroad; 1 – switch with an unsuitable state (motion throw the switch will result do its damaging); 2 – virtual train head/tail point.
The group of methods related to work with tank car train.	

GetLiquidModelCount	long _stdcall GetLiquidModelCount(void); Returns count of available liquid models. The model list is loaded from ..\rw\Train\Liquid folder when train model is loading.
GetLiquidModelName	LPSTR _stdcall GetLiquidModelName([in] long Index); Returns name of the liquid model defined with Index in the model lists or ’Not defined’ if Index is out of bounds [0, … , GetLiquidModelCount].

20.6.2. IVirtualTrain interface

IVirtualTrain should be used in case of simulation of so-called virtual trains. Let us consider methods of *IVirtualTrain*.

Note: Virtual train model enables simplified description of real train, setting of its initial position on railroad, and control on train motion by input of its kinematic characteristics (speed, velocity, target speed). Position of virtual train on railroad is calculated at each step of time domain simulation of the main object dynamics.

Interface: *IVirtualTrain*
Hierarchy: *IUnknown – IVirtualTrain*

Methods	Description
Methods to control parameters of the train	
SetLength	HRESULT _stdcall SetLength([in] double Length); Sets the length of the virtual train (m).
GetLength	double _stdcall GetAcceleration(void); Returns the length of the virtual train (m).
SetCaption	HRESULT _stdcall SetCaption([in] LPSTR TrainCaption); Sets the virtual train caption.
GetCaption	LPSTR _stdcall SetCaption(void); Returns caption of the train.
Methods to start/stop train simulation	
StartSimulation	HRESULT _stdcall StartSimulation(void); Starts the virtual train simulation. Returns S_Ok in success, S_False in oposite case.
StopSimulation	HRESULT _stdcall StopSimulation(void); Stops the virtual train simulation. Returns S_Ok in success, S_False in oposite case.
Methods to control train motion/kinematics	

Distance	double _stdcall Distance(void); Output: distance in meters from train start. It is calculated as and Σvdt, where v is the current speed of the locomotive, dt is the current time step size. Such definition leads to the following feature. If you firstly run the train in the positive direction, then stop and run it back to the start point the Distance value in the start point will be 0 again. To get the unsigned travelled distance as odometer value use GetOdometerValue instead, see below.
GetOdometerValue	double _stdcall GetOdometerValue (void); Output: unsigned true travelled distance in meters from train start. Both positive and negative running directions increase the result.
SetSpeed	HRESULT _stdcall SetSpeed([in] double Speed) Determines constant speed (m/s) of the virtual train. It keeps constant train speed and zero acceleration till it will be changed by SetAcceleration, SetSpeed or SetTargetSpeed methods. Initial train speed is zero by default.
GetSpeed	double _stdcall GetSpeed(void); Returns current speed of the train in m/s.
SetAcceleration	HRESULT _stdcall SetAcceleration([in] double Acceleration) Determines constant acceleration (m/s2) of the virtual train. It keeps constant acceleration till it will be changed by SetAcceleration, SetSpeed or SetTargetSpeed methods. Initial train acceleration is zero by default.
GetAcceleration	double _stdcall GetAcceleration(void); Returns current acceleration of the train in m/s2 set by SetAcceleration or SetTargetSpeed methods. If the current speed of the train is given by SetSpeed then GetAcceleration returns zero.
SetTargetSpeed	HRESULT _stdcall SetTargetSpeed([in] double TargetSpeed, [in] double TargetDistance) TargetSpeed is the speed (m/s) that train should have in TargetDistance meters. The method calculates and keeps the required uniform positive or negative acceleration that should be applied to provide TargetSpeed in TargetDistance taking into account current speed. It keeps constant uniform acceleration till the target distance will be reached; the train has constant TargetSpeed velocity after TargetDistance is reached. The speed control can be changed by SetAcceleration, SetSpeed or SetTargetSpeed methods. Example: SetTargetSpeed(0, 5000) provides uniform deceleration to stop

	the train in 5 km.
The group of methods related to work with railroad track type.	
SetFrontRRPosition	HRESULT _stdcall SetFrontPointRRPosition([in] int ElementID, [in] int SectionID, [in] double LocalPosition, [in] VARIANT_BOOL PositiveDirection); The method describes the (initial) position of the train. Input: ElementID is GlobalID of a railroad element, SectionID is ID of railroad section of railroad element, LocalPosition is local position in meters on the active section of current element, PositiveDirection is the flag of the train direction that influences on the way how the train will be placed along the railroad.
GetDirection	int _stdcall GetDirection(void); Returns direction of the virtual train: 0 – forward; 1 – backward.
GetInitialElementID	int _stdcall GetInitialElementID(void); Returns initial railroad element ID for the train.
GetInitialSectionID	int _stdcall GetInitialSectionID(void); Returns initial railroad element section ID for the train.
GetInitialElementPosition	int _stdcall GetInitialElementPositionID(void); Returns initial position on the railroad element section for the train.
IsInputDataCorrect	HRESULT _stdcall IsInputDataCorrect(void); Returns S_Ok if initial position (InitialElementID, InitialSectionID, InitialElementPosition parameters) are defined correctly; returns S_False in oposite case.
GetRRPosition	HRESULT _stdcall GetFrontPointRRPosition([in] double DistanceFromTrainHead, [out] int * ElementID, [out] int * SectionID, [out] double * LocalPosition, [out] int * ElementType); Returns current position of the front point of the train on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPosition is local position in meters on the active section of current element, ElementType is a flag of positioning of the front point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element).
GetRRPositionRatio	HRESULT _stdcall GetFrontPointRRPosition([in] double DistanceFromTrainHead, [out] int * ElementID, [out] int * Sec-

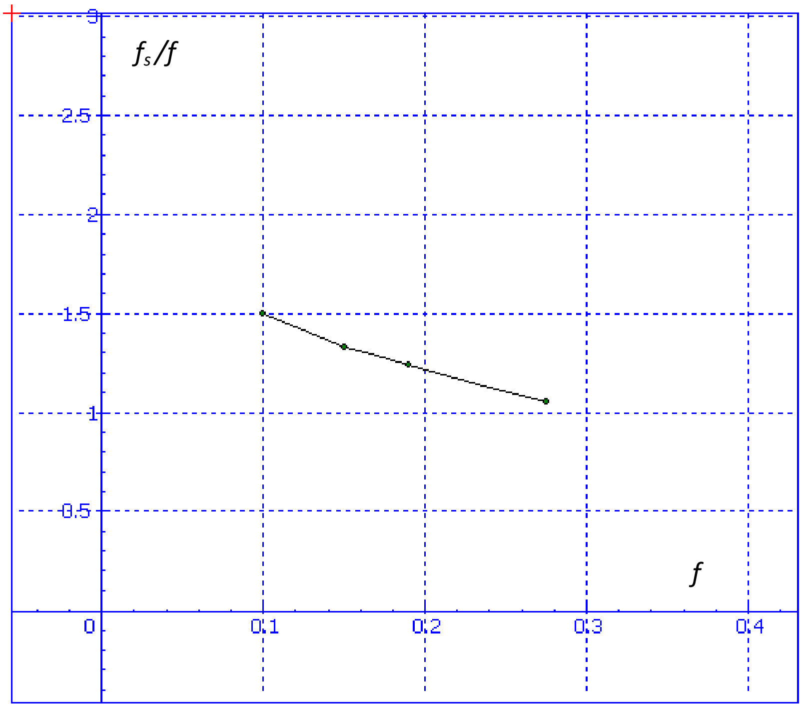
	tionID, [out] double * LocalPositionRatio, [out] int * ElementType); The method is completely identical to the GetFrontPointRRPosition, described above, with the only difference that the LocalPositionRatio is local position ratio in [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section.
GetCurrentElementID	int _stdcall GetCurrentElementID ([in] double DistanceFromTrainHead); Returns GlobalID of current element of the requested point of train. Length is given in meters and signifies the distance from the head end of the train in the direction that is opposite to the running direction. Example: GetCurrentElementID(0) returns GobalID of the head end of the train. GetCurrentElementID(IVirtualTrain.Length) returns GobalID of the tail end of the train.
GetCurrentElementType	int _stdcall GetCurrentElementType ([in] double DistanceFromTrainHead); Returns flag of current element type of the requested point of train (0 if current element is a road; 1 if it is a switch.
GetCurrentSectionID	int _stdcall GetCurrentSectionID ([in] double DistanceFromTrainHead); Returns ID of active section of current element (see method above) of the requested points of the train. The first section has index 1. Length is given in meters and signifies the distance from the head end of the train in the direction that is opposite to the running direction. Example: GetCurrentSectionID(0) returns section index of the head end of the train. GetCurrentSectionID(IVirtualTrain.Length) returns section index of the tail end of the train.
GetLocalSectionPosition	double _stdcall GetLocalSectionPosition ([in] double DistanceFromTrainHead); Returns current local position in meters of the requested point of the train on the active section of current element (see methods above). Returning value is in between by zero and the length of the defined railroad section. Length is given in meters and signifies the distance from the head end of he train in the direction that is opposite to the running direction.

	Example: GetLocalSectionPosition(0) returns local position of the head end of the train. GetLocalSectionPosition(IVirtualTrain.Length) returns local position of the tail end of the train.
GetLocalSectionPositionRatio	double _stdcall GetLocalSectionPositionRatio([in] double DistanceFromTrainHead); Returns current local position as ratio within [0, 1] interval of the requested point of the train on the active section of current element (see methods above). Length is given in meters and signifies the distance from the head end of he train in the direction that is opposite to the running direction. Example: GetLocalSectionPositionRatio(0) returns local position of the head end of the train. GetLocalSectionPositionRatio(IVirtualTrain.Length) returns local position of the tail end of the train.
The group of additional methods for the train state control.	
IsSimulationStarted	HRESULT _stdcall IsInputDataCorrect(void); Returns S_Ok if input data for the train is correct and simulation of the train motion was started; returns S_False in opposite case.
GetSpeedControlHistoryLog	LPSTR _stdcall SetCaption(void); Returns text with log of speed history controls for the train.

20.6.3. Coefficient of contact friction for different state of rail

Condition of rail surface	Traction coefficient
Dry rail (clean)	0.25–0.30
Dry rail (with sand)	0.25–0.33
Wet rail (clean)	0.18–0.20
Wet rail (with sand)	0.22–0.25
Greasy rail	0.15–0.18
Moisture on rail	0.09–0.15
Sleet on rail	0.15
Sleet on rail (with sand)	0.20
Light snow on rail	0.10
Light snow on rail (with sand)	0.15
Wet leaves on rail	0.07

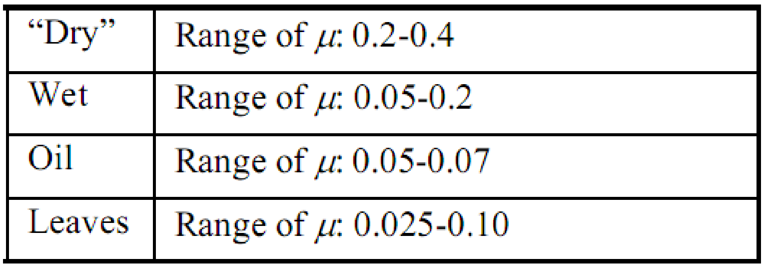
Source: Rolling Contacts (Tribology in Practice Series) by T. A. Stolarski, S. Tobe Published in February 15, 2001, Wiley, 298P.



Ratio of average values of coefficient of friction with (fs) /without (f) sanding

Empirical model for coefficient of friction with sanding:

$$f_s = \begin{cases} 0.15, & f < 0.1 \\ f(1.75 - 2.5f), & 0.1 \le f \le 0.3 \\ f, & f > 0.3 \end{cases}$$



"Dry"	Range of μ: 0.2-0.4
Wet	Range of μ: 0.05-0.2
Oil	Range of μ: 0.05-0.07
Leaves	Range of μ: 0.025-0.10

Source: K. Nagase, A study of adhesion between the rails and running wheels on main lines: results of investigations by slipping adhesion test bogie, Proceedings of the IMechE Part F, Journal of Rail and Rapid Transit 203 (1989), 33-43.

20.6.4. Decrease of adhesion with sliding

The following formula is used for coefficient of friction versus sliding velocity:

$$f = f_0\left((1 - A)e^{-Bv_s} + A\right),$$

where A is the ratio of limit friction coefficient f_∞ at infinity slip velocity to maximum friction coefficient f_0,

$$A = \frac{f_\infty}{f_0},$$

B (s/m) is the coefficient of exponential friction decrease, v_s (m/s) is the sliding velocity.

Typical values for locomotives:

	Dry	Wet
A	0.40	0.40
B (s/m)	0.60	0.20

Source: O. Polach: Creep forces in simulations of traction vehicles running on adhesion limit. Wear 258 (2005) 992–1000.

20.6.5. IUMComTrainVehicle interface

IUMComTrainVehicle is any vehicle of the train, including 1D and 3D vehicles.

Interface: *IUMComTrainVehicle*

Hierarchy: *IUnknown – IUMComTrainVehicle*

Methods	Description
GetFrontCouplingForce	double _stdcall GetFrontCouplingForce(void); Returns front in-train (coupling) forces, N.
GetBCPressure	double _stdcall GetBCPressure([in] int Index); Output: Pressure of brake cylinder by index, Pa. Index starts from 0, so the first brake cylinder has index 0.
GetMainPipePressure	double _stdcall GetMainPipePressure(void); Output: Main pipe pressure, Pa
SetMainPipePressure	HRESULT _stdcall SetMainPipePressure([in] double aPressure); Input: aPressure – main pipe pressure, Pa
SetBCPressure	HRESULT _stdcall SetBCPressure ([in] int Index, [in] double aPressure); Input: Index – brake cylinder index; aPressure – brake cylinder pressure, Pa Sets pressure to brake cylinder by index. Index starts from 0, so he first brake cylinder has index 0.
SetBCPressureByBP	HRESULT _stdcall SetBCPressureByBP ([in] int Index, [in] double aPressure); Input: Index – brake cylinder index; aPressure – brake pipe pressure, Pa Sets brake cylinder pressure according to brake pipe pressure using current control valve characteristics. Index starts from 0, so he first brake cylinder has index 0.
GetPositionInTrain	int _stdcall GetPositionInTrain(void); Output: Position of vehicle in train. Starts with 1
GetRearCouplingForce	double _stdcall GetRearCouplingForce(void); Returns rear in-train (coupling) forces, N.
Is3DVehicle	VARIANT_BOOL _stdcall Is3Dvehicle(void); Output: True (1) if 3D vehicle, False if 1D vehicle
IsLocomotive	VARIANT_BOOL _stdcall IsLocomotive(void); Output: True (1) if current vehicle is locomotive
Speed	double _stdcall Speed(void); Output: Current vehicle speed, m/s
GetLongitudinalAcceleration	double _stdcall GetLongitudinalAcceleration (void); Output: Current vehicle acceleration, m/s2

GetCouplingStretch	double _stdcall GetCouplingStretch (void); Output: Relative stretch of rear coupler of the current vehicle, m
GetCouplingSlack	double _stdcall GetCouplingSlack (void); Output: Slack in rear coupler of the current vehicle, m. To use this method, vehicle model should have two identifiers: MinSlack and MaxSlack which set minimal and maximal stretch of coupler in slack.
GetAdhesionLimitRatio	double _stdcall GetAdhesionLimitRatio([in] int WSIndex); Input: Wheelset index (the first wheelset has index 1). Output: The ratio of current traction (positive) or braking (negative) force on a wheelset to maximal adhesion force. Positive output is available for the tractive vehicles. Note. This method should be used for 1D vehicles only. GetAdhesionLimitRatio = 1 corresponds the case when the traction force is equal to maximal adhesion force. GetAdhesionLimitRatio = -1 corresponds the case when the braking force is equal to maximal adhesion force. Case (GetAdhesionLimitRatio > 1) means that the current traction force is bigger than the maximal adhesion force and the wheelsets is skidding now. Continuous slipping of the locomotive wheels causes wheel-burn defects, especially at zero or low speed. Case (GetAdhesionLimitRatio < -1) means that the current braking force is bigger than the maximal adhesion force and the wheelset is about to be blocked. Depending on exposure time the flat wheel defect might appear. Sanding (Sect. 20.6.3. *"Coefficient of contact friction for different state of rail"*, p. 20-36) increases the adhesion limit and thereby decreases the current sliding ratio.
GetSlidingVelocity	double _stdcall GetSlidingVelocity([in] int WSIndex); Input: Wheelset index (the first wheelset has index 1). Output: Linear sliding velocity in the contact of wheelset and rail. If sliding velocity <> 0 then the wheelset is skidding.

SetParkBrakeState	HRESULT _stdcall SetParkBrakeState([in] VARIANT_BOOL ParkBrakeState); Input: ParkBrakeState – state of park brake: True – park brake is activated, False – park brake is deactivated. The park brake model works in the following way. When the brake is activated, brake force is increasing from 0 to the value which is set by parameter ParkBrakeForce in VP file (in folder ..\rw\Train\Vehicles) during time interval set by parameter ParkBrakeTime in the same file. When the brake is deactivated the force is decresing from the ParkBrakeForce value to 0 during the same time. Park brake parameters (VP file): ParkBrakeForce is the maximal park brake force, N ParkBrakeTime is time inreval when the brake force is increasing from 0 to the maximal value or decreasing from the maximal one to 0, s.
SetLeakage	HRESULT _stdcall SetLeakage ([in] double aLeakage); Input: aLeakage – leakage rate in a vehicle, Pa/min Sets leakage rate [Pa/min] for a vehicle
GetLeakage	double _stdcall GetLeakage(void); Output: leakage rate [Pa/min] for a vehicle
SetCVEnabled	HRESULT _stdcall SetCVEnabled ([in] VARIANT_BOOL aEnabled); Input: if aEnabled is True – control valve is enabled, if aEnabled is False – control valve is disabled
GetCVEnabled	VARIANT_BOOL _stdcall GetCVEnabled(void); Output: True – if control valve is enabled, False – if control valve is disabled
SetBCEnabled	HRESULT _stdcall SetBCEnabled ([in] int Index, [in] VARIANT_BOOL aEnabled); Input: if aEnabled is True – brake cylinder is enabled, if aEnabled is False – brake cylinder is disabled. Index starts from 0, so the first brake cylinder has index 0.
GetBCEnabled	VARIANT_BOOL _stdcall GetBCEnabled(void); Output: True – if brake cylinder is enabled, False – if brake cylinder is disabled. Index starts from 0, so the first brake cylinder has index 0.
GetBCCount	int _stdcall GetBCCount (void); Output: Number of brake cylinders on vehicle
SetARLeakage	HRESULT _stdcall SetARLeakage ([in] double aLe-

	akage); Input: aLeakage – leakage rate in auxiliary reservoirs of vehicle, Pa/min Sets leakage rate [Pa/min] for auxiliary reservoirs
GetARLeakage	double _stdcall GetARLeakage(void); Output: leakage rate [Pa/min] for a auxiliary reservoirs
SetARPressure	HRESULT _stdcall SetARPressure([in] double aPressure); Input: aPressure – auxiliary reservoir pressure, Pa
GetARPressure	double _stdcall GetARPressure(void); Output: Auxiliary reservoir pressure, Pa
SetPressureFactor	HRESULT _stdcall SetPressureFactor ([in] double aFactor); Input: aFactor – pressure factor for pressure in brake cylinder
GetPressureFactor	double _stdcall GetPressureFactore(void); Output: Pressure factor for pressure in brake cylinder
GetDBVPressure	double _stdcall GetDBVPressure(void); Output: Driver's brake valve pressure, Pa
D5On	HRESULT _stdcall D5On (void); Turn on D5 valve.
D5Off	HRESULT _stdcall D5Off(void); Turn off D5.
D6On	HRESULT _stdcall D6On (void); Turn on D6 valve.
D6Off	HRESULT _stdcall D6Off(void); Turn off D6 valve.
IsLiquid	VARIANT_BOOL _stdcall IsLiquid(void); Output: True (1) if current vehicle is tank car
SetLiquidModel	HRESULT _stdcall SetLiquidModel([in] long Index, [in] double h_R, [in] double Density); Set car liquid model parameters. HRESULT is S_Ok if model is set, S_False if not. Input: Index – index of liquid model (0 by default); h_R – fluid level relative to tank radii (1 – half volume filling); Density – density of the liquid, kg/m3. Example: Vehicle := IUNKnown(P) as IUMComTrainVehicle; if Vehicle.IsLiquid then Vehicle.SetLiquidModel(Index, h_R, Density));

