DRIFT COM Interface Guide

ESR/D1000846/SUG02/Issue 5

12h August 2011

ESR Technology
Engineering, Safety & Risk

Authorisation Sheet

Report Title:	DRIFT COM Interface Guide
Customer Reference:	
Project Reference:	D1000846
Report Number:	SUG03
Issue:	Issue 5
Distribution List:	

Author:	E Walton	12/08/2011
Reviewed:	J E Carlisle	12/08/2011
Authorised:	G A Tickle	12/08/2011

Contents

1.0 Introduction

DRIFT 3.1 can be used via Excel through the use of the driftWrapper.dll and its associated driftWrapper.tlb. Although this should automatically be set up on installation, it can be configured by going to the Visual Basic Editor (VBE) and going to Tools / References and clicking on browse to find the driftWrapper.tlb. DRIFT can be used through Visual Basic for Applications (VBA) to generate DRIFT input files, save, run and load .drift files as well as extract and analyse data from a completed DRIFT scenario.

Depending on your IT set-up you may need to copy the file excel.exe.config (which is packaged with the DRIFT install) to the same location as your Microsoft Excel executable (Excel.exe). If driftWrapper.tlb is not recognised in the Excel VBA editor then it is likely that you will need to perform this step before you can use the COM wrapper. When you next open Excel it should pick up the excel.exe.config file automatically and allow you to use the functionality contained in the driftWrapper.dll.

The DRIFT library is hierarchical, beginning with the DRIFT Model class which contains base level methods, such as save and run, as well as the DRIFT Data class to set model properties. The DRIFT Data class extends down through many classes, in which the properties for a full DRIFT run can be set. Figure 1 shows an overview of the class hierarchy used in DRIFT. A full list of members of each class can be found by pressing F2 in the VBE window and searching for the class name. The structure of DRIFT as viewed from VBA mirrors that of the Drift.exe counterpart and so more useful information may be found from the DRIFT User Guide.

VBA can be used to input all data into DRIFT to create a new model. However, this is a little complicated as it is relatively unassisted compared to entering data through the Drift.exe directly. Section 2.0 details all properties within the DRIFT Data class and can be used as a guide to create a complete DRIFT Data set, ready to be set as input data to the DRIFT Model. Section 3.0 details the methods and properties of a DRIFT Model including how to save, load and run DRIFT from VBA and then return results. An example of how to input and save a .drift file from VBA is provided in Section 5.0.

VBA can also be used to load and run existing .drift files which can then be used to retrieve results from a run .drift file. Section 6.0 discusses returning results through VBA to Microsoft Excel and provides an example of how to do so.