GetLiquidModel	HRESULT _stdcall SetLiquidModel([out] long* Index, [out] double* h_R, [out] double* Density); Returns car liquid model parameters. HRESULT is S_Ok if model is set, S_False if not. Output: Index is index of liquid model (0 by default); h_R is fluid level relative to tank radii (1 – half volume filling); Density is density of the liquid, kg/m3.
SetLeverageRatio	HRESULT _stdcall SetLeverageRatio([in] double aRatio); Input: aRatio is leverage ratio of vehicle brake system.
GetLeverageRatio	double _stdcall GetLeverageRatio(void); Output: leverage ratio of vehicle brake system.
LoadCVFromFile	HRESULT _stdcall LoadCVFromFile ([in] LPSTR aFileName); Input: aFileName is file name of control valve parameter file. For example, ‘Ke1a-G.cv’.
SetBrakeFactor	HRESULT _stdcall SetBrakeFactor ([in] double aFactor); Input: aFactor is Brake factor for total brake force of a vehicle.
GetBrakeFactor	double _stdcall GetBrakeFactore(void); Output: Brake factor for total brake force of a vehicle.
SetAmbientTemperature	HRESULT _stdcall SetAmbientTemperature ([in] double aTemperature); Input: aTemperature is ambient temperature in Celsius degree (ºC) for brake fade model. Temperature is
GetAmbientTemperature	double _stdcall GetAmbientTemperature (void); Output: current ambient temperature in Celsius degree (ºC) for brake fade model.
SetCurrentBrakeTemperature	HRESULT _stdcall SetCurrentBrakeTemperature ([in] double aTemperature); Input: aTemperature is temperature in Celsius degree (ºC) of brake shoes for brake fade model.
GetCurrentBrakeTemperature	double _stdcall GetCurrentBrakeTemperature (void); Output: current temperature in Celsius degree (ºC) of brake shoes for brake fade model.
GetBrakeFadeCoef	double _stdcall GetBrakeFadeCoef (void); Output: current brake fade coefficient.
GetOverturningFactor	double _stdcall GetOverturningFactor(void); Output: Returns overturning factor on curved track in [0..1] range. Overturning factor is calculated as a ratio between the current vehicle speed and the maximal

	(critical) vehicle speed in a curve when inner wheels lift off rail. Overturning factor is calculated for quasi-static motion. Influence of railway track irregularities is ignored. Overturning factor is zero in tangent tracks and bigger then zero in curves. Function requires that every vehicle has vertical_mass_center_position parameter that is used for calculation of the overturning factor. If the vehicle has no such a parameter the function returns -1 as a result.
SetOrificeToAtmosphere	HRESULT _stdcall SetOrificeToAtmosphere([in] double aDiameter); Input aDiameter – diameter of orifice from brake pipe to atmosphere, [m]. Sets the diameter of an orifice from brake pipe to atmosphere. Can be applied for simulation of leakages, disjoints in brake pipe and so on.
GetOrificeToAtmosphere	double _stdcall GetOrificeToAtmosphere(void); Output – diameter of orifice from brake pipe to atmosphere, [m].
SetAuxReservoirOrificeToAtm	HRESULT _stdcall SetAuxReservoirOrificeToAtm([in] double aDiameter); Input aDiameter – diameter of orifice from auxiliary reservoir to atmosphere, [m]. Sets the diameter of an orifice from auxiliary reservoir to atmosphere. Can be applied for simulation of leakages, consumption for pantograph actuator and so on.
GetAuxReservoirOrificeToAtm	double _stdcall GetAuxReservoirOrificeToAtm(void); Output – diameter of orifice from auxiliary reservoir to atmosphere, [m].
SetBPCloggedRatio	HRESULT _stdcall SetBPCloggedRatio([in] double aRatio); Input aRatio is the ratio of clogged brake pipe square to full brake pipe square. Sets the ratio of clogged brake pipe square to full brake pipe square: 1 corresponds to fully clogged brake pipe, 0 corresponds to non-clogged (clean) brake pipe.
GetBPCloggedRatio	double _stdcall GetBPCloggedRatio(void) Returns ratio of clogged brake pipe squa re to full brake pipe square.
SetCVReleaseEnabled	HRESULT _stdcall SetCVReleaseEnabled([in] VARIANT_BOOL aReleaseEnabled)

	Sets if a control valve can release brake cylinders or not. Used for simulation of locked brakes.
GetCVReleaseEnabled	VARIANT_BOOL _stdcall GetCVReleaseEnabled(void) Returns if a control valve enables to release brake cylinders. Used for simulation of locked brakes.
GetFeedPipePressure	double _stdcall GetFeedPipePressure(void) Returns the pressure [Pa] in feed pipe of a vehicle. If no feed pipe on the vehicle, returns zero.
SetFeedPipePressure	HRESULT _stdcall SetFeedPipePressure([in] double aPressure) Sets the pressure [Pa] in feed pipe of a vehicle.
SetTargetBCPressureEP	HRESULT _stdcall SetTargetBCPressureEP([in] int anIndex, [in] double aPressure) Set the target pressure for a brake cylinder. This function changes the pressure not instantly; it uses current control valve diagrams of filling and releasing the brake cylinders. If the brake pipe pressure corresponds to higher brake cylinder pressure, this higher pressure will be set in brake cylinders. Input: anIndex – brake cylinder index (starts from 0); aPressure – pressure in brake cylinder, Pa. This function can be used for the simulation of electronically controlled pneumatic brakes.
GetMaxPneumaticBrakeForce	double _stdcall GetMaxPneumaticBrakeForce(void) Returns maximal force [N] of pneumatic brake of a vehicle.
GetMaxBCPressure	double _stdcall GetMaxBCPressure(void) Returns maximal possible brake cylinder pressure [Pa] of a vehicle.
GetBCBrakeForce	double _stdcall GetBCBrakeForce([in] int anIndex) Returns brake force which is provided by a brake cylinder. Input: anIndex is brake cylinder index (starts from 0). For example, if a 4-axle vehicle model has 2 brake cylinders, this function returns the brake force per 2 wheelsets; if the same vehicle has 8 cylinders then the function returns a half of the force per a wheelset and

	so on (see also the example in the description of the CalcBlendingBrakeDemand function).
GetWSCount	int _stdcall GetWSCount(void) Returns the number of wheelsets of a vehicle. The total number of wheelsets and the number of motor wheelsets are set by identifiers wheelset_count and motor_count respectively. By default, wheelset_count=4 and motor_count=0, that corresponds to 4-axle car. Note that in UM one-dimensional vehicle models, motor wheelsets are always last ones in a model.
SetBCUSignalUsing	HRESULT _stdcall SetBCUSignalUsing([in] VARIANT_BOOL aState); If aState=true then a control valve on a vehicle uses the control pressure calculated by BCU, otherwise – the control valve uses pressure from the brake pipe.
GetBCUSignalUsing(void)	VARIANT_BOOL _stdcall GetBCUSignalUsing(void); Returns the state of BCU signal using.
SetEmergencyState	HRESULT _stdcall SetEmergencyState([in] VARIANT_BOOL aState); If aState=true, it opens an orifice from brake pipe to atmosphere to provide the brake pipe pressure descreasing in emergency barking mode. If aState=false, it closes the orifice.
GetEmergencyState	VARIANT_BOOL _stdcall GetEmergencyState(void); Returns the state of an orifice from brake pipe to atmosphere for emergency braking. Output: True – the orifice is open (emergency braking is on); False – the orifice is closed (emergency braking is off).
GetActiveForce	double _stdcall GetActiveForce(void); Returns active (traction or brake) force applied to a vehicle not taking into account adhesion limit. Force in Newtons.
SetATSState	HRESULT _stdcall SetATSState([in] VARI-

	ANT_BOOL aState); Set the state of automatic train stop system. If aState=true, the system is activated that leads to opening the orifice in brake pipe to atmosphere. If aState=false, the system is deactivated and there is no connection with atmosphere.
GetATSState	VARIANT_BOOL _stdcall GetATSState(void); Returns the state of automatic train stop system.
SetCompressorSupply	HRESULT _stdcall SetCompressorSupply([in] double aSupply); Sets the supply of compressor, cubic meter per second.
GetCompressorSupply(void)	double _stdcall GetCompressorSupply(void); Returns the supply of compressor, cubic meter per second.
GetIsAuxReservoirFed	VARIANT_BOOL _stdcall GetIsAuxReservoirFed(void); Returns if auxiliary reservoir is fed.
SetIsAuxReservoirFed	HRESULT _stdcall SetIsAuxReservoirFed([in] VARIANT_BOOL aState); Sets or cuts off feeding of auxiliary reservoir.
SetEPUsing	HRESULT _stdcall SetEPUsing([in] VARIANT_BOOL aState); Sets the activity of electro-pneumatic brake on the vehicle. This function sets the availability of electro-pneumatic brake but not applies it.
GetEPUsing	VARIANT_BOOL _stdcall GetEPUsing(void); Returns the activity of electro-pneumatic brake on the vehicle: is it available or not.
GetTargetCVPressure	double _stdcall GetTargetCVPressure([in] long anIndex); Returns pressure in brake cylinders which should be obtained by using control signal from control valve.
GetTargetLocoBVPressure	double _stdcall GetTargetLocoBVPressure([in] long anIndex);

	Returns pressure in brake cylinders which should be obtained by using control signal from auxiliary brake valve.
GetTargetEPPressure	double _stdcall GetTargetEPPressure([in] long anIndex); Returns pressure in brake cylinders which should be obtained by using control signal from electro-pneumatic brake system.
GetTargetBCUPressure	double _stdcall GetTargetBCUPressure([in] long anIndex); Returns pressure in brake cylinders which should be obtained by using control signal from brake control unit.
ApplyHoldingBrakeInManualMode	HRESULT _stdcall ApplyHoldingBrakeInManualMode([in] VARIANT_BOOL Apply); Apply holding brake in manual mode. Input: true – apply holding brake, false – release holding brake.
IsAppliedHoldingBrakeInManualMode	VARIANT_BOOL _stdcall IsAppliedHoldingBrakeInManualMode(void); Returns if holding brake is apllied in manual mode.
The group of methods related to work with railroad track type.	
GetFrontWSetElementID	int _stdcall GetFrontWSetElementID(void); Returns GlobalID of current element under the first wheelset of the vehicle. Global ID's of railroad elements are defined from reading of railroad XML file, Sect. 20.6.8. *"IRailRoad interface"*, p. 20-70.
GetFrontWSetSectionID	int _stdcall GetFrontWSetSectionID(void); Returns ID of active section of current element under the first wheelset of the vehicle (see method above). 1– first section; 2 – second one, etc.
GetFrontWSetLocalSectionPosition	double _stdcall GetFrontWSetLocalSectionPosition(void); Returns current local position in meters of the first wheelset of the vehicle on the active section of current element (see methods above). Returning value is limited by zero and limit length defined for the sec-

	tion in railroad XML file, Sect. 20.6.8. *"IRailRoad interface"*, p. 20-70.
GetFrontWSetLocalSectionPositionRatio	double _stdcall GetFrontWSetLocalSectionPosition-Raio(void); Returns current local position of the first wheelset of the vehicle on the active section of current element (see methods above) as ratio within [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. In comparision with the GetLocalSectionPosition presented above it provides more smooth train visualization for the external graphical engine.
GetFrontWSetSlope	double _stdcall GetFrontWSetSlope(void); Returns slope of railroad track under the first wheelset of the vehicle in ppm.
GetFrontWSetRRPosition	HRESULT _stdcall GetFrontWSetRRPosition([out] int * ElementID, [out] int * SectionID, [out] double * LocalPosition, [out] int * ElementType); Returns current position of the first wheelset of the vehicle on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPosition is local position in meters on the active section of current element, ElementType is a flag of positioning of the front point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).
GetFrontWSetRRPositionRatio	HRESULT _stdcall GetFrontWSetRRPosition ([out] int * ElementID, [out] int * SectionID, [out] double * LocalPositionRatio, [out] int * ElementType); Returns current position of the first wheelset of the vehicle on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPositionRatio is local position ratio in [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. ElementType is a flag of positioning of the wheelset on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start

	mode is active).
GetFrontPointElementID	int _stdcall GetFrontPointElementID(void); Returns GlobalID of current element under the front point of the vehicle. Global ID's of railroad elements are defined from reading of railroad XML file, Sect. 20.6.8. *"IRailRoad interface"*, p. 20-70.
GetFrontPointSectionID	int _stdcall GetFrontPointSectionID(void); Returns ID of active section of current element under the front point of the vehicle (see method above). 1– first section; 2 – second one, etc.
GetFrontPointLocalSectionPosition	double _stdcall GetFrontPointLocalSectionPosition(void); Returns current local position in meters of the front point of the vehicle on the active section of current element (see methods above). Returning value is limited by zero and limit length defined for the section in railroad XML file, Sect. 20.6.8. *"IRailRoad interface"*, p. 20-70.
GetFrontPointLocalSectionPositionRatio	double _stdcall GetFrontPointLocalSectionPosition-Raio(void); Returns current local position of the front point of the vehicle on the active section of current element (see methods above) as ratio within [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. In comparision with the GetLocalSectionPosition presented above it provides more smooth train visualization for the external graphical engine.
GetFrontPointRRPosition	HRESULT _stdcall GetFrontPointRRPosition([out] int * ElementID, [out] int * SectionID, [out] double * LocalPosition, [out] int * ElementType); Returns current position of the front point of the vehicle on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPosition is local position in meters on the active section of current element, ElementType is a flag of positioning of the front point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and

	start mode is active).
GetFrontPointRRPositionRatio	HRESULT _stdcall GetFrontPointRRPosition ([out] int * ElementID, [out] int * SectionID, [out] double * LocalPositionRatio, [out] int * ElementType); Returns current position of the front point of the vehicle on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPositionRatio is local position ratio in [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. ElementType is a flag of positioning of the front point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).
GetLastPointElementID	int _stdcall GetLastPointElementID(void); Returns GlobalID of current element under the last point of the vehicle. Global ID’s of railroad elements are defined from reading of railroad XML file, Sect. 20.6.8. *"IRailRoad interface"*, p. 20-70.
GetLastPointSectionID	int _stdcall GetLastPointSectionID(void); Returns ID of active section of current element under the last point of the vehicle (see method above). 1– first section; 2 – second one, etc.
GetLastPointLocalSectionPosition	double _stdcall GetLastPointLocalSectionPosition(void); Returns current local position in meters of the last point of the vehicle on the active section of current element (see methods above). Returning value is limited by zero and limit length defined for the section in railroad XML file, Sect. 20.6.8. *"IRailRoad interface"*, p. 20-70.
GetLastPointLocalSectionPositionRatio	double _stdcall GetLastPointLocalSectionPositionRaio(void); Returns current local position of the last point of the vehicle on the active section of current element (see methods above) as ratio within [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. In comparision with the GetLocalSectionPosition presented above it

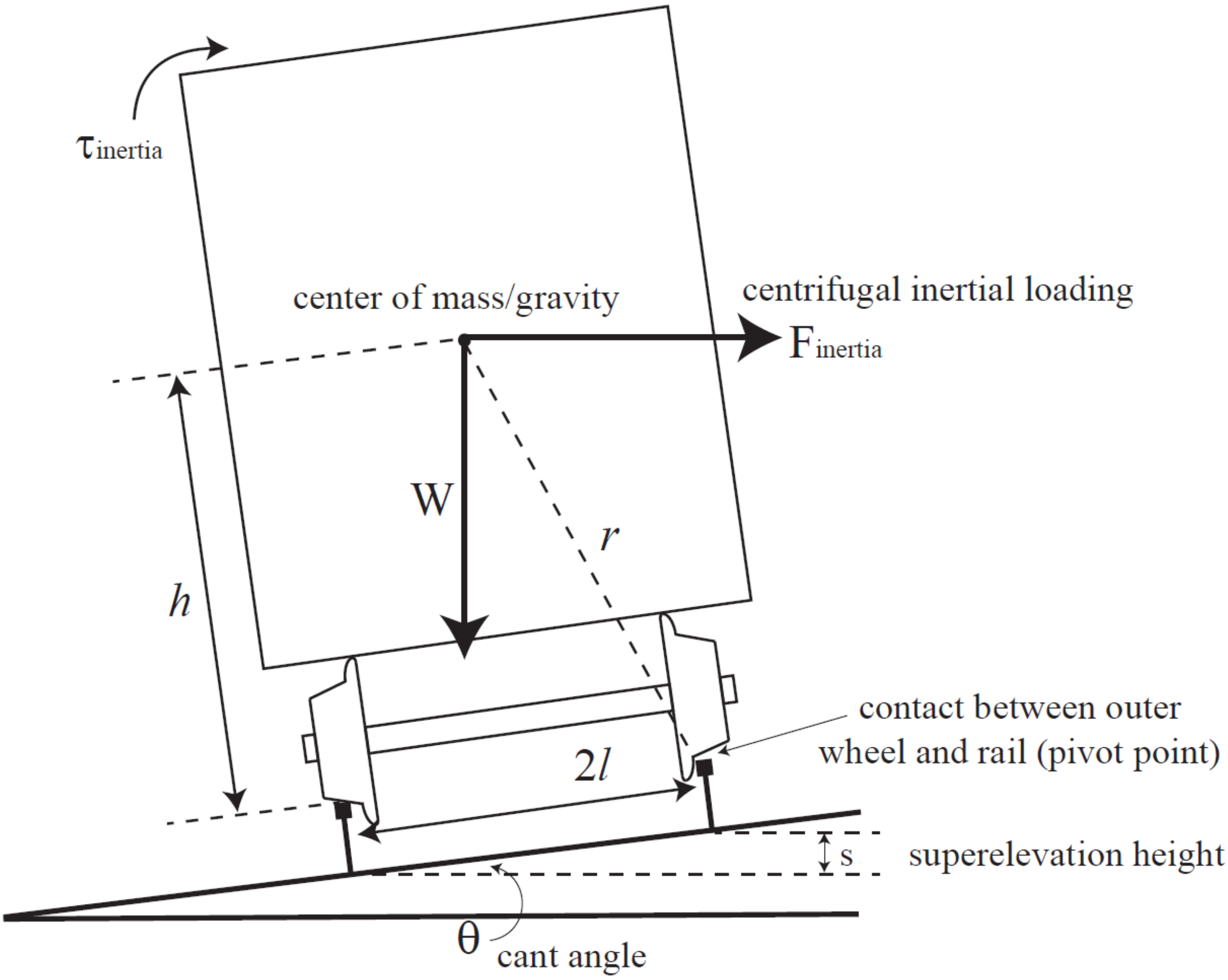
	provides more smooth train visualization for the external graphical engine.
GetLastPointRRPosition	HRESULT _stdcall GetLastPointRRPosition([out] int * ElementID, [out] int * SectionID, [out] double * LocalPosition, [out] int * ElementType); Returns current position of the last point of the vehicle on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPosition is local position in meters on the active section of current element, ElementType is a flag of positioning of the last point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).
GetLastPointRRPositionRatio	HRESULT _stdcall GetLastPointRRPosition ([out] int * ElementID, [out] int * SectionID, [out] double * LocalPositionRatio, [out] int * ElementType); Returns current position of the last point of the vehicle on the railroad. See methods above. Output: ElementID is GlobalID of current element, SectionID is ID of active section of current element, LocalPositionRatio is local position ratio in [0, 1] range, where 0 corresponds to the beginning of the section and 1 corresponds to the end of the section. ElementType is a flag of positioning of the last point on the railroad (0 if current element is a road; 1 if it is a switch; 2 if position is out of element and start mode is active).

20.6.5.1. Overturning factor

Overturning factor is the ration between the current speed of the vehicle and the critical speed of the vehicle in the particular curve. Critical speed on curved track, which corresponds to the situation when inner wheels lift off rail, can be calculated according to the following formula:

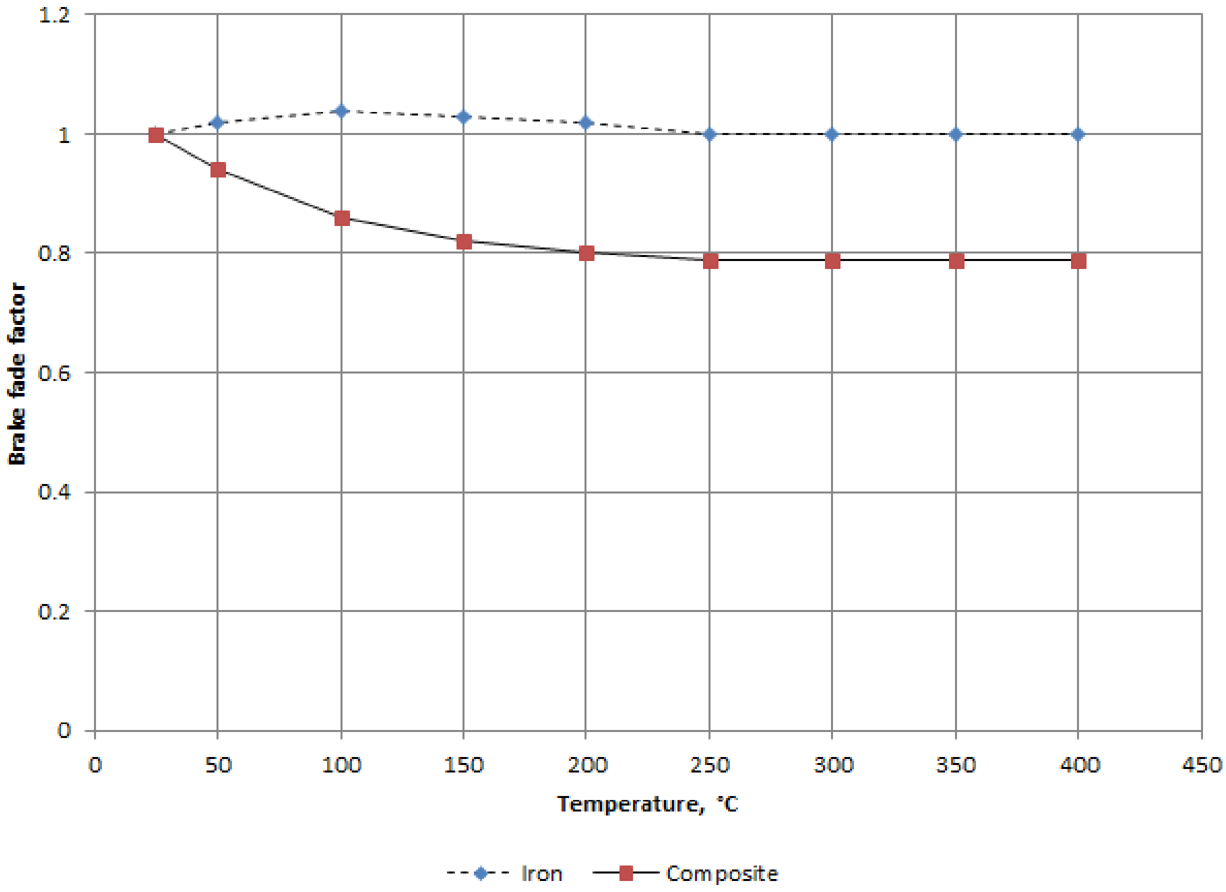
$$V_{max} = \sqrt{\frac{Rg(h\ \sin\theta + l\cos\theta)}{h\ \cos\theta - l\sin\theta}}, \text{where}$$

R is the current radius of curvature of the transient curve or constant radius curve,
g is free fall acceleration,
h is the vertical position of the vehicle center of mass relative to rail head,
l is a half gauge,
θ is a cant angle.
Railcar force diagram on a superelevated curve is given below.



20.6.5.2. Brake fade factor

Brake fade factor depends on shoes temperature and expressed by the equation and diagram like showed below.



You can set this function by setting curve property in the brake fade coefficient file.

Current temperature is calculated by heat transfer of brake equipment.

Heat balance on every integration time step is described with the following equation:

$$C_p \cdot M \cdot \frac{dT_b}{dt} = \dot{Q}_{in} - \dot{Q}_{out}, \text{ where}$$

C_p is the specific heat of shoes,

M is the mass of the shoes and other brake equipment;

$\dot{Q}_{in}$ is the heat input applied to a brake shoes during brake application (power);

$\dot{Q}_{out}$ is the transfer of heat from the shoes to the air around (power);

T_b is the current temperature of the shoes.

Please note, you can set the specific heat of shoes and the mass of the shoes in the brake fade coefficient file.

The power of the heat input is

$$\dot{Q}_{in} = F_b \cdot V, \text{ where}$$

F_b is the current brake force;
V is the current vehicle speed;

The power of the heat output is

$$\dot{Q}_{out} = h_c \cdot A \cdot (T_b - T_\infty), \text{ where}$$

h_c is the film coefficient and is velocity dependent;
A is the area of the shoes.

$(h_c \cdot A)$ product – a factor in the equation for $\dot{Q}_{out}$ – can be expressed as polynom shown below.

$$h_c \cdot A = b_1 V + b_2, \text{where}$$

V is vehicle speed,
b_1, b_2 are polynom coefficients.
You can set values of the polynomes in the brake fade coefficient file.

20.6.6. IUMComLocomotive interface

IUMComLocomotive is a locomotive. The interface is used for control of traction and braking modes.

Interface: *IUMComLocomotive*

Hierarchy: *IUnknown – IUMComTrainVehicle – IUMComLocomotive*

Methods	Description
AuxBrakePositionCount	int _stdcall AuxBrakePositionCount(void); Returns count of auxiliary brake positions
BrakeValvePositionCount	int _stdcall BrakeValvePositionCount(void); Returns count of brake valve positions
DynamicBrakePositionCount	int _stdcall DynamicBrakePositionCount(void); Returns count of dynamic brake positions
GetAuxBrakePosition	int _stdcall GetAuxBrakePosition(void); Returns current auxiliary brake valve position
GetBrakeValvePosition	int _stdcall GetBrakeValvePosition(void); Returns current brake valve position
GetBrakeValvePositionComment	LPSTR _stdcall GetBrakeValvePositionComment([in] long Index); Input: Index=1.. BrakeValvePositionCount – Position of Brake Vale Output: Comment to the brake valve position
GetAuxBrakeValvePositionComment	LPSTR _stdcall GetAuxBrakeValvePositionComment([in] long Index); Input: Index=1..AuxBrakePositionCount – Position of Locomotive Brake Valve Output: Comment to the locomotive brake valve position
GetDynamicBrakePosition	int _stdcall GetDynamicBrakePosition(void); Returns current dynamic brake position
GetDynBrakeCurrentFactor	double _stdcall GetDynBrakeCurrentFactor(void); Returns scale factor for dynamic brake current calculation
GetEntranceCurrent	double _stdcall GetEntranceCurrent(void); Returns entrance current, A
GetEqualizingReservoirPressure	double _stdcall GetEqualizingReservoirPressure(void); Output: Equalizing reservoir pressure
GetGearRatio	double _stdcall GetGearRatio(void); Returns ratio of reducer gear
GetMainReservoirPressure	double _stdcall GetMainReservoirPressure(void); Output: Main reservoir pressure, Pa
SetMainReservoirPressure	HRESULT _stdcall SetMainReservoirPressure([in] double aPressure); Input: aPressure – main reservoir pressure, Pa

GetBrakeValvePressure	double _stdcall GetBrakeValvePressure(void); Output: Pressure in brake pipe directly after driver brake valve, Pa
GetMotorActive	VARIANT_BOOL _stdcall GetMotorActive([in] long Index); Input: Index = 0..GetMotorCount-1 – index of motor Output: True (1) is the motor is active
GetMotorCount	long _stdcall GetMotorCount(void); Output: number of traction motors. For 3D locomotive models only
GetMotorCurrent	double _stdcall GetMotorCurrent([in] long Index); Output: electrical current in anchor of a traction motor number Index, A
GetMotorName	LPSTR _stdcall GetMotorName([in] long Index); Input: Index = 0..GetMotorCount-1 – index of motor Output: Name of UM force element for traction torque or name in form of Motor#N – where N – is the Index+1
GetMotorRPM	double _stdcall GetMotorRPM([in] long Index); Output: rotation frequency of a traction motor number Index, RPM
GetPowerConsumption	double _stdcall GetPowerConsumption(); Output: electrical motor power consumption for locomotives with motor voltage 900 V (E43000), kW
GetReversePosition	int _stdcall GetReversePosition(void); Returns position of reverser: -1 – backward, 0 – neutral, 1 – forward run
GetRWheel	double _stdcall GetRWheel(); Output: radius of a loco wheel, m
GetThrottlePosition	int _stdcall GetThrottlePosition(void); Returns current throttle position
GetThrottleContPosition	double _stdcall GetThrottlePosition(void); Returns current throttle position
GetThrottleCurrentFactor	double _stdcall GetThrottleCurrentFactor(void); Returns scale factor for throttle current calculation
GetTractionEffort	double _stdcall GetTractionEffort (); Output: traction effort (N) in of a loco as a sum of traction efforts of wheel-motor sets
SetPowerConsumption	HRESULT _stdcall SetPowerConsumption([in] double aPowerConsumption); Resets electrical motor power consumption by aPowerConsumption in kW
SetAuxBrakePosition	HRESULT _stdcall SetAuxBrakePosition([in] int Value); Sets auxiliary brake position. Value starts from 1 up to AuxBrakePositionCount.

SetBrakeValvePosition	HRESULT _stdcall SetBrakeValvePosition([in] int Value); Sets brake valve position. Value starts from 1 up to BrakeValvePositionCount.
SetDynamicBrakePosition	HRESULT _stdcall SetDynamicBrakePosition([in] int Value); Sets dynamic brake position. Value starts from 0 up to DynamicBrakePositionCount.
SetDynBrakeCurrentFactor	HRESULT _stdcall SetDynBrakeCurrentFactor([in] int aDynBrakeCurrentFactor); Sets scale factor for dynamic brake current calculation
SetEntranceCurrent	HRESULT _stdcall SetEntranceCurrent ([in] double aCurrent); Sets entrance current in amperes.
SetGearRatio	HRESULT _stdcall SetGearRatio([in] double aGearRatio); Sets reducer gear ratio.
SetMotorActive	HRESULT _stdcall SetMotorActive([in] long Index, [in] VARIANT_BOOL Active); Input: Index = 0..GetMotorCount-1 – index of motor Active: True (1) to set the motor active, False (0) to make it inactive
SetReversePosition	HRESULT _stdcall SetReversePosition([in] long Value); Sets position of reverser: -1 – backward, 0 – neutral, 1 – forward run
SetRWheel	HRESULT _stdcall SetRWheel([in] double aRWheel); Sets radius of loco wheels in meters.
SetThrottlePosition	HRESULT _stdcall SetThrottlePosition([in] int Position); Sets throttle (traction) position
SetThrottleContPosition	HRESULT _stdcall SetThrottleContPosition([in] double Position); Sets throttle (traction) position
SetThrottleCurrentFactor	HRESULT _stdcall SetThrottleCurrentFactor([in] double aThrottleCurrentFactor); Sets scale factor for throttle current calculation
ThrottlePositionCount	int _stdcall ThrottlePositionCount(void); Returns count of throttle positions of the locomotive
EmergencyBrakeOn	HRESULT _stdcall EmergencyBrakeOn(void); Opens a valve in brake pipe for emergency braking. It is a feature of driver’s brake valve (DBV), works only if DBV exists on the current locomotive
EmergencyBrakeOff	HRESULT _stdcall EmergencyBrakeOff(void); Closes a valve in brake pipe for emergency braking. It is a feature of driver’s brake valve (DBV), works only if DBV exists on the current locomotive

AutoPurgerOn	HRESULT _stdcall AutoPurgerOn(void); Opens so-called autopurger.
AutoPurgerOff	HRESULT _stdcall AutoPurgerOff(void); Closes so-called autopurger.
C12On	HRESULT _stdcall C12On (void); Turn on C12 valve (Stop Cock)
C12Off	HRESULT _stdcall C12Off(void); Turn off C12 valve (Stop Cock)
C5On	HRESULT _stdcall C5On (void); Turn on C5 valve
C5Off	HRESULT _stdcall C5Off(void); Turn off C5 valve
CompressorOn	HRESULT _stdcall CompressorOn(void); Turn compressor on, if compressor exists on the current locomotive
CompressorOff	HRESULT _stdcall CompressorOff(void); Turn compressor off, if compressor exists on the current locomotive
PurgerOn	HRESULT _stdcall PurgerOn(void); Opens purger. It works if brake pipe pressure is within 3-4.9 bar.
PurgerOff	HRESULT _stdcall PurgerOff(void); Closes purger.
SetDBVLeakageFlow	HRESULT _stdcall SetDBVLeakageFlow ([in] double aFlow); Set leakage flow for a vehicle with brake valve. For example, for open B2 or hose cock it must be approximately 0.5. Do not use very high value, more than 10.
GetDBVLeakageFlow	double _stdcall GetDBVLeakageFlow (void); Returns leakage flow for a vehicle with brake valve.
SetERLeakageFlow	HRESULT _stdcall SetERLeakageFlow ([in] double aFlow); Set leakage flow for equalizing reservoir of brake valve.
GetERLeakageFlow	double _stdcall GetERLeakageFlow (void); Returns leakage flow for equalizing reservoir of brake valve.
SetPurgerFactor	HRESULT _stdcall SetPurgerFactor([in] double aFactor); Sets scale factor for brake cylinders release speed by purger
GetPurgerFactor	double _stdcall GetPurgerFactor (void); Returns scale factor for brake cylinders release speed by purger
SetAutoPurgerFactor	HRESULT _stdcall SetAutoPurgerFactor([in] double aFactor); Sets scale factor for brake cylinders release speed by autopurger

GetAutoPurgerFactor	double _stdcall GetAutoPurgerFactor (void); Returns scale factor for brake cylinders release speed by autopurger
GetTractionPower	double _stdcall GetTractionPower (void); Returns power of traction force (W).
SetTargetBPPressure	HRESULT _stdcall SetTargetBPPressure ([in] double aTargetBPPressure); Sets the target pressure aTargetBPPressure in brake pipe which should be reached for the current brake valve position. As a rule the target brake pipe pressure is constant for certain brake valve position and must not be changed. This function is used if it is necessary to change brake pipe pressure value on the current brake valve position, for example when the pressure in brake pipe depends on the angle of rotation of brake valve handle. Initial brake pipe pressure for a brake valve position is set by parameter bptargetpressure in brakevalveposition block in BV file. Example:

```
with brakevalveposition;
  mode="sbraking";
  name="Service brake";
  comment="Braking position";
  position=3;
  gradualchanging=true;
  ReleaseEnabled=true;
  bptargetpressure=340000;
```

In this example, when the brake valve handle is moved to the third position (position=3), the pressure will decrease till 340000 Pa (bptargetpressure=340000) by the rate of service braking (mode="sbraking"). Parameter gradualchanging=true means that this handle position has an area in which the brake pipe pressure depends on the angle of rotation of the handle and user can change the target pressure in the brake pipe.

GetTargetBPPressure	double _stdcall GetTargetBPPressure (void); Returns the target pressure in brake pipe which should be reached for current brake valve position.
GetGradualChanging	VARIANT_BOOL _stdcall GetGradualChanging (void); Returns possible (true) or not (false) to change the target brake pipe pressure on the current brake valve position. It is just information for user that this brake valve position has an area where the brake pipe pressure depends on the angle of

	rotation of the handle. This parameter does not enable or disable the changing the brake pipe pressure.
GetBrakeValvePositionName	LPSTR _stdcall GetBrakeValvePositionName ([in] long aIndex); Returns the name of brake valve position with index aIndex. The name of a position is set by parameter name in BV file. For example:

```
with brakevalveposition;
  mode="sbraking";
  name="Service brake";
  comment="Braking position";
  position=2;
  gradualchanging=false;
  bptargetpressure=300000;
```

Here the name of the position is "Service brake".

SetTargetBCPressure	HRESULT _stdcall SetTargetBCPressure ([in] double aFactor); Sets the target pressure in locomotive brake cylinder (aTargetBCPressure) which should be reached for current auxiliary (locomotive) brake valve position. This function is used if it is necessary to change brake cylinder pressure value on the current auxiliary brake valve position, for example when the pressure in brake cylinder depends on the angle of rotation of auxiliary brake valve handle. Initial brake cylinder pressure for a brake is set by parameter bctargetpressure in the auxbrakevalveposition block in LBV file. Example:

```
with auxbrakevalveposition;
  name="Braking";
  comment="Incremental braking position";
  mode="braking";
  position=2;
  GradualChanging=true;
  bctargetpressure=380000;
```

In this example, when the auxiliary brake valve handle is moved to the second position (position=2), the pressure in brake cylinders of a locomotive will increase till 380000 Pa (bctargetpressure=380000). Parameter gradualchanging=true means that this handle position has an area in which the brake cylinder pressure depends on the angle of rotation of the handle.

GetTargetBCPressure	double _stdcall GetTargetBCPressure (void);

	Returns the target pressure in brake cylinder of a locomotive which should be reached for current auxiliary brake valve position.
GetAuxGradualChanging	VARIANT_BOOL _stdcall GetAuxGradualChanging (void); Returns possible (true) or not (false) to change the target pressure in locomotive brake cylinders on the current auxiliary brake valve position. It is just information for user that this auxiliary brake valve position has an area where the brake cylinder pressure depends on the angle of rotation of the handle. This parameter does not enable or disable the changing the locomotive brake cylinder pressure.

GetAuxBrakeValvePositionName	LPSTR _stdcall GetAuxBrakeValvePositionName ([in] long aIndex);
	Returns the name of auxiliary brake valve position with index aIndex. The name of a position is set by parameter name in LBV file. For example:
	with auxbrakevalveposition;
	name="Full brake";
	comment="Full braking position";
	mode="braking";
	position=3;
	GradualChanging=false;
	bctargetpressure=380000;
	Here the name of the position is "Full brake".

GetReleaseEnabled	VARIANT_BOOL _stdcall GetReleaseEnabled (void); Returns if it is possible or not to increase brake pipe pressure (release brakes) for current brake valve position. This parameter is used only for braking positions. As a rule the value of the parameter depends on the type of brake system: false – for trains with graduated release operations; true – for trains which have only direct release operations.
SetReleaseEnabled	HRESULT _stdcall SetReleaseEnabled([in] VARIANT_BOOL aState); Sets the possibility to increase brake pipe pressure for the current brake valve position. This parameter is used only for braking positions. As a rule the value of the parameter depends on the type of brake system: false – for trains with graduated release operations; true – for trains which have only direct release operations.
GetInstantFuelConsumption	double _stdcall GetInstantFuelConsumption (void); Returns instant fuel consumption, kg/min