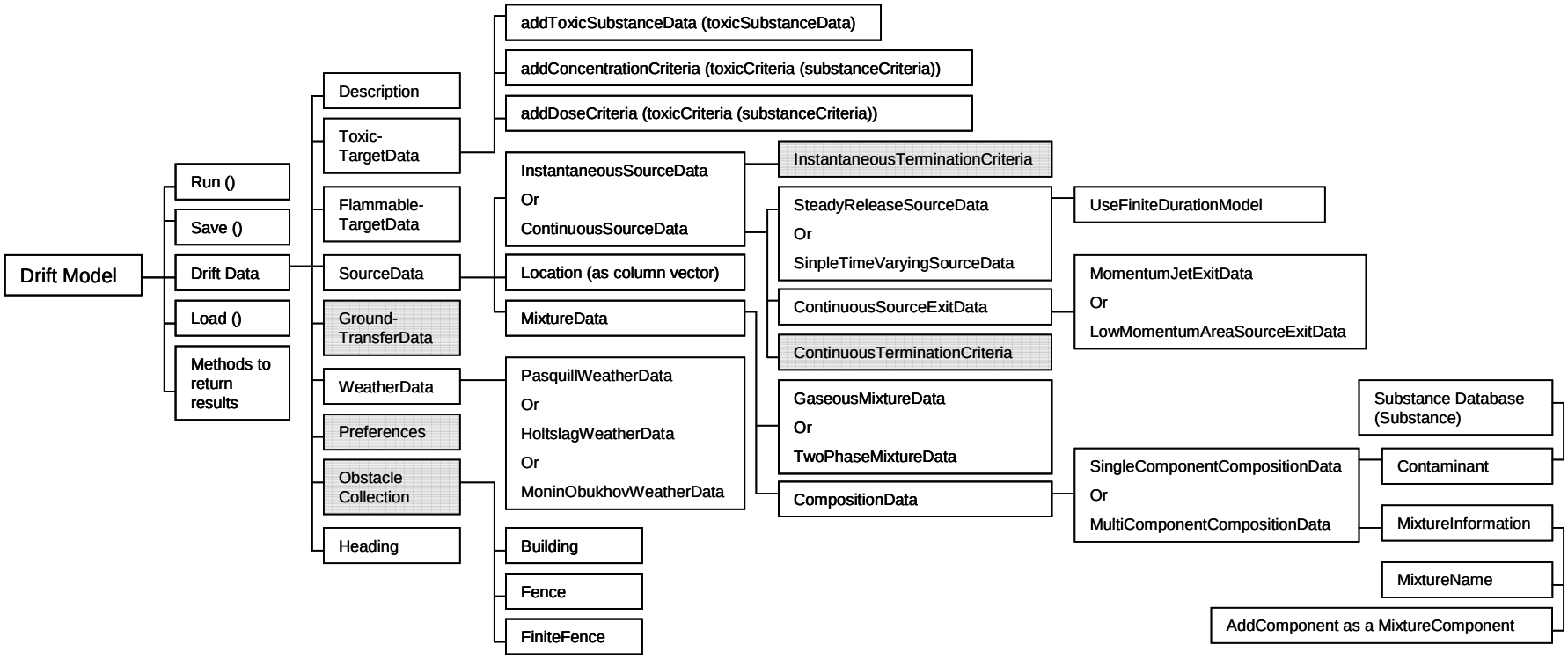

Figure 1: An overview of the class hierarchy used in DRIFT. All aspects of the DRIFT Data tree need to be set before any other method on DRIFT Model can be called. Classes in grey can be set to default values if no distinct changes to any of the settings are necessary. In such a case the user does not need to enter any data into that class.

2.0 Members of DRIFT Data

Below is a table of read/write properties belonging to the DRIFT Data class and any subsequent classes. These properties should be input before driftModel.run() or driftModel.save() is called. Once all data has been entered into the DRIFT data it can then be set to the inputData on the DRIFT model class ready to be run or saved.

"Property name" is the name of properties belonging to the data class of the current sub heading. The "Description" column provides a brief description of what the property is. "Type" provides information about what type of input the code is expecting. The "units" column gives the units of the input and the valid range column states the appropriate range any input value should lie within.

Property Name	Description	Type	Units	Valid Range
DRIFTData				
Heading	The name of the file.	String	None	Any string
Description	A description of the file. Can be multiple lines.	String	None	Any string
IsDataValid	Returns a Boolean stating whether is possible to save/run the DRIFT Data or not. If a string is passed in as an argument, a reason for any invalidity will be returned in that string.	Boolean	None	
SourceData				
MixtureData	Must be set to MixtureData. See MixtureData section.	MixtureData		
Localtion	The location of source in x,y,z which needs to be set as a ColumnVector (x is aligned with East, y along North and z upwards). A demonstration of this is given in section 5.0.	Double	metres	Any value
InstantaneousSourceData				
CrossWindRadius	Radius of cloud perpendicular to wind direction.	Double	metres	> 0
DownWindRadius	Radius of cloud parallel to wind direction.	Double	metres	> 0
IsUnitAspectRatio	If set true assumes height of cloud is equal to the radius of cloud.	Boolean	None	True/False
Mass	The mass of the released cloud.	Double	Kg	> 0
TerminationCriteria	Termination criteria to be set as InstantaneousTerminationCriteria class. If using default values can set "= New InstantaneousTerminationCriteria"	Instantaneous-TermiationCriteria		
ContinuousSourceData				
ExitData	Needs to be set as continuousSourceExitData, which can in turn be set to MomentumJetExitData or LowMomentumAreaSourceData.	ContinuousSource-ExitData		
TerminationCriteria	Needs to be set as ContinuousTerminationCriteria. If using default values set to "= New ContinuousTerminationCriteria"	ContinuousTerminationCriteria		
SteadyReleaseSourceData				
ReleaseRate	The rate of gas cloud released.	Double	kg/s	> 0
Duration	The duration of the release.	Double	Seconds	> 0