GetRemainingFuel	double _stdcall GetRemainingFuel (void); Returns remaining fuel, kg
SetRemainingFuel	HRESULT _stdcall SetRemainingFuel ([in] double Value); Sets the value of remaining fuel, kg
GetCommonFuelConsumption	double _stdcall GettCommonFuelConsumption (void); Returns common fuel consumption from start simulation
SetCommonFuelConsumption	HRESULT _stdcall SetRemainigFuel ([in] double Value); Sets the value of common fuel consumption from start simulation, kg
GetTotalMotorForce	double _stdcall GetTotalMotorForce(void) Returns the total motor force of a locomotive. If motor are in traction mode then the sign of the force will be as velocity sign, else in brake mode the sign of the force will be opposite.
GetDynamicBrakeContPosition	double _stdcall GetDynamicBrakeContPosition(void) Returns current dynamic brake position.
SetDynamicBrakeContPosition	HRESULT _stdcall SetDynamicBrakeContPosition([in] double aPosition) Sets continuous dynamic brake position.
GetMaxDynamicBrakeForce	double _stdcall GetMaxDynamicBrakeForce([in] double aVelocity, [in] int aPosition) Returns maximal dynamic brake force [N] of a locomotive.
SetNominalBPPressureByDBV	HRESULT _stdcall SetNominalBPPressureByDBV([in] double aPressure) Sets nominal (regular operation) pressure in train brake pipe. When pressure changed, the brake pipe will be feeded or released through driver’s brake valve till the needed value.
GetNominalBPPressureByDBV	double _stdcall GetNominalBPPressureByDBV(void) Returns nominal (regular operation) pressure in train brake pipe.
SetFastFill	HRESULT _stdcall SetFastFill([in] VARIANT_BOOL aState) If aState = True then this function sets release mode for the driver’s brake valve. The current position of driver’s barke valve is ignored. Can be used for example, to model a “fast fill” button for old locomotives which have no the “release” position on driver’s brake valves. If aState = False then the driver’s brake valve works according to the current position.
GetFastFill	VARIANT_BOOL _stdcall GetFastFill(void) Returns the state of “fast fill” button.
GetWSMotorForce	double _stdcall GetWSMotorForce([in] int aWSIndex) Returns the applied motor force for the wheelset defined by wheelset index aWSIndex. Depending on the mode, it can be

	traction or ED brake force. This force value takes into account adhesion limit.
GetWSMotorPotentialForce	double _stdcall GetWSMotorPotentialForce([in] int aWSIndex) Returns the potential motor force for the wheelset defined by wheelset index aWSIndex. Depending on the mode, it can be traction or ED brake force. This force value does not take into account adhesion limit.
SetDynBrakeEnabled	HRESULT _stdcall SetDynBrakeEnabled([in] VARIANT_BOOL aState); Sets dynamic brakes enabled.
GetDynBrakeEnabled	VARIANT_BOOL _stdcall GetDynBrakeEnabled(void); Returns if dynamic brake enabled or not.
SetBrakeValveEnabled	HRESULT _stdcall SetBrakeValveEnabled([in] VARIANT_BOOL aState); Sets brake valve enabled.
GetBrakeValveEnabled	VARIANT_BOOL _stdcall GetBrakeValveEnabled(void); Returns if brake valve enabled or not.
GetDieselRPM	double _stdcall GetDieselRPM(void); Returns diesel RPM.
GetERAdjustmentPressure	double _stdcall GetERAdjustmentPressure (void); Returns adjustment pressure for equalizing reservoir.
SetERAdjustmentPressure	HRESULT _stdcall SetERAdjustmentPressure([in] double aPressure); Sets the adjustment pressure for equalizing reservoir.
GetFlowRateVariable	VARIANT_BOOL _stdcall GetFlowRateVariable(void); Returns information for user, is it specified by the construction of the driver’s brake valve to change flow rate on the current position.
GetFlowRatio	double _stdcall GetFlowRatio(void); Returns the flow rate for current driver’s brake valve position. By default, this value is 1.
SetFlowRatio	HRESULT _stdcall SetFlowRatio([in] double aRatio); Sets the flow rate for current driver’s brake valve position.
GetTargetBPMaxPressure	double _stdcall GetTargetBPMaxPressure (void); Returns the maximal target brake pipe pressure for current driver brake valve position. Used for position when the pressure in brake pipe depends on the angle of rotation of brake valve handle.
GetTargetBPMinPressure	double _stdcall GetTargetBPMinPressure (void); Returns the minimal target brake pipe pressure for current driver brake valve position. Used for position when the pres-

	sure in brake pipe depends on the angle of rotation of brake valve handle.
GetFlowRatioMax	double _stdcall GetFlowRatioMax (void); Returns the maximal flow rate ratio for current driver brake valve position. Used for position when the flow rate to and from brake pipe in brake and release modes depends on the angle of rotation of brake valve handle.
GetFlowRatioMin	double _stdcall GetFlowRatioMin (void); Returns the mimnimal flow rate ratio for current driver brake valve position. Used for position when the flow rate to and from brake pipe in brake and release modes depends on the angle of rotation of brake valve handle.
SetDBVClosed	HRESULT _stdcall SetDBVClosed([in] VARIANT_BOOL aState); Close or opens the connection between brake pipe and driver brake valve.
GetDBVClosed	VARIANT_BOOL _stdcall GetDBVClosed(void); Returns if the connection between brake pipe and driver brake valve is closed or not.
GetMotorTractionCurrent	double _stdcall GetMotorTractionCurrent ([in] long Index); Output: electrical current [A] of a traction motor number Index in traction mode.
GetMotorDynBrakeCurrent	double _stdcall GetMotorDynBrakeCurrent ([in] long Index); Output: electrical current [A] of a traction motor number Index in brake mode.
GetERClosed	VARIANT_BOOL _stdcall GetERClosed(void); Returns if equalizing reservoir is closed i.e. no connection to any other device and pressure in equilizing reservoir keeps constant.
SetERClosed	HRESULT _stdcall SetERClosed([in] VARIANT_BOOL aState); Close or open connection of equilizing reservoir to any other device. If equilizing reservoir is closed its pressure keeps constant.
GetERClosed	VARIANT_BOOL _stdcall GetERClosed(void); Returns if equalizing reservoir is closed i.e. no connection to any other device and pressure in equilizing reservoir keeps constant.
SetERClosed	HRESULT _stdcall SetERClosed([in] VARIANT_BOOL aState); Close or open connection of equilizing reservoir to any other device. If equilizing reservoir is closed its pressure keeps constant.

SetMotorVoltage	HRESULT _stdcall SetMotorVoltage([in] double Value); Sets electrical motor entrance voltage by Value in V
GetEnergyConsumption	double _stdcall GetEnergyConsumption(void); Returns electric energy consumption of all traction motors of loco in traction mode, in kWh
GetEnergyRecuperation	double _stdcall GetEnergyRecuperation(void); Returns electric energy recuperation of all traction motors of loco in dynamic brake mode, in kWh
SetLineVoltage	HRESULT _stdcall SetLineVoltage([in] double Value); Sets line voltage by Value in kV
GetEntranceCurrent	double _stdcall GetEntranceCurrent(void); Returns entrance current of loco, in A Entrance current is calculated as: $I_e = \sum_i^N A_i \cdot (Notch^{a_{i1}} \cdot Speed^{a_{i2}} \cdot CM1^{a_{i3}} \cdot LineV^{a_{i4}})$, where *Notch* is a notch position, *Speed* is speed of the locomotive, *CM1* is current on the first working motor, *LineV* is line voltage. Coefficients of polinomial interpolation are stored in ECI file, which has the following format: $\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} & A_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & A_2 \\ & & \cdots & & \\ a_{N1} & a_{N2} & a_{N3} & a_{N4} & A_N \end{matrix}$ "entrancecurrentfile" parameter of TMC file shows to UMComSolver where shall it get polinomial interpolation coefficients for calculation of entrance current.
SetTransmissionMode	HRESULT _stdcall SetTransmissionMode([in] long Value); Traction motors are connected in a circuit with the main generator/alternator. The circuit may be series, series/parallel or parallel, but series/parallel and parallel circuits are more common. Each type of circuit has different voltage and current characteristics. Input: 0 corresponds to MANUAL transmission 1 corresponds to AUTOMATIC transmission Returns S_False is no transmission mode is available.
GetTransmissionIndex	int _stdcall GetTransmissionIndex(void); Returns current transmission index. Index starts with 0.
SetTransmissionIndexUp	int _stdcall SetTransmissionIndexUp(void); Increases transmission index.

	Output: resulting transmission index. Transmission index will not be changed if no upper transmission positions.
SetTransmissionIndexDown	int _stdcall SetTransmissionIndexDown(void); Decreases transmission index. Output: resulting transmission index. Transmission index will not be changed if no lower transmission positions.
GetTransmissionLosses	double _stdcall GetTransmissionLosses; Output: returns transmission losses of a diesel loco traction motors equipped by hydro-turbine transmission, %
GetMotorLosses	double _stdcall GetMotorLosses ([in] long Index); Output: transmission losses of a traction motor number Index, %
GetDieselPower	double _stdcall GetDieselPower; Output: returns indicated diesel power percentage, 100%

20.6.7. IUMCom3DTrainVehicle interface

IUMCom3DtrainVehicle is a 3D model of railway vehicle. The interface is used for getting position, acceleration of a car body of 3D vehicle, wheelset rotation data.

Interface: *IUMCom3DtrainVehicle*
Hierarchy: *Iunknown – IUMComTrainVehicle – IUMCom3DtrainVehicle*

Methods	Description
GetCarBodyAcceleration	HRESULT _stdcall GetCarBodyAcceleration([in] int SC_ID, [in] double X, [in] double Y, [in] double Z, [out] double * AX, [out] double * AY, [out] double * AZ); Returns accelerations (m/s2) along to X, Y and Z (AX, AY and AZ parameters correspondingly) of the point with coordinates specified by X, Y and Z parameters. SC_ID specifies system of coordinate to resolve accelerations: 0 – inertial system of coordinates, 1 – car-body-fixed system of coordinates, 2 – way-fixed system of coordinates, 3– modified way-fixed system of coordinates.
GetCarBodyPosition	HRESULT _stdcall GetCarBodyPosition([in] int SC_ID, [out] double * X, [out] double * Y, [out] double * Z, [out] double * a11, [out] double * a12, [out] double * a13, [out] double * a21, [out] double * a22, [out] double * a23, [out] double * a31, [out] double * a32, [out] double * a33); Returns position of origin (m) and components of rotation cosine matrix of CarBody coordinate system relative to the specified SC: SC_ID =0 – SC0, SC_ID =2 – relative to way-fixed system of coordinates, SC_ID =3 – relative to modified way-fixed system of coordinates. Output: X, Y, Z : coordinates of CarBody SC in specified SC; a11, a12, a13, a21, a22, a23, a31, a32, a33 – components of rotation cosine matrix. Example (SC_ID = 0): G1 = A10*G0 , where $$A_{10} = \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix},$$ G0 – projection of vector G to SC0, G1 – projection of vector G to SC1.
GetCarBodyHPR	HRESULT _stdcall GetCarBodyHPR([in] int SC_ID, [out] double * h, [out] double * p, [out] double * r);

	Output: Orientation of car body (degree) relative to the specified SC: SC_ID =0 – SC0, SC_ID =2 – relative to way-fixed system of coordinates, SC_ID =3 – relative to modified way-fixed system of coordinates. h : heading (yaw) (Z axis) p : pitch (Y axis) r : roll (X axis)
GetCarBodyAngVelocity	HRESULT _stdcall GetCarBodyAngVel([in] int SC_ID, [in] double OmX, [in] double OmY, [in] double OmZ); Output: Angular velocity of car body (rad/s). SC_ID specifies system of coordinate to resolve accelerations: 0 – inertial system of coordinates, 1 – car-body-fixed system of coordinates, 2 – way-fixed system of coordinates.
GetWheelsetCount	long _stdcall GetWheelsetCount(void); Intput: number of wheelsets in current rail vehicle
GetWheelsetSlipping	double _stdcall GetWheelsetSlipping([in] long Index); Input: Index = 0.. GetWheelsetCount-1 – index of wheelset Output: angular velocity of wheelset slipping, rad/s (OM-V/R), where OM – angular spinning velocity, V – speed, R – radius Returns zero if Index is out of range. Note. Wheel slipping can change from minus infinity to plus infinity
GetWheelsetSlippingPercentage	double _stdcall GetWheelsetSlippingPercentage([in] long Index); Input: Index = 0.. GetWheelsetCount-1 – index of wheelset Output: percent of wheelset slipping, % (OM-V/R)/(V/R)*100, where OM – angular spinning velocity, V – speed, R – radius Returns zero if Index is out of range. Note. Wheel slipping can change from minus infinity to plus infinity
GetWheelsetSpinVelocity	double _stdcall GetWheelsetSpinVelocity([in] long Index); Input: Index = 0.. GetWheelsetCount-1 – index of wheelset Returns zero if Index is out of range. Output: angular velocity of wheelset, rad/s
GetWheelNormalForce	double _stdcall GetWheelNormalForce([in] int WheelSetIndex, [in] int WheelIndex); Output: normal force at the first contact point of wheel with rail, Newton Input: WheelSetIndex = 0.. GetWheelsetCount-1 – index of

	wheelset (WheelIndex = 1) for left wheel, (WheelIndex = 2) for right wheel. Output: angular velocity of wheelset, rad/s Returns zero if WheelSetIndex or WheelIndex are out of range.

An example of overturning control procedure:

```
function IsOverturning: Boolean;
var i: integer;
  ComTrain: IUMComTrain;
  l3DVehicle: IUMCom3DtrainVehicle;
  _H, _P, _R, lSumNLeft,lSumNRight: double;
Begin
   ComTrain.GetVehicle3DByIndex(0, P);
   l3DVehicle:= IUNknown(P) as IUMCom3DtrainVehicle;
   l3DVehicle.GetCarBodyHPR(_H,_P,_R);
   Result:= (_R>8);  // Check for inadmissible inclination of CarBody
   if not Result then begin
      lSumNLeft:= 0;      lSumNRight:= 0;
      for i:= 0 to l3DVehicle.GetWheelsetCount - 1 do begin
         lSumNLeft:= lSumNLeft + l3DVehicle.GetWheelNormalForce(i,1);
         lSumNRight:= lSumNRight + l3DVehicle.GetWheelNormalForce(i,2);
      end;
      Result:= ((lSumNLeft=0) or (lSumNRight=0));
   end;
end;
```

20.6.8. IRailRoad interface

IRailRoad is an interface for work with the rail road model. The interface is used for loading and checking of railroad XML file, setting of initial position of train on the railroad, irregularity type, train direction and states of switches.

IRailRoad interface supports two types of files: (1) text XML files and (2) binary RRD files. Binary RRD file format helps to decrease reading and parsing time.

Note. Initial position is a desired position of the front point of the train at the moment of test start. In effect simulation of train motion starts not from the initial position, but from the position evaluated from it with a glance of parameters of the train. This stage of motion, so-called “start mode stage”, is necessary to realize moving in the railroad geometry and track irregularities.

IRailRoad interface enables organization of virtual train motion, Sect. 20.6.2. *"IVirtualTrain interface"*, p. 20-31.

Interface: *IRailRoad*

Hierarchy: *IUnknown – IRailRoad*

Methods	Description
ReadFromFile	HRESULT _stdcall ReadFromFile([in]LPSTR FileName, [in] VARIANT_BOOL CanRepair); Loads and checks railroad description from specified *.xml or *.rrd file. Tries to repair detected errors if CanRepair is true. Returns S_OK in successful termination, S_FALSE in case of non-successful termination.
GetXMLFilePath	LPSTR _stdcall GetXMLFilePath(void); Returns path to the loaded railroad *.xml file.
GetElementCount	int _stdcall GetElementCount (void); Returns count of railroad elements.
GetRoadCount	int _stdcall GetRoadCount (void); Returns count of roads.
GetSwitchCount	int _stdcall GetSwitchCount (void); Returns count of switches.
InitTrack	HRESULT _stdcall InitTrack([in] int InitialElementID, [in] int InitialSectionID, [in] double InitialPosition, [in] int IrregularityType, [in] VARIANT_BOOL PositiveDirection, [in] double ScaleFactorY, [in] double ScaleFactorZ); Setting of initial position, track irregularities type and train direction before simulation start. Returns S_OK in successful termination, S_FALSE in case of non-successful termination. If InitialPosition is greater than element section length or less than zero the result is S_FALSE. Input:

	InitialElementID is the global ID of initial element; InitialSectionID is ID of active section of initial element; InitialPosition is initial local position on active section of initial element in meters; IrregularityType is the index of track irregularity type (0 – generated from UIC spectra for bad maintenance track, 1 – generated from UIC spectra for good maintenance track, 2 – special track, 3 – even track; 4 – loaded from way files, see ReadIrregularityFromFile method); PositiveDirection is flag of train direction; ScaleFactorY is horizontal irregularity scale factor; ScaleFactorZ is vertical irregularity scale factor.
GetStartPosition	HRESULT _stdcall GetStartPosition ([out] int * StartElementID, [out] int * StartSectionID, [out] double* StartModeTracklength); Returns S_OK in case of successful start position evaluation, S_FALSE in other case of non-successful evaluation. Output: StartElementID is global ID of element from which train moving starts; StartSectionID is ID of active section of start element; StartModeTrackLength is length of track from start position to the initial one.
GetStartMode	VARIANT_BOOL _stdcall GetStartMode(void); Returns start mode state during simulation: true – train moving to the initial position, false – initial position is arrived, test is started.
SetSwitchState	HRESULT _stdcall SetSwitchState([in] int SwitchID, [in] int SwitchFlag); Changes the state of switch if it is not under train. Returns S_OK if the state is set, S_FALSE if switch is not found or its state cannot be changed at the moment. Intput: SwitchID is global ID of the switch, SwitchFlag is flag of desired switch state (0 – switch is set, 1 – switch is not set).
GetSwitchState	long _stdcall GetSwitchState ([in] int SwitchID); Input: SwitchID – global ID of switch Output: Returns flag of current state of switch (0 – switch is set, 1 – switch is not set, (-1) – switch is not described).
GetActiveTrackElementList	LPSTR _stdcall GetActiveTrackElementList([in] int StartElementID, [in] int StartSectionID, [in] VARIANT_BOOL ForwardDirection); Returns the string with sequence of '[ElementID; SectionID]' records from the start position up to the end of railroad track.

SetDrawingMode	HRESULT _stdcall SetDrawingMode([in] int DrawingModeIndex); Changes drawing mode of railroad in animation windows, see Sect. 20.2. *"IUMObject interface"*, p. 20-5. Input: DrawingModeIndex = 0 – railroad image is generated, all railroad elements are displayed at once; DrawingModeIndex = 1 – railroad image is generated, railroad element becomes visible when first point of train reaches; DrawingModeIndex = 2 or any other value – railroad image is not generated
SetScaleFactors	HRESULT _stdcall SetScaleFactors([in] double ScaleFactorY, [in] double ScaleFactorZ); Changes scale factors of horizontal and vertical track irregularities. Input: ScaleFactorY is horizontal irregularity scale factor; ScaleFactorZ is vertical irregularity scale factor. These parameters are multipliers (1.0 value means that real scale of generated or read from file irregularities will be used).
ReadIrregularityFromFile	HRESULT _stdcall ReadIrregularityFromFile([in] LPSTR FileName, [in] int AType); Reads track irregularity from preliminary created track irregularity (*.way) file. Returns S_OK if file is read, S_FALSE in other case. Input: FileName is path to the (*.way) file; AType is flag of rail and axis: 0 – vertical irregularities of left rail ; 1 – vertical irregularities of right rail ; 2 – horizontal irregularities of left rail; 3 – horizontal irregularities of right rail;
HideProgressBars	HRESULT _stdcall HideProgressBars ([in] VARIANT_BOOL OnHide); If OnHide = true then progress bars of railroad reading and irregularity generation are not displayed. OnHide = false by default.
GetElementDataByIndex	HRESULT _stdcall GetElementDataByIndex([in] int aIndex, [out] int* aGlobalID, [out] int* aElementTypeIndex, [out] int* aSwitchTypeIndex); Detects parameters of railroad element by its serial index in the railroad description. Returns S_OK if aIndex value is in the [0, GetElementCount-1] range, S_FALSE in other case. Outputs: aGlobalID is global ID of the railroad element aElementTypeIndex is flag of the railroad element type: 0 – Road;

	1 – Switch; aSwitchTypeIndex is flag of the switch type: -1 – the element is Road; 0 – B-Switch; 1 – C-Switch; 2 – S-Switch; 3 – V-Switch.
RefineRailroadFile	HRESULT _stdcall RefineRailroadFile ([in]LPSTR aSource, [in] LPSTR aDest); Loads railroad description from specified by aSource *.xml or *.rrd file, verify and automatically correct data, and save the refined description to aDest *.xml or *.rrd file. Returns S_OK in successful termination, S_FALSE in case of non-successful termination. This function can be used for XML to RRD and RRD to XML file format convertions.
SetDetailedLog	HRESULT _stdcall SetDetailedLog([in] VARIANT_BOOL aDetailedLog); Effects on content of log file during working ReadFromFile and RefineRailroadFile. Set aDetailedLog to TRUE to generate detailed log file. Set aDetailedLog to FALSE to generate brief log file. By deault brief log file is generated to provide minimal CPU efforts for reading railroad file.
Interfaces for virtual train models, Sect. 20.6.2. "*IVirtualTrain interface*", p. 20-31.	
AddVirtualTrain	HRESULT _stdcall AddVirtualTrain([out] void* Train); Adds new virtual train to a model. Returns the interface of the virtual train.
GetVirtualTrainCount	int _stdcall GetVirtualTrainCount(void); Returns the count of created virtual trains.
GetVirtualTrainByIndex	HRESULT _stdcall GetVirtualTrainByIndex ([in] int Index, [out] void* Train); Returns the virtual train interface by train index. First item has index 1.
DeleteVirtualTrainByIndex	HRESULT _stdcall DeleteVirtualTrain([in] int Index); Deletes the virtual train by index. First item has index 1.
SetUseRRSpecificIrrScale	HRESULT _stdcall SetUseRRSpecificIrrScale([in] VARIANT_BOOL aUseRRSpecificIrrScale); Enables track irregularities scaling by IrrScaleZ, IrrScaleY factors defined in the railroad description file.
SetRRElementIrrScales	HRESULT _stdcall SetRRElementIrrScales([in] int aElementID, [in] double aIrrScaleZ, [in] double aIrrScaleY); Set railroad element specific track irregularitiy scales. Note: Use SetUseRRSpecificIrrScale method to enable the rail-

	road specific scaling. Warning: Changing of the irregularities scales of the element the 3D vehcile is currently placed on during simulation can result in dramatic dynamic effects.

20.6.9. Data file description

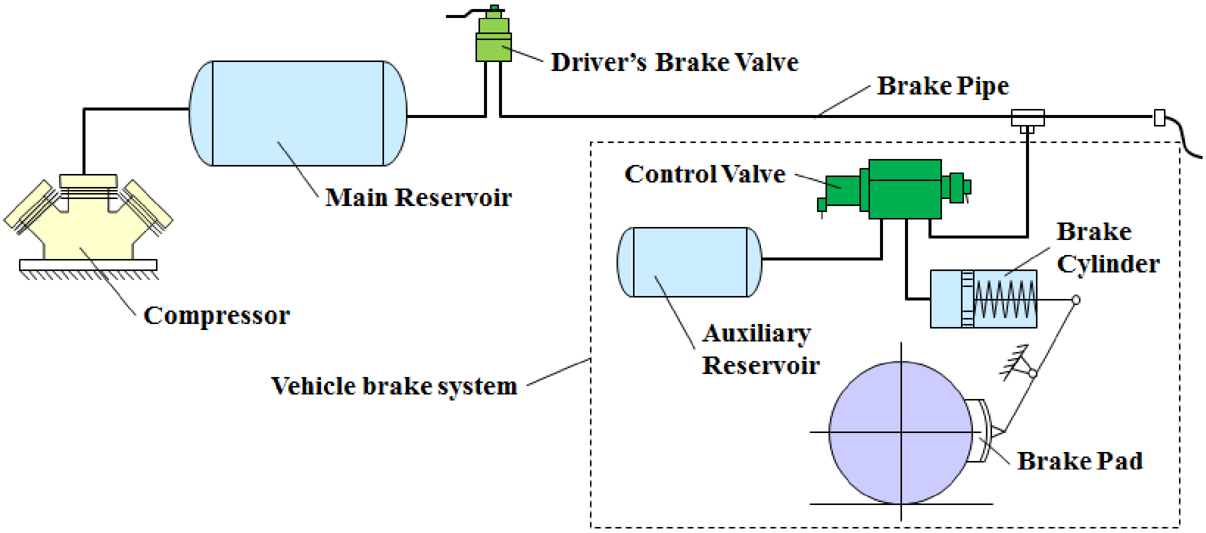


Figure 20.1. Brake system diagram

20.6.9.1. *.pf file description

**.pf-files* describe parameters of brake's leverage mechanism in between a brake cylinder and brake pads, see figure 20.2.

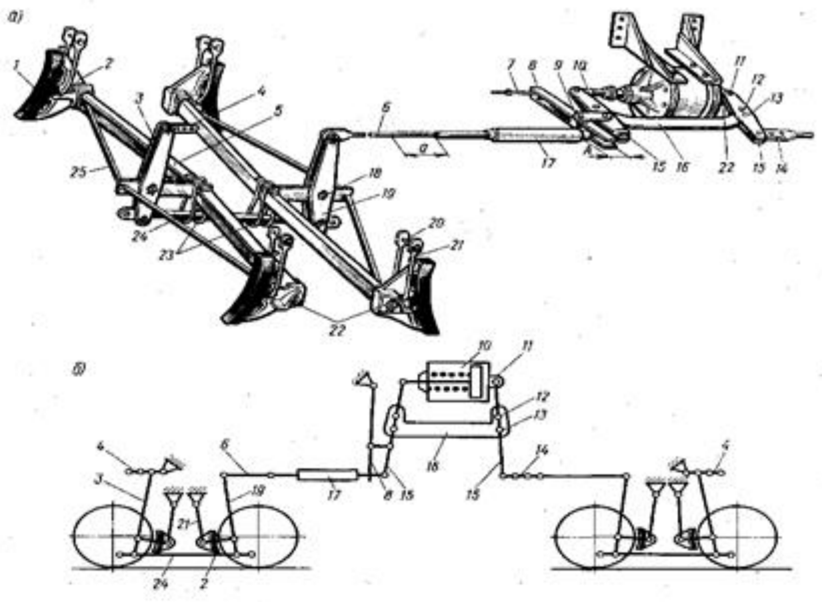


Figure 20.2. Typical brake leverage mechanism for a freight car

Normal force between a brake pad and a wheel (N) is calculated according to the following formula:

$$N = C_{eff} PSRL_{eff} - F_{spr}, \text{ where}$$

C_{eff} is the dimensionless efficiency factor for the brake cylinder (see *cylindereff* parameter),

P is the current pressure in the brake cylinder (Pa),

S is the piston square, see *pistonsquare* parameter (m^2),

R is the dimensionless leverage ratio between a brake cylinder and a brake pad, see *gearratio* parameter,

L_{eff} is the dimensionless efficiency factor for the leverage system, see *levereff* parameter,

F_{spr} is the spring force that releases brake pads when there is no pressure in a brake cylinder, see *springforce* parameter (N).

Potential (maximum) braking (friction) force for a contact pair (F_{fr}) (brake shoe/disk vs. wheel) is calculated according to the following formula:

$$\mathrm{F_{fr}} = \mathrm{fN}\frac{\mathrm{r}}{\mathrm{R}}, \text{ where}$$

f is the current dimensionless friction coefficient,

N is the normal force in the contact pair (N),

is the ratio of the arm of braking force (r) to the wheel radius (R). It is applicable for disk brakes only. It is 1 for shoe brakes, see *radiusratio* field in **.pf-file*.

Potential (maximum) braking (friction) force for a whole vehicle (F_{car}) is calculated according to the following formula:

$$F_{car} = nF_{fr}, \text{ where}$$

n is the number of contact pairs per vehicle, see *pairnumber* parameter,

F_{fr} is the friction force per contact pair.

Field	Description
forcemode	Type: integer; It is reserved for future use. This parameter should be set to 1.
pairnumber	Type: integer; Count of contact pairs per vehicle
radiusratio	Ratio (r/R) of the arm of braking force (r) to the wheel radius (R). It is applicable for disk brakes only. It is 1 for shoe brakes. See Figure 20.3 for details.
cylindernumber	Type: integer; Count of brake cylinders per vehicle.
pistonsquare (S)	Square of a brake cylinder piston (m^2)
cylindereff (C_{eff})	Efficiency factor for the brake cylinder
gearratio	Leverage ratio between a brake cylinder and a brake pad
levereff (L_{eff})	Efficiency factor for the leverage system
springforce (F_{spr})	Force in the prestressed spring that releases brake pads (N)

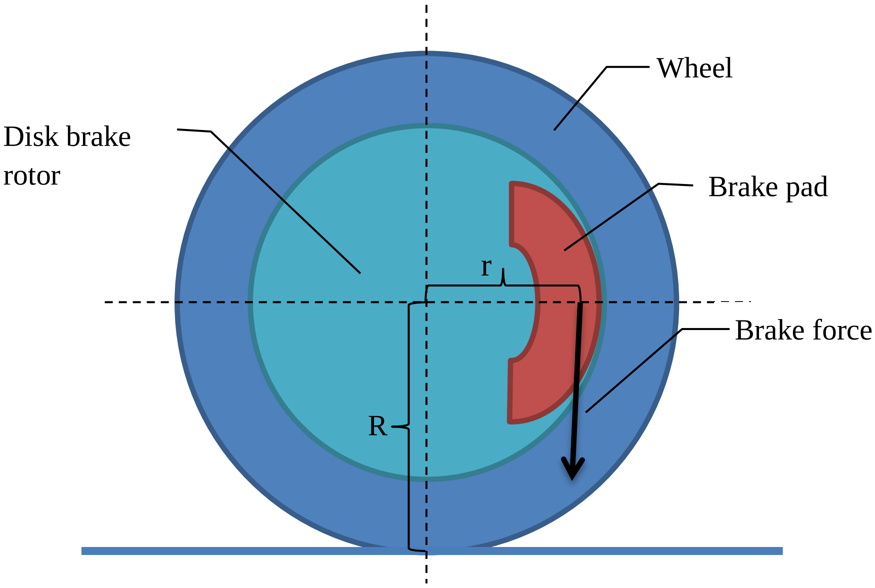


Figure 20.3. On braking force radius ratio (r/R) definition

20.6.9.2. Cars//input.dat file description*

Identificator	**Description**
AxleOver	
BodyLength	
BodyZ	
BogieBase	
CouplingBase	
CouplingHeight	
CouplingLength	
CouplingOver	
CouplingPoint	
Mass	
resistance_scale	
VehicleBase	
VehicleHeight	
VehicleWidth	
vertical_mass_center_position	
WheelBase	
WheelRadius	
wheelset_count	

20.7. Parameters of railway vehicles

20.7.1. Diesel model

Diesel parameters are described in **.dm* (**d**iesel **m**odel) files that are located in *'..\Train\Diesel'* folder. Typical **.dm* file includes the following fields:

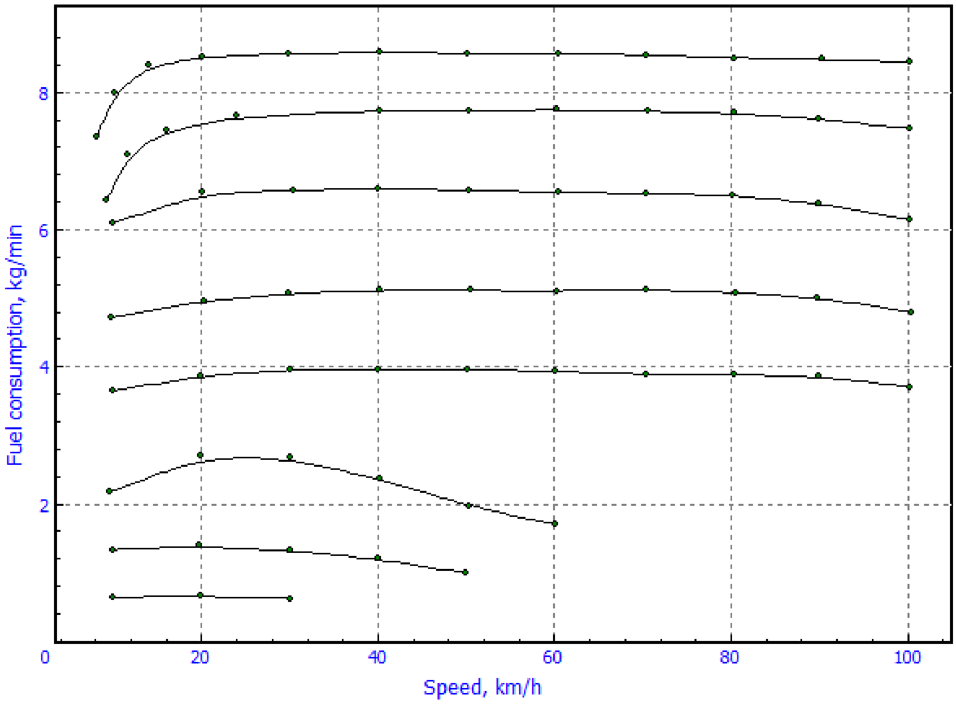
```
name="Diesel model for DE33000";
comment="Diesel model for DE33000";
idlefuelconsumption=0.189;
factor=1.0;
FuelConsumptionCurves=DE33000Fuel.crv;
RPMCurve=DE33000RPM.crv;
```

IdleFuelConsumption sets the fuel consumption for the idle mode of the engine in kg/min.

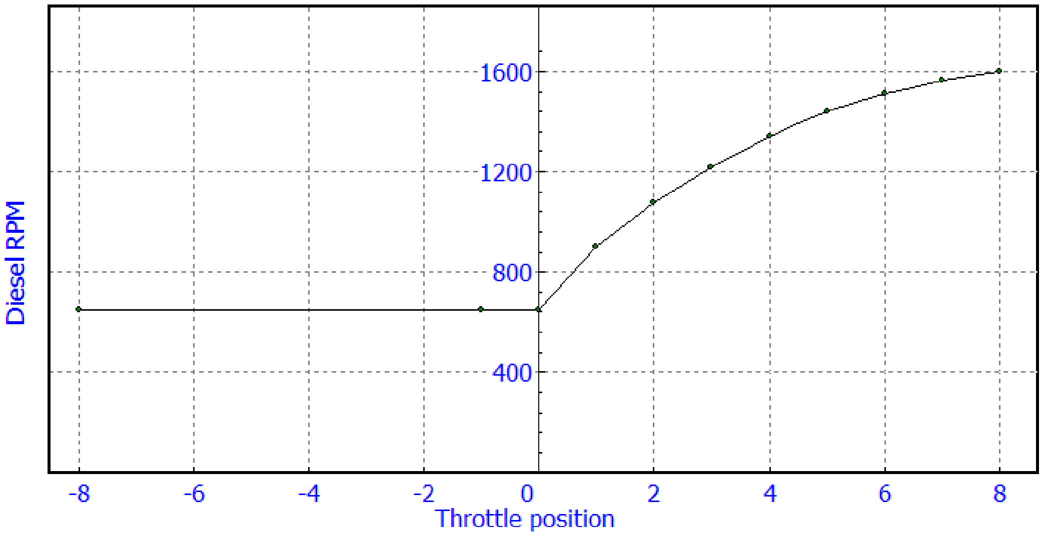
Field *factor* sets the multiplication factor to easily adjust some nominal fuel consumption curves for better agreement with field tests when available. *Factor* value does not effect on idling fuel consumption that should be adjusted independently.

FuelConsumptionCurves refers to a file where fuel consumption diagram(s) are described. This file should be located in the same folder. This file describes the fuel consumption diagram as a dependence between current vehicle speed in km/h and fuel consumption in kg/min for several given throttle positions or just for the only maximal throttle position as shown in the figure below.

Please note that the count of curves that describe fuel consumption should be (1) equal to count of throttle curves then fuel consumption is calculated according to the given throttle position or (2) should be 1 then the only curve describes the maximal fuel consumption and intermediate values are calculated proportionally.



RPMCurve refers to a file where diesel RPM diagram versus throttle position is described. This file should be located in the same folder. This file describes the diesel RPM diagram as a dependence between throttle position and diesel RPM (rotations per minute) as shown in the figure below. Zero throttle position on the diagram refers to idle mode RPM; negative values – to dynamic brake positions.



20.8. IUMEventHandler interface

Interface: *IUMEventHandler*

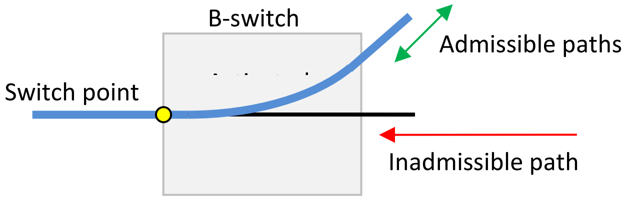
Hierarchy: *IUnknown – IRailRoad*

IUMEventHandler is an interface that is intended to handle the errors that arise during the simulation process. Object of this interface should be described within the client application and assigned with the UM COM server via the following method of the *IUMObject* interface:

HRESULT _stdcall SetEventHandler([in] IInpObjectEventHandler EventHandler)*

Methods	Description
OnError	HRESULT _stdcall OnError([in] int ErrorCode, [in] int Tag, [in] LPSTR ErrorMessage); This method is called by *IUMObject* interface if the event handler is assigned. Please check the *ErrorCode* and *Tag* in the table below.

Error codes of *OnError* method:

Error codes	Tag	Description
0	0	Empty
1	<Switch GlobalID>	Railroad model message: 'Error! Switch [<Switch GlobalID>] was corrupted!' Message is generated if train or virtual train passes switch point of the switch element in direction not allowed by current state of the switch. Example: Admissible/Inadmissible paths throw the activated B-Switch 
2	<Switch GlobalID>	Railroad model message: 'Error! Cannot change state of switch [<Switch GlobalID>]. Switch in under train [<Train Caption>]!'

20.9. Interfaces for simulator of road vehicles

General information about loading a UM car model, preparing, starting and finishing simulation process and so on can be found Sect. 20.2. *"IUMObject interface"*, p. 20-5. Here we consider an interface for control of a vehicle based on simulator output data as well as for getting vehicle specific kinematic and dynamic performances.

20.9.1. IComCar interface

Interface: IComCar

Hierarchy: Iunknown – IcomCar

Methods	Description
SteeringAngle	HRESULT _stdcall SteeringAngle([in] double Value); Input: value of steering wheel rotation in degrees.
ThrottlePosition	HRESULT _stdcall ThrottlePosition([in] double Value); Value∈[0, 100%] Input: value of the engine throttle position in percent base on the accelerator pedal position.
ClutchPedalPosition	HRESULT _stdcall ClutchPedalPosition([in] double Value) Value∈[0, 1] Input: value of the clutch pedal position (0- no pressure on the pedal, 1 – fully pressed pedal).
GearPosition	HRESULT _stdcall GearPosition([in] int Value); Input: value of the gear position. -1 – reverse 0 – neutral 1,2.. – I, II … gears for forward movement
BrakePedalPosition	HRESULT _stdcall BrakePedalPosition([in] double Value); Value>0 Sets applied brake pedal force in N
HandBrakePosition	HRESULT _stdcall HandBrakePosition([in] double Value); Value∈[0, 1] Input: value of the hand brake lever position (0 – no braking).
ABSState	int _stdcall ABSState(void); Output: -1: ABS is not presented 0: ABS is not active 1: ABS operates
VehicleSpeed	HRESULT _stdcall VehicleSpeed([out] double * Value); Output: vehicle speed in km/h
EngineRPM	HRESULT _stdcall EngineRPM([out] double * Value); Output: ICE shaft angular velocity in rpm

GetCarBodyPosition	HRESULT _stdcall GetCarBodyPosition([out] double * X, [out] double * Y, [out] double * Z, [out] double * Yaw, [out] double * Pitch, [out] double * Roll); Output Cartesian coordinates of the car body center of gravity in meters: X(longitudinal), Y(lateral, left positive), Z(vertical, upward positive) Orientation angles in degrees (yaw, pitch, roll)
GetWheelPosition	HRESULT _stdcall GetWheelPosition([in] int Index, [out] double * X, [out] double * Y, [out] double * Z, [out] double * AxisX, [out] double * AxisY, [out] double * AxisZ); Input: index of wheel (see figure) Index= 1 (front left) 2 (front right) 3 (rear left) 4 (rear right) Output: wheel position and orientation Cartesian coordinates of the car body center of wheel in meters: X(longitudinal), Y(lateral, left positive), Z(vertical, upward positive) Components of unit vector along the wheel rotation axis in SC0 (left positive, see figure) AxisX, AxisY, AxisZ
GetSteeringState	int _stdcall GetSteeringState(void); Output: 1 – driver controls steering wheel angle; 0 – steering wheel is free.
SetSteeringState	HRESULT _stdcall SetSteeringState([in] int AState); Input: 1 – driver controls steering wheel angle; 0 – steering wheel is free.
GetSteeringAngle	HRESULT _stdcall GetSteeringAngle([out] double* Value); Output: Value – angle of steering wheel rotation in degrees. If steering state corresponds to the driver control, the result is equal to the value specified by the SteeringAngle method. If the steering wheel is free, the output value is equal to the dynamically computed angle.
SetRoadGeometry	HRESULT _stdcall SetRoadGeometry([in] int Iwheel, [in] double Z, [in] double NormalX, [in] double NormalY, [in] double NormalZ); Input: Iwheel – index of wheel; Z (m) – verticut road coordinate under the wheel; NormalX, NormalY, NormalZ – normal to the road surface under the wheel in SC0.