Property Name	Description	Type	Units	Valid Range
EndTime	READ ONLY. The time at which contaminant stops being released.	Double	Seconds	> 0
UseFiniteDurationModel	Set to true to enable a finite duration release.	Boolean	None	Yes/No
StartTime	The start time of the release.	Double	Seconds	≥ 0
SimpleTimeVaryingSourceData				
CloudSegment()	READ ONLY. Argument takes an index and returns the SteadyReleaseSourceData cloud segment associated with the index.	SteadyReleaseSourceData		
divideIntoDurationSegments()	Takes a tolerance and minimum segment duration as arguments. Divides a continuous exit profile into segments as specified by the arguments given.			
ReleaseRateProfile	Used to set the profile of release by time. Needs to be set to a UserDefinedReleaseRateProfile which enables release rates at up to 100 time intervals to be set.	UserDefinedReleaseRateProfile		
NumberOfSegments	READ ONLY. Returns the number of segments the release profile is calculated to.	Long		
UserDefinedReleaseRateProfile				
addDataPoint()	Sets the release profile by adding a point at a time. Takes an argument of time as a double and a second argument of the corresponding release rate as a double.	Double	Seconds kg/s	
MomentumJetExitData				
AngleFromHorizontal	The jet angle from horizontal.	Double	Degrees	-90 to 90
AngleFromNorth	Jet angle bearing from north.	Double	Degrees	0 to360
DischargeCoeffcient	Coeffcient relating release rate and exit velocity. Typically 0.61 for liquids and 0.8 for gasses.	Double	None	0 to 1
OrrificeRadius	The radius of circular orifice.	Double	Metres	> 0
LowMomentumAreaSourceExitData				
CrosswindSourceRadius	Radius of cloud perpendicular to wind direction.	Double	metres	
DownwindSourceRadius	Radius of cloud parallel to wind direction.	Double	metres	
IncludeDilutionOverTheSource	Set to true to allow solver to include dilution effects over the source location due to mixing with air.	Boolean	None	Yes/No
MixtureData				

Property Name	Description	Type	Units	Valid Range
CompositionData	Needs to be set to CompositionData class, which contains information on the contaminant(s).	CompositionData		
ContanintnatMassFraction	The fraction of cloud that is the specified contaminant, by mass, at the source.	Double	Kg/Kg	0 to 1 (pure contaminant)
Temperature	Temperature of the released gas cloud.	Double	Kelvin	
CompositionData				
HasLiquid	READ ONLY. Returns if the substance(s) have a liquid phase.	Boolean		
NumberOfLiquidPhases	READ ONLY. Returns the number of liquid phases of substance(s).	Long		
SingleComponent-CompositionData				
Contaminant	Can be set to a substance database substance or a user defined substance. For a database substance: Set SingleComponentCompositionData.contaminant = SubstanceDatabase.substance(SubstanceName as string)	Substance		
LiquidFraction	The fraction of substance which is in liquid state on exit. Only needs to be set if using a TwoPhaseCompositionData.	Double		0 (gaseous) to 1 (pure Liquid)
SubstanceDatabase				
DataSource	Use to specify which database a substance will be found from, either DataSourceType_SPI or DataSourceType_SRD.	DataSourceType		
setFilePath()	Takes a file location as an argument to set where the databases are stored. For the ESRsubstance database a full directory should be set along with the ending "\ESRsubstance.mdb". For SPI files the folder contining the SPI files should be given	String	None	
Substance	Sets a substance from a database to the composition data. Takes an argument of the substance name as a string.	String		
MultiComponentCompositionData				
MixtureInformation	Specifies the substances and their properties. Needs to be set as MixtureInformation class.	MixtureInformation		