GetSteeringWheelTorque	HRESULT _stdcall GetSteeringWheelTorque([out] double* Value); Output: Value (Nm) – steering wheel torque
SetWindData	HRESULT _stdcall SetWindData([in] double Speed, [in] double Angle); Input: Speed (m/s) – wind speed; Angle (degrees) – wind direction angle relative to SC0 (see figure for positive direction), Sect. 20.9.3. *"Angle of wind direction"*, p. 20-90.
GetTravelledDistance	HRESULT _stdcall GetTravelledDistance([out] double* Value); Output: Value (v) – travelled distance
GetChassisAcceleration	HRESULT _stdcall GetChassisAcceleration([in] int SCType, [out] double* AX, [out] double* AY, [out] double* AZ, [out] double* EX, [out] double* EY, [out] double* EZ); Input: SCType – corrdinate system in which accelerations are computed: 0 – SC0; 1 – chassis-fixed SC Output: AX, AY, AX – components of acceleration of origin of chassis-fixed SC (m/s) ; EX, EY, EX – components of angular acceleration of chassis-fixed SC (rad/s^2);
SetRollingFriction	HRESULT _stdcall SetRollingFriction([in] double f0, [in] double k1, [in] double k2); Input: f0, k1, k2 – parameters for computation of coefficient of rolling friction
GetWheelVelocities	HRESULT _stdcall GetWheelVelocities([in] int Index, [out] double* VX, [out] double* VY, [out] double* VZ, [out] double* Omega); Input: Index – Index of wheel Output: VX, VY,VZ – components of velocity of the wheel center in SC0 (m/s); Omega – rolling angular vrlocity of wheel (rad/s)/
GetTireUnloadedRadius	HRESULT _stdcall GetTireUnloadedRadius([in] int Index, [out] double* Radius); Input: Index – index of wheel; Output: Radius – radius of undeformed wheel (m)
GetTireContactData	HRESULT _stdcall GetTireContactData([in] int Index, [out] double* Fx, [out] double* Fy, [out] double* Fz, [out] double* Mx, [out] double* My, [out] double* Mz, [out] double*

	SlipX, [out] double* SlipY); Input: Index – Index of wheel Output: Fx (longitudinal tire force: traction, braking), Fy (tire lateral force), Fz (tire normal force), Mx (torque about longitudinal axis); My (rolling resistance torque), Mz (aligning torque). Forces are measured in N, toques in Nm.
SetTerrainCurveCount	HRESULT _stdcall SetTerrainCurveCount([in] int Index, [in] int NPoints); Input: Index – index of wheel; NPoints – number of points in polygon specifying the terrain curve
SetTerrainCurvePoint	HRESULT _stdcall SetTerrainCurvePoint([in] int WheelIndex, [in] double X, [in] double Z, [in] double NormalX, [in] double NormalY, [in] double NormalZ); Input: Index – index of wheel; X, Y, Z – coordinates of a terrain curve point in SC of wheel (m); NormalX, NormalY, NormalZ –components of normal to the terrain curve section between the current and the previous point in polygon, see Sect. 20.9.4. *"Terrain curve"*, p. 20-91.
SetInitialSpeed	HRESULT _stdcall SetInitialSpeed([in] double Value); Input: Value – initial speed of vehicle (m/s);
SetTireUnloadedRadius	HRESULT _stdcall SetTireUnloadedRadius([in] int Index, [in] double Value); Input: Index – index of wheel; if Index=0 the value is assigned to all of the wheels; Value – radius of undeformed wheel (m)
SetTireSectionWidth	HRESULT _stdcall SetTireSectionWidth([in] int Index, [in] double Value); Input: Index – index of wheel; if Index=0 the value is assigned to all of the wheels; Value – tire section width (m)
SetTireContactType	HRESULT _stdcall SetTireContactType([in] int AType); Input: AType – type of tire/terrain contact model: 0 – single point contact; the terrain is set by SetRoadGeometry method 1 – multiple contact; the terrain is set by a terrain curve
TireBlowOut	HRESULT _stdcall TireBlowOut([in] int Index, [in] double RimRadius, [in] double DragCoefficient); Input: Index – index of wheel; RimRadius- radius of wheel rim (m);

	DragCoefficient – coefficient of drag of the blowout tire 0.3-0.4. See Sect. 20.9.5. *"Tire blowout"*, p. 20-94.
SetFrictionCoefficient	HRESULT _stdcall SetFrictionCoefficient([in] double PeakValue, [in] double SlidingValue); Adhesion coefficient for all of the tires. Input: PeakValue – the maximal value of coefficient; Sliding – the minimal value of coefficient by pure sliding. See Sect. 20.9.6. *"Road coefficients of friction"*, p. 20-94.
SetTireRatedPressure	HRESULT _stdcall SetTireRatedPressure([in] int Index, [in] double Value); Sets tire rated inflation pressure. Input: Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; Value – the tire inflation pressure (kPa).
SetTireVericalStiffness	HRESULT _stdcall SetTireVerticalStiffness([in] int Index, [in] double StiffnessZ, [in] double DampingZ, [in] double Pressure); Sets tire vertical stiffness and damping for the rated inflation pressure and load. Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; StiffnessZ– tire vertical stiffness, N/m; DampingZ– tire vertical damping constant, Ns/m.
SetTireCorneringStiffness	HRESULT _stdcall SetTireCorneringStiffness ([in] int Index, [in] double Value); Sets tire cornering stiffness for the rated inflation pressure and load. Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; Value– tire cornering stiffness, N/rad.
SetTireLongitudinalStiffness	HRESULT _stdcall SetTireLongitudinalStiffness([in] int Index, [in] double Value); Sets tire longitudinal stiffness for the rated inflation pressure and load. Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; Value– tire longitudinal stiffness, N.
SetAutoEvaluationTireStiffness	HRESULT _stdcall SetAutoEvaluationTireStiffness([in] int Vertical, [in] int Cornering, [in] int Longitudinal); Specifies automatic evaluation of tire stiffness properties according to simplifies analytic expressions. Set 1 to the corre-

	sponding direction to specify automatic evaluation. Input : Vertical – vertical tire stiffness; Cornering – cornering stiffness Longitudinal – longitudinal stiffness.
SetCorneringCoefficient	HRESULT _stdcall SetCorneringCoefficient([in] int Index, [in] double Value); Sets tire cornering coefficient for the rated inflation pressure and load used if Cornering=1 in procedure SetAutoEvaluation-TireStiffness Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; Value (0.07-0.2)– tire cornering coefficient, unitless.
SetTireDampingRatio	HRESULT _stdcall SetTireDampingRatio([in] int Index, [in] double Value); Specifies the vertical tire damping properties by damping ratio; Is used if Vertical=1 in procedure SetAutoEvaluation-TireStiffness Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; Value (0.3-0.75) – damping ratio.
SetTireRatedPressure	HRESULT _stdcall SetTireRatedPressure([in] int Index, [in] double Value); Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; Pressure– the rated value of pressure (kPa).
SetTireRatedLoad	HRESULT _stdcall SetTireRatedLoad([in] int Index, [in] double Value); Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; Value – the rated tire load (N);
EvaluationTireRatedStiffness	HRESULT _stdcall EvaluationTireRatedStiffness(void); The method computes stiffness parameters of tires specified by the SetAutoEvaluationTireStiffness procedure.
SetTireCurrentPressure	HRESULT _stdcall SetTireCurrentPressure([in] int Index, [in] double Pressure); Input: Index – index of wheel; if Index=0 the values are assigned to all of the wheels; Value– the rated value of pressure (kPa).
DoEquilibriumTest	HRESULT _stdcall DoEquilibriumTest([in] double TMax, [in] double StopEnergy); The method computes the equilibrium position of vehicle tak-

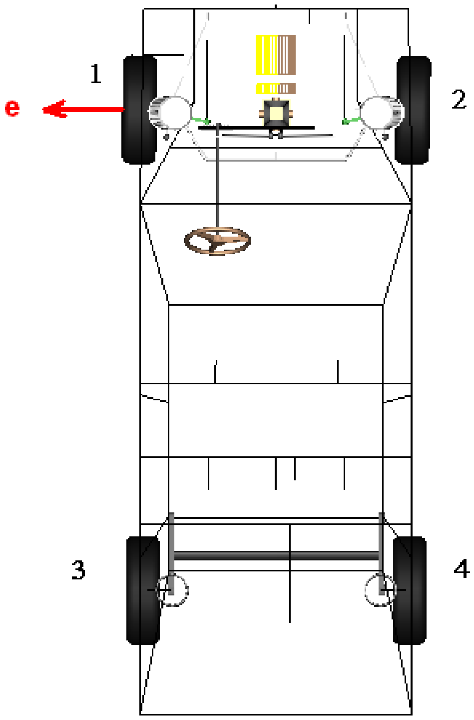
	ing into account the terrain geometry. Input : TMax>=20s the – maximal duration of simulation while computation the equilibrium StopEnergy > =1.-e-5 (J) – computation stops when the kinetic energy of the vehicle is less than the specified value/ See Sect. 20.9.9. *"Computation of vehicle equilibrium"*, p. 20-98.
SetSteadyTestType not used	HRESULT _stdcall SetSteadyTestType(void); The method sets the mode of simulation for computation of the car equilibrium position depending of occupation state and the current macro geometry data. Use TestFinish function to get the information success of the equilibrium computation process.
TestFinished not used	HRESULT _stdcall TestFinished(void); The function specifies the end of equilibrium computation in The procedure sets the simulator mode. The equilibrium position is recommended to be computed before start of the simulator.
SetSimulatorTestType not used	HRESULT _stdcall SetSimulatorTestType(void); The method sets the simulator mode. The equilibrium position is recommended to be computed before start of the simulator.
LoadRoadRoughness	HRESULT _stdcall LoadRoadRoughness([in] LPSTR FileLeft, [in] LPSTR FileRight); The method loads *.irr files with the left and right road roughness data. Input : FileLeft, FileRight – full paths to *.irr files with roughness functions. If the files are located in the vehicle model directory, FileLeft, FileRight may contain file names only (without direct path)
UseRoadRoughness	HRESULT _stdcall UseRoadRoughness([in] int Value); The methods sets usage of road roughness if Value=1. If Value=0, the road is ideal, and no roughness is taken into account.
SetTerrainRoughnessFactor	HRESULT _stdcall SetTerrainRoughnessFactor([in] double Factor, [in] double TransitionLength); The method specifies the terrain roughness level. Input: Factor– factor for increase/decrease the standard roughness level; TransitionLength – distance of transition to the new roughness, m. See Sect. 20.9.7. *"Rolling resistance of tires"*, p. 20-95.
CreateCollisionEvent	HRESULT int _stdcall CreateCollisionEvent([in] double Stiffness, [in] double DampingRatio, [in] double cFriction);

	The method creates a collision event between two bodies. One-point contact is allowed for an event. To handle multiple contacts, use different events. Return value: handle identifying the event Input: Stiffness (N/m) – stiffness constant of the contact DampingRatio – the damping ration of the collision spring cFriction – coeffitiant of friction in contact
ReleaseCollisionEvent	HRESULT _stdcall ReleaseCollisionEvent([in] int Handle); Use the method to release the event after the collision when the distance between the colliding bodies become big enough. Input: handle of the event returned by the CreateCollision-Event method
SetCollisionData	RESULT _stdcall SetCollisionData([in] int Handle, [in] double Delta, [in] double Xa, [in] double Ya, [in] double Za, [in] double VXb, [in] double VYb, [in] double VZb, [in] double NX, [in] double NY, [in] double NZ); The method must be called in the StartComputeForces method of the UM Event handler Input: handle of the event returned by the CreateCollision-Event method; Delta (m) – depth of penetration, negative if now penetration take place Xa, Ya, Za (m) – coordinates of contact point in SC of the car body; VXb, VYb, VZb (m/s) – velocti of contact point of external body in SC0; Nx, Ny, Nz – normal to the contact surface external to the car body in SC0
GetCollisionForce	HRESULT _stdcall GetCollisionForce([in] int Handle, [out] double* FX, [out] double* FY, [out] double* FZ); The method is used for getting the collision force applied to the external body. The method must be called in the SingleStepEnd method of the UM Event handler Input: handle of the event returned by the CreateCollision-Event method; FX, FY, FZ (N) – components of contact force in SC0.
GetCarBodyCGPosition	HRESULT _stdcall GetCarBodyCGPosition([in] double* XCG, [in] double* YCG, [in] double* ZCG, [in] int SCRef); Input: SCRef – reference frame 0: SC0, 1: local SC of car body.

	Output: XCG, YCG, ZCG (m) – coordinates of center of gravity of car body in the specified SC.
GetChassisInterface	HRESULT _stdcall GetChassisInterface([out] void* CarBody); Access to the interface of the chassis (the car body) Output: CarBody – IBody interface to the car body.
SetPebbleUnderTire	HRESULT _stdcall SetPebbleUnderTire([in] int Index, [in] double kp); The method dynamically sets a pebble under the tire during the simulation Input: Index – index of wheel; kp – pebble size factor See Sect. 20.9.13. *"Run over the pebble"*, p. 20-104.
GetICEngine	HRESULT _stdcall GetICEngine([out] void* AICEngine); Access to the interface of internal combustion engine Output: AICEngine – ICOMICEngine interface to the ICE, see Sect. 20.10. *"Interface for internal combustion engine (ICE)"*, p. 20-105.

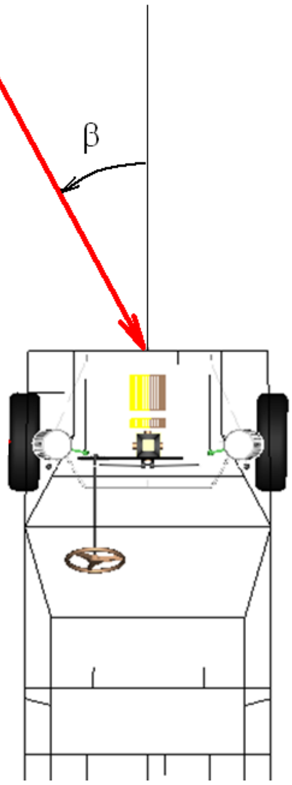
Most of the methods return S_OK in successful termination and S_FALSE in case of non-successful termination.

20.9.2. Indexing of wheels



Indexing of wheels. Wheel rotation vector **e**.

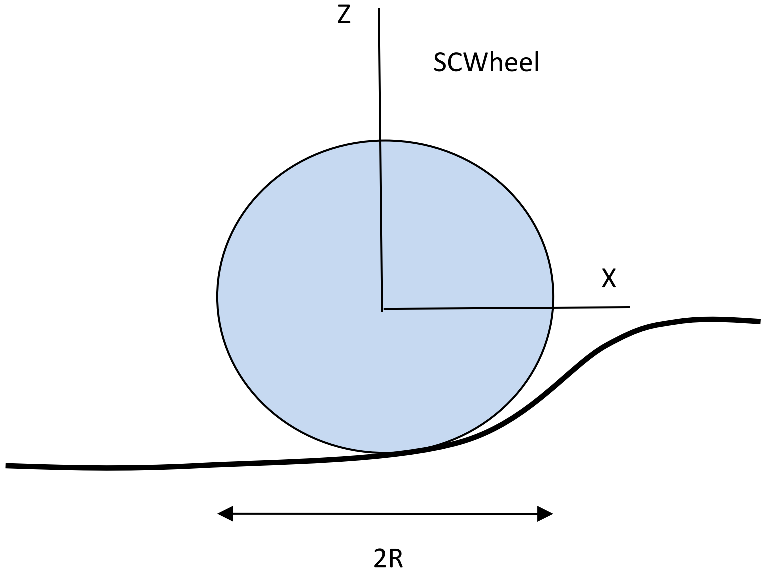
20.9.3. Angle of wind direction



Positive angle of wind speed direction

20.9.4. Terrain curve

20.9.4.1. Definition of terrain curve



The curve is a polyline, which is specified by a sequence of points in SCWheel. This system of coordinates coincides with the wheel plane, axis X is horizontal, axis Z is perpendicular to X. If necessary, normals to terrain (to triangles of the terrain surface) must be specified for each straight section of the polyline. The length of the curve must be approximately 2R.

To set the terrain curve application in tire-terrain contact evaluation, the SetTireContactType(1) method of ICComCar interface must be called before start the motion.

Terrain curve is sent to the solver in the **OnStartComputeForces** method of UMEventHandler. In this method, the following procedure of IComCar interface must be called for EACH of the wheels:

1. Call GetWheelPosition to obtain the position of the wheel.
2. Compute terrain curve for the givel wheel position.
3. Set number of points in the curve SetTerrainCurveCount(…)
4. Set the terrain curve points and normals in the loop by the method SetTerrainCurvePoint

A normal vector corresponds to the perpendicular to the terrain triangle specified by the current and previous points. The normal for the first point is ignored. The vectors are normalized in COM server.

Points must be ordered by increasing abscissa value (X).

20.9.4.2. Computation of unit vectors along SCWheel axes

Let e_x, e_y, e_z are the unit vectors along the x,y,z axes of SCWheel.

The components of the e_y vector in SC0 can be obtained directly by the call of the GetWheelPosition method. The following formulas specify other vectors:

$$\vec{e}_X = \frac{\vec{e}_y \times \vec{e}_{Z0}}{\left\| \vec{e}_y \times \vec{e}_{Z0} \right\|}, \quad \vec{e}_{Z0} = (0,0,1),$$
$$\vec{e}_z = \vec{e}_x \times \vec{e}_y.$$

Here × denotes the cross product of vectors.

20.9.4.3. Computation of terrain curve by the triangular mesh

At first, consider conditions for intersection of the wheel plane with a line segment AB. Let $\overline{R}_w$ be the radius vector to the wheel center in SC0 specified by the **GetWheelPosition** method; $\overline{R}_A, \overline{R}_B$ are the radius-vectors to the segment ends relative to SC0. Compute y coordinates of points A,B in SCWheel as

$$y_A = \vec{e}_y \bullet (\vec{R}_A - \vec{R}_w),$$
$$y_B = \vec{e}_y \bullet (\vec{R}_B - \vec{R}_w),$$

Here $\bullet$ denotes the scalar product of vectors.

Five variants take place:

1. $y_A y_B > 0$ – the plane does not intersect the segment;
2. $y_A y_B < 0$ – the plane intersect the segment in point C, which coordinates in SCWheel are

$$x_C = x_A + (x_B - x_A)\frac{y_A}{y_A - y_B}$$
$$z_C = z_A + (z_B - z_A)\frac{y_A}{y_A - y_B}$$

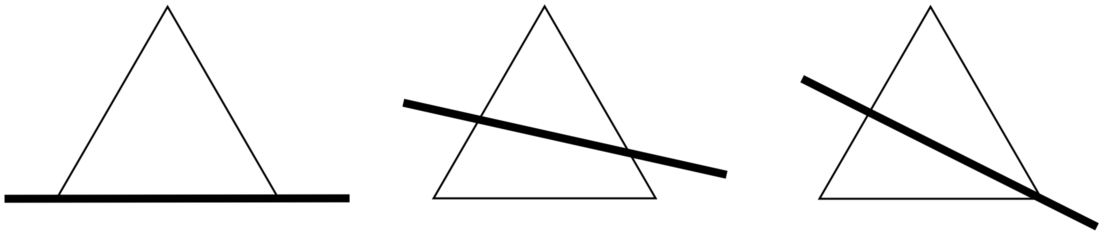
where

$$x_A = \vec{e}_x \bullet (\vec{R}_A - \vec{R}_w),\ z_A = \vec{e}_z \bullet (\vec{R}_A - \vec{R}_w),$$
$$x_B = \vec{e}_x \bullet (\vec{R}_B - \vec{R}_w),\ z_B = \vec{e}_z \bullet (\vec{R}_B - \vec{R}_w).$$

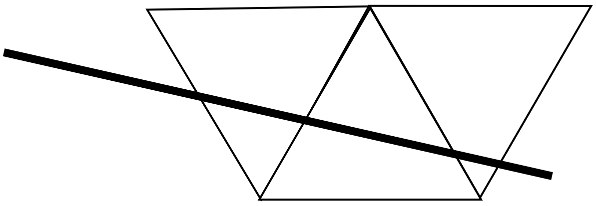
3. $y_A = 0, y_B \neq 0$ – the plane intersect the segment in point A;
4. $y_B = 0, y_A \neq 0$ – the plane intersect the segment in point B;
5. $y_A = 0, y_B = 0$ – the segment lies in the plane.

Now the condition of intersection of the wheel plane with a triangle ABC can be formulated:

- the plane intersects the triangle if the conditions 2) or 5) are fulfilled at least for one of the segments (AB, BC, CA)
- if the case of condition 5, the corresponding segment is added to the terrain polyline (Fig. a);
- if condition 2 is valid for two segments, the intersection points between the segments and plane specify the segment, which is added to the polyline (Fig. b);
- if condition 2 and 3) or 4) take place, the segment is added, which connects the intersection points according to 2) with the opposite vertex of the triangle (Fig. c).



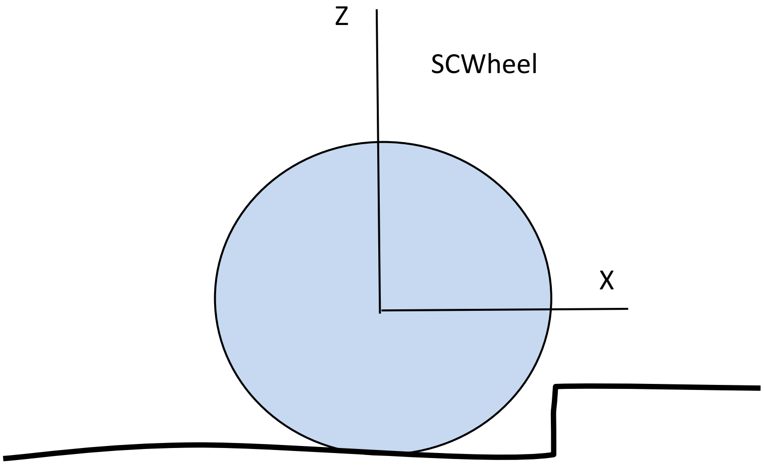
If the mesh topology is available (triangles for edges and vertices), this algorithm is applied to the first found segment of the polyline, and other triangles can be found as neighbor ones. For example, in case on Fig. b, two triangles having common edges with the first one should be considered.