Property Name	Description	Type	Units	Valid Range
IsFlammable	READ ONLY. Returns whether the mixture is flammable as a byte.	Boolean		
LowerFlammableLimit	READ ONLY. Returns the LFL.	Double		
saturatedVapourPressure()	Takes a temperature as an argument and sets the saturated vapour pressure of the mixture.	Double		
saturationTemperature()	Takes a pressure as an argument and sets the saturationtemperature of the mixture.	Double		
UpperFlammableLimit	READ ONLY. Returns the UFL.	Double		
MixtureInformation				
addComponent()	Adds a component to the mixture. Must be set to a MixtureComponent.	MixtureComponent		
DefineByMass	Set true to define the mixture by mass rather than moles.	Boolean	None	Yes/No
MixtureName	The name of the mixture	String	None	Any String
MixtureComponent				
Substance	The substance which forms this component of the overall mixture. Must be set as a substance which can be set to be a database substance: .substance = □atabase.Substance(Substance as string).	Substance		
AmountOfVapour	Set the fraction of this component which is vapour.	Double		
AmountOfLiquid	Set the fraction of this component which is liquid	Double		
LiquidPhaseIndex	Sets the liquid Phase index, which can take the following values: 0 – Miscible with Water, 1 – Liquid Phase 1 or 2 – Liquid Phase 2.	Long	None	0,1,2
GaseousMixtureData – ***Contains no further properties to MixtureData but needs to be set"= New GaseousMixtureData" so the model knows which class of mixture data to use.***				
TwoPhaseMixtureData				
Pressure	The Pressure of the substance on release	Double	Pa	
RainoutFraction	The fraction of substance which will undergo rainout as liquid drops on exit.	Double	Fraction	0 to 1
WeatherData				
PerformAutomaticMixingHeightCalculation	Determines whether the solver automatically calculates the mixing (or inversion) height or if the user inputs a value.	Boolean	None	True/False

Property Name	Description	Type	Units	Valid Range
RelativeHumidity	The relative humidity as a fraction of saturated conditions.	Double	None	0 to 1.07
RoughnessLength	The characteristic roughness length of the ground.	Double	Metres	
Temperature	The temperature at the reference height.	Double	Kelvin	
UserInputMixingHeight	Use to manually input the height of the atmospheric boundary layer.	Double	Metres	> 0
WindAngleFromNorth	The angle between north and the direction the wind is blowing from. 0 means the wind is blowing from the north.	Double	Degrees	0 to 360
WindReferenceHeight	The height at which stated wind speed is set.	Double	Metres	
PasquillWeatherData				
PasquillStability	Pasquill stability class. Needs to be set as a stability class. From A: very unstable to F: stable.	Stability		Stability_A to Stability_F
WindSpeed	Wind speed in m/s	Double	Metres	
HoltslagWeatherData				
CloudCover	Fraction of sky covered by cloud.	Double	None	0 to 1
Year/Month/Day/Hours/ Minutes/Seconds	Time setting	Double	App. Units	
Lattitude	Release Latitude.	Double	Degrees	-90 to 90
Longitude	Release Longitude.	Double	Degrees	-180 to 180
Windspeed	Wind speed in m/s.	Double	m/s	> 0
MoninObukhovWeatherScheme				
InverseObukhovLength	The inverse Obukhov length, scaling length for atmospheric boundary layer .	Double	/m	
Ustar	The friction velocity. Scaling velocity for lower parts of atmospheric boundary layer.	Double	m/s	> 0
ToxicTargetData				
MaximumExposureDuration	Maximum time exposed to Toxic cloud.	Double	Seconds	
NumberOfConcentration-Criteria	READ ONLY. Returns the number of concentration criteria on model	Integer	None	
NumberOfDoseCriteria	READ ONLY. Returns the number of dose criteria on model	Integer	None	
NumberOfToxicSubstances	READ ONLY. Returns the number of toxic substances set to model.	Long		