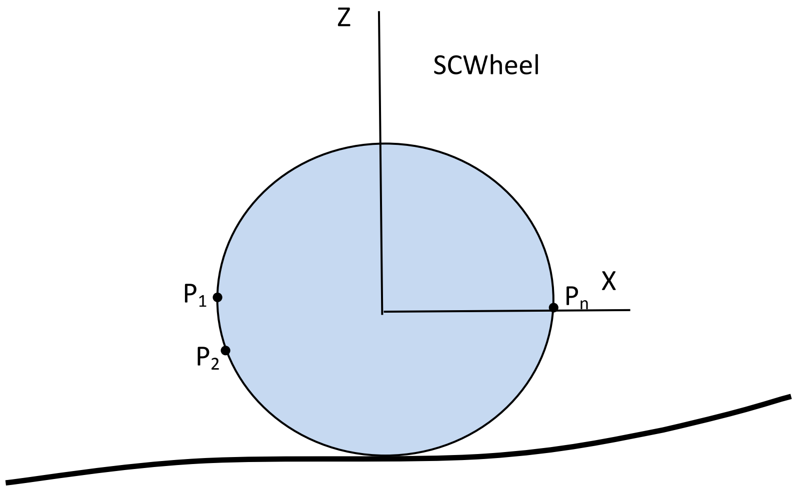
In case of Fig. a, c, triangles for the vertex intersected by the curve are considered.

20.9.4.4. Simplified terrain curve

If the mesh topology is not available, the algorithm described in the previous section could be time consuming. The simplified method for computation of the terrain curve can be used if a fast computation of Z (vertical) coordinate of the terrain surface according to X,Y coordinates is available. The method is good in case of smooth terrain surface and bad in non-smooth cases like in figure below.



Let a sequence of points $P_1, P_2 \dots, P_n$ lie on the wheel undeformed circle.



Coordinates of points are constant in SCWheel,

$$P_i = (x_i, 0, z_i), i = 1 \dots n$$

and variable in SC0 (R_i is the radius vector of point P_i in SC0):

$$P_i = (X_i, Y_i, Z_i) \infty \vec{R}_i, i = 1 \dots n,$$
$$\vec{R}_i = \vec{R}_w + x_i \vec{e}_x + z_i \vec{e}_Z$$

The definitions of the vectors in this expression are given in the previous section.

Let Z_{Ti} be the Z coordinate (vertical) of terrain directly under the point P_i in SC0, i.e. it corresponds to X_i, Y_i coordinates in plane OXY of SC0. The following point in SCWheel must be added to the terrain curve by the method **SetTerrainCurvePoint**:

$$x_{Ti} = x_i, z_{Ti} = z_i + Z_{Ti} - Z_i, \qquad i = 1 \dots n$$

and the normal is computed for the corresponding point on the terrain.

20.9.5. Tire blowout

Tire blowout is modeled by [1]

- instantaneous decrease of the tire radius to the rim radius;
- increase tire stiffness constant 4 times;
- increase tire rolling resistance to the drag value 0.3-0.4;
- no side force.

See:

20.9.6. Road coefficients of friction

Average values of friction coefficient [2].

Surface	**Peak value**		**Sliding value**	
	Dry	Wet	Dry	Wet
Asphalt	0.8-0.9	0.5-0.7	0.75	0.45-0.6

Concrete	0.8-0.9	0.8	0.75	0.7
Earth road	0.68	0.55	0.65	0.4-0.5
Gravel	0.6		0.55	
Snow (packed)	0.2		0.15	
Ice	0.1		0.07	

20.9.7. Rolling resistance of tires

The rolling resistance is considered as a torque $T_{rf} = F_{rf}R$ applied to the wheel directed opposite to the wheel roll, R is the rolling radius of the tire. According to the Wong [2], the resistance force is

$$F_{rf} = fN$$

where f is the coefficient of friction, and N is the tire normal force. The coefficient of friction depends on the vehicle speed as [2]

$$f = f_0 + k_1 v + k_2 v^2$$

Here v is the speed in km/h, and f_0, k_1, k_2 are empirical constants, which values are set by the SetRollingFriction method. Typical values of the coefficients can be found in [2], see the table

Parameters of rolling friction

Tire	f_0	k_1	k_2
radial-ply passenger car tire	0.0136	0	0.4e-7
bias-ply passenger car tire	0.0169	0	0.19e-6
radial-ply truck tire	0.006	0	0.23e-6
bias-ply truck tire	0.007	0	0.45e-6

20.9.8. Terrain roughness

20.9.8.1. Format of roughness file *.irr

Text files *.irr contain discrete roughness data for the left and right tracks. A file includes two columns. The first column contains the distance in meters, and the second one corresponds to the roughness height in meters. The recommended distance between points is 0.1m.

Example:

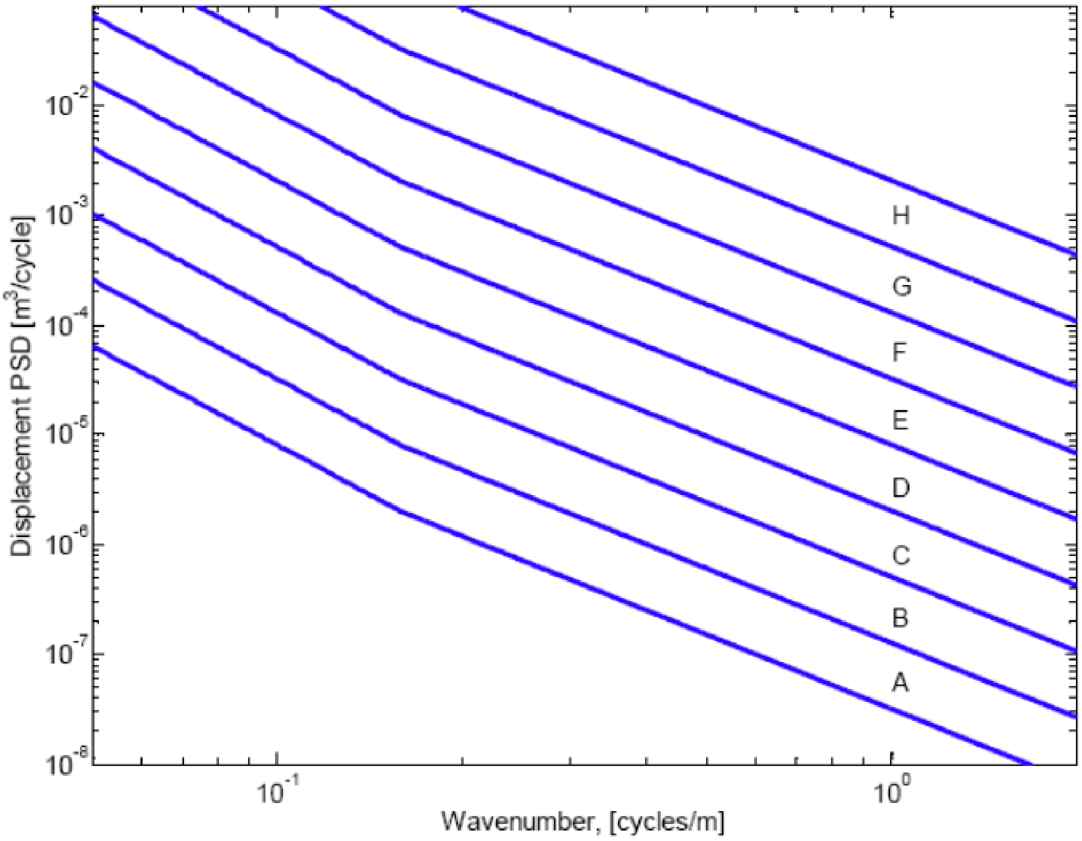
```
0  -0.0036114
0.1  -0.00382723
0.2  -0.00394107
0.3  -0.00395296
0.4  -0.00386557
0.5  -0.00368393
0.6  -0.00341505
```

```
0.7  -0.00306755
0.8  -0.00265132
0.9  -0.00217713
  1  -0.00165643
     -0.00110107
```

The user can either use the standard UM files or generate own files. The procedure LoadRoadRoughness is used for assignment of files with roughness functions to the left and right tracks. The UseRoadRoughness enables and disables the usage of the road roughness.

20.9.8.2. ISO 8608

The ISO 8608 1995 (e) classification (A-H) is used for generation of terrain roughness of different level [2]. The ISO standard specifies a power spectral density function (PSD):



The PSD function is [2]

$$S(n) = \begin{cases} S_0(n/n_0)^{N_1}, n < n_0 \\ S_0(n/n_0)^{N_2}, n > n_0 \end{cases}$$

where n is the spatial frequency. In the ISO 8608 the following values are recommended:

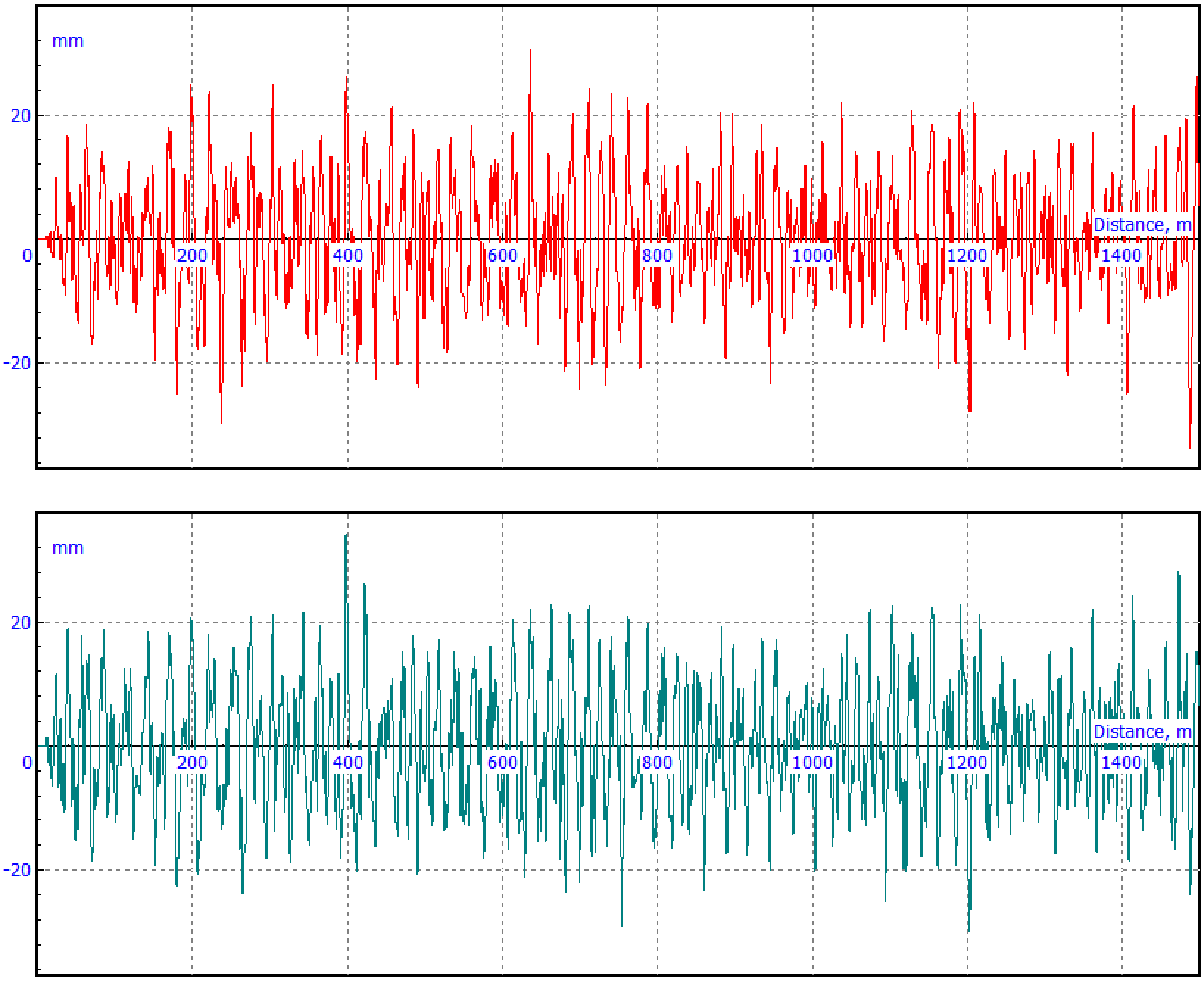
$$n_0 = {}^1\!/_2\,\pi, N_1 = 2, N_2 = 1{,}5$$

The parameter S_0 is the degree of roughness according to the table

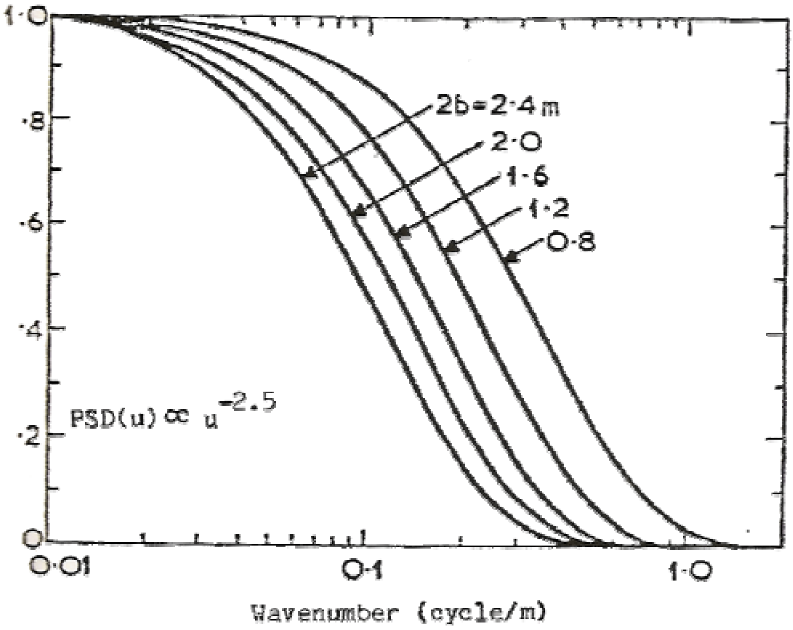
Road class	Degree of Roughness, S_0 ($\times 10^{-6} m^3$/cycles)	Factor to UM standard roughness $\sqrt{S_0/S^*}$
A(Very Good)	<8	0-0.63
B(Good)	8-32	0.63-1.26
C(Average)	32-128	1.26-2.53
D(Poor)	128-512	2.53-5.06
E(Very Poor)	512-2048	5.06-10.12
F	2048-8192	10.12-20.24
G	8192-32768	20.24-40.48
H	>32768	>40.48

20.9.8.3. UM standard roughness

UM uses the value $S_0 = S^* = 20 \times 10^{-6}$ (average value for the B class) as the standard one. The height/distance functions of the standard irregularities are generated for the left and right track and stored in the files iso_b_left_1500.irr and iso_b_right_1500.irr. The corresponding irregularities are shown in the figure below. By default, the roughness files must be located in the directory of the UM vehicle model.



The coherence function from [3] is used for generation of two-track irregularities.



Coherence function for different values of track width

20.9.8.4. Change of roughness

The **SetTerrainRoughnessFactor** method is used for change of the road class. The Factor= $\sqrt{S_0/S^*}$ parameter specifies the level of roughness. The value Factor=1 corresponds to the UM standard irregularities. Use the factor value from the above table to set the desired roughness.

The TransitionLength sets the distance for the uniform transitions to the new irregularity level. It is zero if the roughness is set before the simulation start. Otherwise if must be positive.

20.9.9. Computation of vehicle equilibrium

The method **DoEquilibriumPosition** is used for computation of equilibrium position of vehicle taking into account the terrain geometry. This position is used as initial one in simulation of vehicle motion.

Computation of equilibrium is executed as simulation of vehicle dynamics at which the car horizontal motion is locked. The simulation stops when the kinetic energy is less that the value of the StopEnegry parameter specified by the user. Simulation time is limited to the TMax parameter.

The following steps are required.

1. Specification of terrain geometry for each of the wheels by the **GetWheelPosition** and **SetRoadGeometry** methods. The components of normal in the SetTerrainGeometry method are ignored.
2. Call of the **DoEquilibriumPosition** method.

If succeed, the DoEquilibriumPosition method returns S_OK otherwise S_FALSE

Recommended values are as follows:

TMax = 30;

StopEnergy = 0.001.

20.9.10. Change of inertia parameters. Car occupants

Here we discuss how number of occupants and their inertia parameters as well as load of a truck can be changed. The car model must include bodies corresponding to occupants rigidly connected to the car body. If masses and moments of inertia are zeros, it is equivalent to absence of the occupant. Inertia parameters of occupants can be easily changed like it is described below.

Inertia parameters of bodies (mass and moments of inertia) can be changed if they are parameterized by identifiers. The following steps are recommended.

1. Get interface to the necessary identifier by the GetElementByNameEx method of the IUMObject interface, Sect. 20.2. *"IUMObject interface"*, p. 20-5.
2. Set the desired value to the identifier by the SetValue method of the IComInterface interface, Sect. 20.4. *"IComIdentifier Interface"*, p. 20-10.

If steps 1,2 are made before call of the method PrepareIntegration of IUMObject interface, no additional steps are required. If inertia parameters are changed during the simulation process, two additional steps must be done.

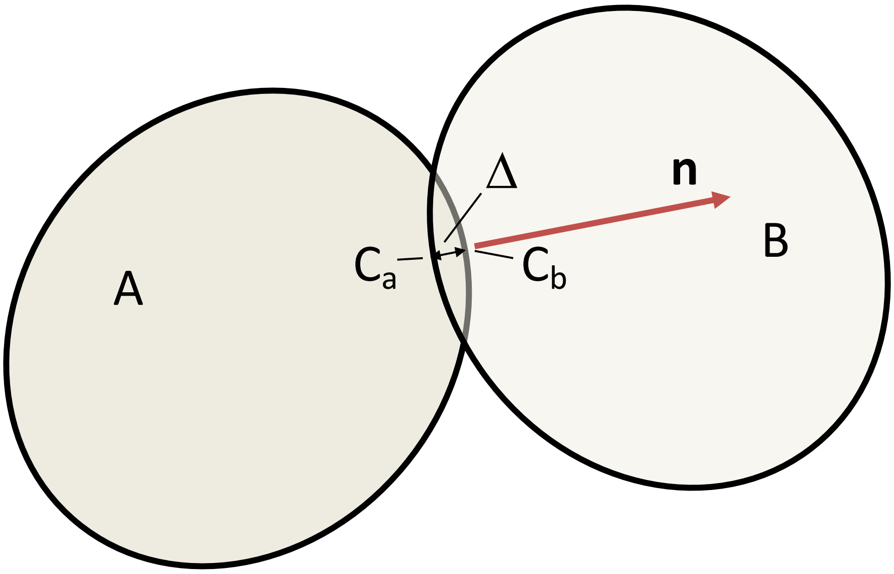
3. Get interface to the body which inertia parameters are changes by the GetElementByNameEx method of the IUMObject interface, Sect. 20.2. *"IUMObject interface"*, p. 20-5.
4. Call the RefreshExpressions method of the IBody interface to accept new values of the parameters.

```
Example:
var ptr: pointer;
      Indentifier : IComIdentifier;
Body : IBody;

      UMObject.GetElementByNameEx(eltIdentifier, 'car.moccupant1', ptr);
      Identifier:=IUnknown(ptr) as IComIdentifier;
 Identifier.SetValue(75)

      UMObject.GetElementByNameEx(eltBody, 'car.occupant1', ptr);
      Body:=IUnknown(ptr) as IBody;
Body.RefreshExpressions;
```

20.9.11. Collisions



Colliding bodies A (car body), B – second (external) body.

Penetration depth Δ: the maximal penetration of shapes. Contact points: C_a, C_b correspond to the maximal penetration.

Data necessary for computation of contact forces:

- Δ,
- coordinates of point Ca,
- velocity of point Cb of body B
- normal **n** to one of the colliding shapes external with respect to the car body.