Property Name	Description	Type	Units	Valid Range
Substance_Data	READ ONLY. Takes an index as an argument and returns the substance data associated with the index.	ToxicSubstanceData		
ThresholdConcentration	The Threshold concentration	Double		
addConcentrationCriteria()	Adds a toxic concentration exponent for each toxic substance in model. Must be set as a ToxicCriteria which in terms takes a SubstanceCriteria class as an argument.	ToxicCriteria		
addDoseCriteria()	Adds a toxic dose exponent for each toxic substance in model. Must be set as a ToxicCriteria which in terms takes a SubstanceCriteria class as an argument.	ToxicCriteria		
addToxicSubstanceData()	Sets a new toxic subatance's data to model. Argument needs to be given as a toxicSubstanceData class.	ToxicSubstanceData		
UseTimeAveraging	Tells the model whether or not to account for the effects of time-averaging (lateral and vertical meander). Note that flammable results **never** account for time-averaging.	Bool		
AveragingTime	The averaging time over which toxic dose and concentrations will be calculated.	Double	Seconds	0 -
ToxicDoseFractionMethod	Tells the model which method to use for calculating toxic dose fraction (refer to user guide for explanation of these).	DoseFractionMethod		
ClearConcentrationCriteria	Clears all concentration criteria			
ClearToxicSubstanceData	Clears all toxic substance data.			
ConcentrationCriteria	READ ONLY. Takes an index as an argument and returns the concentration criteria associated with the index.	ToxicCriteria		
DoseCriteria	READ ONLY. Takes an index as an argument and returns the dose criteria associated with the index.	ToxicCriteria		
IndoorsLagTime	The toxic lag time for indoor locations.	Double	Seconds	0 -
IndoorsVentilationRate	The rate of ventilation for indoor locations.	Double	Kg/s	0 -
ToxicSubstanceData				
SubstanceName	The name of the toxic substance	String	None	

Property Name	Description	Type	Units	Valid Range
ToxicDoseExponent	Toxic exponent. The power to which concentration is raised when calculating toxic dose.	Double	None	
ToxicCriteria				
addSubstanceCriterion()	Add a substance criteria. Must be set to a SubstanceCriteria class, in which the toxic substance name and concentration or dose level of interest need to be set.	SubstanceCriteria		
clearCriteria()	Clears the toxic criteria.			
Location	The location where toxic effects will be evaluated. Must be set to a ReceiverLocation (possible values ReceiverLocationIndoors and ReceiverLocationOutdoors).	ReceiverLocation	None	See description
NumberOfSubstanceCriteria	READ ONLY. Returns the number of substance criteria.	Long		
SubstanceCriterion	READ ONLY. Takes an index as an argument and returns the substance criterion associated with the index.	SubstanceCriteria		
FlammableTargetData				
addTargetLevels()	Takes a flammable target level, as a fraction of LFL, as an argument.	Double	Fraction of LFL	> 0
clearTargetLevels()	Clears the flammable target levels.			
UserDefinedFlammabilities	If set to true allows user to specify the LFL and UFL of the flammable sunbstance.	Boolean	None	Yes/No
NumberOfTargetLevels	READ ONLY. Returns the number of flammable target levels.	Long		
TargetLevel	READ ONLY. Takes an index as a Long and returns the flammable target level associated with the index.	Double		
UpperFlammableLimit	Allows the Upper Flammable limit of current substance to be set.	Double		0 – 100
LowerFlammableLimit	Allows the Lower Flammable limit of current substance to be set.	Double		0 – 100
GroundTransferData – If default values are to be used then can be set "= New GroundTransferData".				
MeanDropRadius	Mean radius of gas cloud particles.	Double	Metres	> 0
SurfaceTemperature	Temperature of the ground surface.	Double	Kelvin	> 0