20.9.12. Setting and evaluation of tire stiffness characteristics

20.9.12.1. Tire stiffness parameters

The following tire stiffness and damping characteristics are necessary in simulation of road vehicle dynamics:

C_s – longitudinal stiffness (N);

C_a – cornering stiffness (N/rad);

C_z – vertical static stiffness (N/m);

C_x – longitudinal static stiffness (N/m);

C_y – lateral static stiffness (N/m);

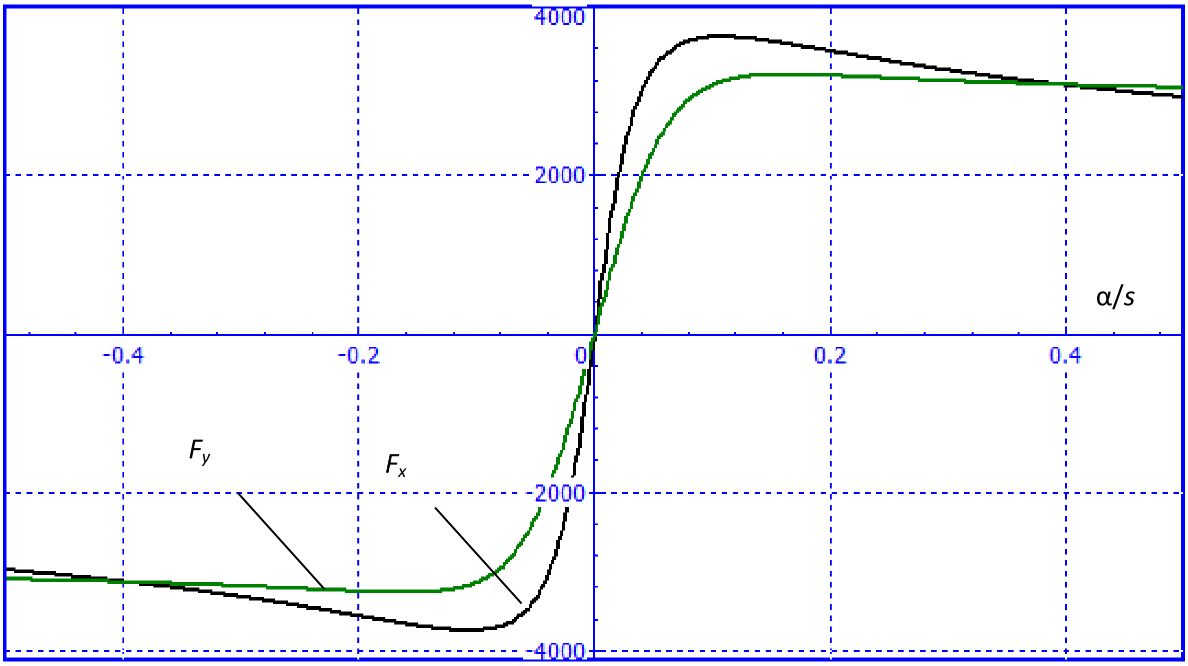
d_z – vertical damping constant (Ns/m).

Vertical, lateral and longitudinal stiffness parameters correspond to spring constants of unrolling tire in the corresponding direction. The vertical stiffness and damping constants C_z, d_z are important parameters. They are used for computation of vertical force F_z acting on the tire from the road both is standstill and motion

$$F_z = C_z\Delta_z + d_z\Delta_z,$$

where Δ_z is the tire vertical deflection. It is important to know the value stiffness C_z more or less exactly, Sect. 20.9.12.5. *"Approximate vertical stiffness and damping"*, p. 20-103.

In comparison with the vertical stiffness C_z, the longitudinal C_x and lateral C_y tire spring constants are of minor importance, they are used at standstill of a road vehicle only.



Typical dependences of tire cornering (F_y) and tractive/braking (F_x) forces on slip angle (α) and longitudinal slip (s)

The cornering stiffness C_s and longitudinal stiffness C_s are used in computation of creep forces (cornering and tractive/braking forces in figure above),

$$C_a = \left.\frac{\partial F_y}{\partial \alpha}\right|_{\alpha = 0}, \quad C_s = \left.\frac{\partial F_x}{\partial s}\right|_{s = 0},$$

So that for small slips

$$F_y \approx C_a\alpha, F_x \approx C_s s.$$

In the UM car simulator, the Fiala model [4] is implemented for computation of cornering and tractive/braking forces. The Fiala tire model requires cornering stiffness C_s and longitudinal stiffness C_s.

Tire rated stiffness parameters can be assigned either directly i.g. from experiments of computed according to approximate analytic expressions. In any case, the *EvaluationTireRatedStiff-*

ness method of the ICOMCar interface must be called right before the start of simulation and after setting the necessary tire parameters.

20.9.12.2. Rated stiffness parameters. Influence of inflation pressure

Values of tire stiffness parameters depend on the tire inflation pressure. Let $C_{s0}, C_{a0}, C_{z0}, d_{z0}$ be the values of the stiffness and damping parameters for rated inflation pressure p_0 and rated load W_0. The following simplified dependences on the inflation pressure p are accepted:

$$C_s = \frac{p}{p_0} C_{s0}, \qquad C_a = \frac{p}{p_0} C_{a0}, \qquad C_z = \frac{p}{p_0} C_{z0}, \qquad d_z = \sqrt{\frac{p}{p_0}} d_{z0}.$$

The actual pressure can be assigned to each of the tires both before the simulation start and during the simulation, the method *SetTireCurrentPressure*.

20.9.12.3. Direct assignment of rated stiffness parameters

By default, the rated parameters $C_{s0}, C_{a0}, C_{z0}, d_{z0}$ are assigned directly by the methods (Sect. 20.9.1. *"IComCar interface"*, p. 20-81).

SetTireLongitudinalStiffness
SetTireCorneringStiffness
SetTireVericalStiffness

20.9.12.4. Approximate evaluation of tire rated stiffness parameters

To specify the approximate assessment of tire rated stiffness parameters, the SetAutoEvaluationTireStiffness method of ICOMCar interface is used (Sect. 20.9.1. *"IComCar interface"*, p. 20-81). The argument values denotes

Vertical = 1 : evaluation of C_{z0}, d_{z0}
Cornering = 1 : evaluation of C_{a0}
Longitudinal = 1 : evaluation of C_{s0}
Example: SetAutoEvaluationTireStiffness(0, 1, 1) : evaluation of C_{a0}, C_{s0}.

The following additional tire parameters are necessary for approximate computation of the tire stiffness characteristics:

R – tire unloaded radius (the method SetTireUnloadedRadius);
w – tire section width (the method SetTireSectionWidth);
W0 – tire rated load (the method SetTireRatedLoad);

$\beta_z \in [0.3, 0.75]$ – vertical damping ratio (the method SetTireDampingRatio) for evaluation of the vertical damping constant d_{z0} only;

$\lambda_y \in [0.8, 0.18]$ – cornering coefficient (the method SetCorneringCoefficient) for evaluation of the cornering stiffness C_{a0} only, Sect. 20.9.12.6. *"Approximate cornering stiffness"*, p. 20-103. Typical value of cornering coefficient is 0.12 for bias-ply tires and 0.16 for radial ply tires [5].

20.9.12.5. Approximate vertical stiffness and damping

According to [6], the vertical tire spring constant with less than 20% error tolerance can be computed as

$$C_{z0} = \pi p_0 \sqrt{2Rw}.$$

The damping constant is evaluated according to the damping ratio parameter

$$d_{z0} = 2\beta_z \sqrt{C_{z0} m_w}$$

Here m_w is mass of wheel.

20.9.12.6. Approximate cornering stiffness

According to [2], the cornering coefficient $\lambda_y \in [0.8, 0.18]$ is equal to the ratio of the lateral force at 1 degree of sleep angle to the tire load, so the cornering stiffness is

$$C_{a0} = \lambda_y W_0 \frac{\pi}{180}.$$

Table of cornering coefficient values for different tires is presented in [2].

20.9.12.7. Approximate longitudinal stiffness

Expression for evaluation of the longitudinal tire stiffness C_{s0} is proposed in the report [7]

$$C_{s0} = \frac{2}{L_t^2} \boldsymbol{\kappa} W_0$$

where $\boldsymbol{\kappa} \approx 18$, L_t – length of tire contact patch,

$$L_t \approx 2R \sqrt{\frac{2W_0}{C_{z0} R}}$$

20.9.12.8. Longitudinal and lateral static stiffness

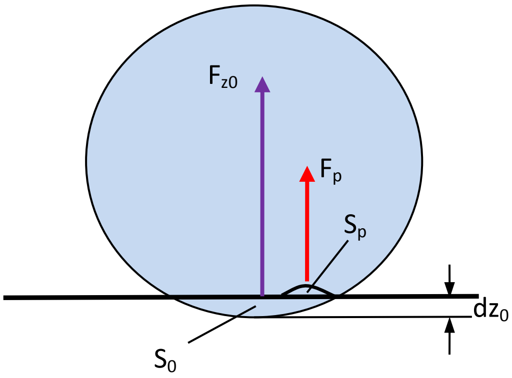
Lateral static stiffness is computed in term of the cornering stiffness [2] as

$$C_y = \frac{C_a}{0.8R + L_t/2}.$$

We take the value of longitudinal static stiffness equal to the lateral static stiffness

$$C_x = C_y$$

20.9.13. Run over the pebble



Model of a pebble under the tire

Consider a simplified model of a tire runs over a pebble. The model includes an additional vertical force F_p, which is proportional to some effective pebble area S_p. Let dz_0 be the static deflection of the tire, which we interpret here as a penetration of the tire circle into the road surface, and S_0 be the area of the penetration. These parameters are dependent as

$$dz_0 = \frac{S_0^2}{2R^3}$$

According to the linear model of the vertical tire force, the static tire force is proportional to the deflection dz_0

$$F_{z0} = C_z dz_0 = C_z \frac{S_0^2}{2R^3}$$

Consider a pebble under the tire. If we assume that the penetration area in this case is increased to the value $S = S_0 + S_p$, and accept the square dependence of the tire force on the penetration area, we obtain

$$F_z = F_{z0} + F_p = C_z \frac{\left(S_0 + S_p\right)^2}{2R^3} = C_z \frac{S_0^2\left(1 + S_p/S_0\right)^2}{2R^3} \approx F_{z0} + F_{z0}\frac{2S_p}{S_0},$$

and finally

$$F_p = 2F_{z0}k_p, \;\; k_p = \frac{S_p}{S_0}.$$

Here we have introduced the pebble size factor k_p, which specifies the value of the additional vertical force F_p.

Use the SetPebbleUnderTire method to set a pebble under any of the tires during the simulation process.

Remark. The feature is used for simulation of small pebbles, so that kp<0.5.

20.10. Interface for internal combustion engine (ICE)

See the user's manual, Chapter 22, file for detailed description of the ICE models in UM.

Interface: *IComICEngine*

Hierarchy: *IUnknown – IComICEngine*

20.10.1. Methods of IComICEngine interface

The interface is available by the GetICEngine method of the IComCar interface, Sect. 20.9.1. *"IComCar interface"*, p. 20-81.

Methods	**Description**
SetICEType	HRESULT _stdcall SetICEType([in] int Value); Sets the engine type Input: value: 0 – none, 1 – spark ignition, 2 – diesel.
GetICEType	int _stdcall GetICEType(void); Returns the engine type. Result: 1 : spark ignition engine; 2 – diesel engine
SetNCylinders	HRESULT _stdcall SetNCylinders([in] int Value); Sets number of cylinders
GetNCylinders	int _stdcall GetNCylinders(void); Returns number of cylinders as a result
SetNStrokes	HRESULT _stdcall SetNStrokes([in] int Value); Sets number of strokes
GetNStrokes	int _stdcall GetNStrokes(void); Returns number of strokes as a result
SetPistonStroke	HRESULT _stdcall SetPistonStroke([in] double Value); Sets length of piston stroke, mm
GetPistonStroke	HRESULT _stdcall GetPistonStroke([out] double* Value); Returns length of piston stroke, mm
SetCapacity	HRESULT _stdcall SetCapacity([in] double Value); Sets engine capacity, L
GetCapacity	HRESULT _stdcall GetCapacity([out] double* Value); Returns engine capacity, L
SetEniginePower	HRESULT _stdcall SetEniginePower([in] double Value); Sets the engine power, kW
GetEnginePower	HRESULT _stdcall GetEnginePower([out] double* Value); Returns the engine power, kW
SetMaxTorque	HRESULT _stdcall SetMaxTorque([in] double Value, [in] double Speed);

	Sets the engine maximal torque value and the corresponding speed. Input: Value – maximal torque, Nm Speed : engine speed for maximal torque, rpm
GetMaxTorque	HRESULT _stdcall GetMaxTorque([out] double* Value, [out] double* Speed); Returns the current values of engine maximal torque value and the corresponding speed. Output: Value – maximal torque, Nm Speed : engine speed for maximal torque, rpm
SetMinMaxSpeed	HRESULT _stdcall SetMinMaxSpeed([in] double MinSpeed, [in] double MaxSpeed); Sets values of the minimal and maximal (speed for nominal power) engine speed, rpm
GetMinMaxSpeed	HRESULT _stdcall GetMinMaxSpeed([out] double* MinSpeed, [out] double* MaxSpeed); Returns values of the minimal and maximal (speed for nominal power) engine speed, rpm
SetGovernorType	HRESULT _stdcall SetGovernorType([in] int Value); Sets the engine governor type depending on the engine type. Input: Spark ignition engine Value : 0 (None), 1 (One speed) Diesel engine Value: 2 (MinMax or two speed), 3 (all speed). See Chapter 22, Sect. Engine governors
GetGovernorType	int _stdcall GetGovernorType(void); Returns the engine governor type depending on the engine type. Result : 0 (None), 1 (One speed), 2 (MinMax or two speed), 3 (all speed). See Chapter 22, Sect. Engine governors
SetEngineStart	HRESULT _stdcall SetEngineStart(void); Starts the engine
SetEngineStop	HRESULT _stdcall SetEngineStop(void); Engine stalls
GetEngineState	int _stdcall GetEngineState(void); Returns the engine state. Result : 0 (off), 1 (start mode, the engine speed increases to the idle value), 2 (operation mode).
GetEngineSpeed	HRESULT _stdcall GetEngineSpeed([out] double* RPM);

	Returns the engine speed, rpm
SetAnalyticICEModelType	HRESULT _stdcall SetAnalyticICEModelType(void); Sets analytic model type for the torque map model. The method must be used for ICE models constructed with UM COM (not with UM full)/ See Chapter 22, Sect. Engine torque map
SetMapFormFactor	HRESULT _stdcall SetMapFormFactor([in] double Value); Sets the engine torque map form factor s. See Chapter 22, Sect. Analytic engine map for spark ignition engine; Analytic engine map for diesel engine
GetMapFormFactor	HRESULT _stdcall GetMapFormFactor([out] double* Value); Returns the current value of the engine torque map form factor s. See Chapter 22, Sect. Analytic engine map for spark ignition engine; Analytic engine map for diesel engine
SetMapSpecialSpeed	HRESULT _stdcall SetMapSpecialSpeed([in] double Value); Sets the special speed for engine torque n*, rpm (spark ignition engine only). See Chapter 22, Sect. Analytic engine map for spark ignition engine
GetMapSpecialSpeed	HRESULT _stdcall GetMapSpecialSpeed([out] double* Value); Returns the special speed for engine torque n*, rpm (spark ignition engine only). See Chapter 22, Sect. Analytic engine map for spark ignition engine
SetFullLoadCurveCoefs	HRESULT _stdcall SetFullLoadCurveCoefs([in] double a, [in] double b, [in] double c); Sets Lederman parameters of the full load torque-speed curve for both spark ignition and diesel engine. Input: a, b, c – values of Lederman parameters See Chapter 22, Sect. Full load torque-speed curve
GetFullLoadCurveCoefs	HRESULT _stdcall GetFullLoadCurveCoefs([out] double* a, [out] double* b, [out] double* c); Returns Lederman parameters of the full load torque-speed curve for both spark ignition and diesel engine. Output: a, b, c – values of Lederman parameters See Chapter 22, Sect. Full load torque-speed curve
ComputeDieselFullLoadCurveCoefs	HRESULT _stdcall ComputeDieselFullLoadCurveCoefs([in] int ModelType, [in] double

	KM, [in] double Kn); Computes the Leiderman a, b, c parameters for the full load torque-speed curve in case of diesel engine. Input: Model type: 1 (Variant 1 of the model); 2 (Variant 2 of the model) See Chapter 22, Sect. Full load torque-speed curve
SetTorqueLost	HRESULT _stdcall SetTorqueLost([in] double A, [in] double B); Sets torque lost parameters $M_{\mathrm{fa}}, M_{\mathrm{fb}}$, see Chapter 22, Sect. Torque lost
GetTorqueLost	HRESULT _stdcall GetTorqueLost([out] double* A, [out] double* B); Returns torque lost parameters $M_{\mathrm{fa}}, M_{\mathrm{fb}}$, see Chapter 22, Sect. Torque lost
GetFMEP	HRESULT _stdcall GetFMEP([out] double* p0, [out] double* p1); Returns current values of fmep parameters p1, p2, see Chapter 22, Sect. Torque lost
SetFMEP	HRESULT _stdcall SetFMEP([in] double p0, [in] double p1); Sets fmep parameters p1, p2, see Chapter 22, Sect. Torque lost
ComputeTorqueLostParams	HRESULT _stdcall ComputeTorqueLostParams(void); Computes torque lost parameters $M_{\mathrm{fa}}, M_{\mathrm{fb}}$ for the current values of fmep parameters p1, p2, see Chapter 22, Sect. Torque lost
SetMinMaxSpeedDroop	HRESULT _stdcall SetMinMaxSpeedDroop([in] double MinSpeedDroop, [in] double MaxSpeedDroop); Sets the governor parameters. Input: MinSpeedDroop δ_{min} (for two- and all-speed governors), %, MaxSpeedDroop δ_{max} (for any governor), %. See Chapter 22, Sect. Engine governors
GetMinMaxSpeedDroop	HRESULT _stdcall GetMinMaxSpeedDroop([out] double* MinSpeedDroop, [out] double* MaxSpeedDroop); Returns the governor parameters. Output: MinSpeedDroop δ_{min} (for two- and all-speed governors), %, MaxSpeedDroop δ_{max} (for any governor), %. See Chapter 22, Sect. Engine governors
ReadFromFile	HRESULT _stdcall ReadFromFile([in] LPSTR FileName); Reads engine model parameters from a text file *.ice.

	Input: FileName – direct path to the file
SaveToFile	HRESULT _stdcall SaveToFile([in] LPSTR FileName); Saves engine model parameters to a text file *.ice. Input: FileName – direct path to the file

Most of the methods return HResult=S_OK in the case of a successful termination and S_FALSE in case of failure.

20.10.2. Development of ICE model with IComICEngine interface

Here we consider a calling sequence of the *IComICEngine* method for full description of a ICE model with UM COM server.

1. Load a car model by the **LoadObjectFromFile** method of the *IUMObject* interface.
2. Get the *IComCar* interface then the **GetCar** method of the *IUMObject* interface
3. Get the *IComICEngine* interface by the **GetICEngine** method of the *IComCar* interface
4. Set general information about the engine by the methods

   SetICEType
   SetNCylinders
   SetNStrokes
   SetPistonStroke
   SetCapacity
   SetEniginePower
   SetMaxTorque
   SetMinMaxSpeed
   SetAnalyticICEModelType
5. Specify analytic full load torque curve by the method

   SetFullLoadCurveCoefs

   In the case of a diesel engine, the method **ComputeDieselFullLoadCurveCoefs** can be used instead of the **SetFullLoadCurveCoefs**
6. Specify the friction torque lost.

   Use the parameters M_{fa}, M_{fb} are available, use the **SetTorqueLost** method, otherwise use the methods

   SetFMEP
   ComputeTorqueLostParams
7. Set the torque map parameters by the methods

   SetMapFormFactor
   SetMapSpecialSpeed (for spark ignition engine only)
8. Specify the governor type and parameters by the methods

   SetGovernorType
   SetMinMaxSpeedDroop (if a governor is presented)
9. If necessary, save the model to a file by the method **SaveToFile**.

References

[1] Bareket Z., Blower D. F., MacAdam C., Blowout Resistant Tire Study for Commercial Highway Vehicles, Final Technical Report for Task Order No. 4 (DTRS57-97-C-00051), UMTRI, 2000.

[2] Wong J.Y. Theory of Ground Vehicles. 4th Edition. Wiley. 2008.

[3] Robson J.D., (1979) Road Surface Description and Vehicle Response, International Journal of Vehicle Design,. 1(1), 25–35.

[4] UM User’s manual. Simulation of Road Vehicle Dynamics, file 12_UM_Automotive.pdf.

[5] John C. Dixon. Tires, Suspension and Handling. Cambridge University Press, 1996. Second Edition.

[6] Nybakken G.H., Clark S.K., Vertical and lateral stiffness characteristics of aircraft tires. NASA contractor report NAS CR-1488, University of Michigan, 1969.

[7] Szostak H.T., Allen W.R., Rosenthal T.J., Analytical Modeling of Driver Response in Crash Avoidance Maneuvering Volume II: An Interactive Model for Driver/Vehicle Simulation, U.S Department of Transportation Report NHTSA DOT HS-807-271, April 1988.