Property Name	Description	Type	Units	Valid Range
UseDeposition	Set to true to allow model to deposit mass to the ground	Boolean	None	Yes/No
UseHeatTransfer	Set to true to allow model to transfer heat to the ground.	Boolean	None	Yes/No
Preferences – If default values are to be used then can be set "= New Preferences".				
ContinuousMaximumSlice-Seperation	The maximum distance step of slices for the continuous model.	Double	Metres	> 0
ContinuousMinimumSlice-Seperation	The minimum distance step of slices for the continuous model.	Double	Metres	≥ 0
DistancePrecision	The precision with which to solve for distances to concentration and dose.	Double	Metres	> 0
Finite Duration Model Time Series Precision	Determines the accuracy to which DRIFT calculates the dose received in the finite duration and time-varying models. The smaller the value the more accurate the results, but the longer the model will take to run.	Double	None	≥ 10^{-9}
Finite Duration Model Time Series Extent	This parameter is used in the calculation of dose in the finite duration and time-varying models. It controls how far into the tails of the dose distribution to calculate the concentration time series. The number entered is internally multiplied by a typical timescale taken for the cloud to pass through the particular point of interest. Larger values of this parameter lead to more accurate results. The default value is 3.	Double	None	≥ 0
InstantaneousMaximumSlice-Seperation	The maximum distance step of slices for the instantaneous model.	Double	Metres	> 0
InstantaneousMinimumSlice-Seperation	Minimum slice separation for the continuous model.	Double		≥ 0
InverseBowdenRatio	The inverse Bowden ratio.	Double	None	
MinimumSegmentDuration	The minimum duration of a segment when converting from a a time varying release to discrete segments	Double	Seconds	> 0
NumberOfCentrelineResults	The number of results which will be calculated and used to generate a contour plot.	Long	None	> 0

Property Name	Description	Type	Units	Valid Range
ReleaseSegmentorTolerance	Sets the maximum fractional change in release rate between segments.	Double	None	> 0
SolverTolerance	Determines the accuracy to which DRIFT will solve its equation set.	Double	None	10^{-9}-10^{-1}
TablularOutputDecimal-Precision	The number of decimal places results will be given to in tables.	Long	None	> 0
TimePrecision	The precision when solving for the time of maximum concentration or maximum flammable volume etc.	Double	Seconds	> 0
ObstacleCollection – If no obstacles are needed then can be set "= New ObstacleCollection".				
addObstacle()	Adds an obstacle to model. Takes an Obstacle as an argument	Building, Fence or FiniteFence		
clearObstacles()	Clears all obstacles from the collection.			
NumberOfObstacles	READ ONLY. Returns the number of obstacles in the collection.	Long		
Obstacles	READ ONLY. Takes an index as an argument and returns the obstacle data associated with the index.	Obstacle		
Building				
Angle	The angle between North and the Length side of the building.	Double	Degrees	0 to 360
CentreX	The position of the centre of the obstacle in the x-direction.	Double	Metres	
CentreY	The position of the centre of the obstacle in the y-direction.	Double	Metres	
Height	The height of the obstacle.	Double	Metres	> 0
Length	The length of the obstacle	Double	Metres	
Name	The name of the obstacle.	String	None	Any String
Width	The width of the obstacle.	Double	Metres	
Fence				
Height	The height of the fence.	Double	Metres	> 0
Name	The name of the obstacle.	String	None	Any String
Position	The position of the fence.	Double	Metres	
FiniteFence – See Building. Properties are the same but no width is specified.				

3.0 Methods and Properties of DRIFT Model

Methods act on the DRIFT Model to either compute results or save/run/load files. They often require some form of input, which could be data or a filename, and then will return an output. Some of the methods on DRIFT model return cloud results and more detail on how to do so is given in section 6.0.

Method Name	Inputs	Returns	Description
run	None directly, but requires all necessary input information to be completed.	Nothing directly, but cloud results can then be evaluated.	This runs the solver for the given model scenario. A model can be run() after either inputting all necessary information or after load() of a previously saved file.
save()	Filename as string (including the .drift extension)	.drift File	When *save()* is called a html format .drift file is created in the file directory provided as input. This method saves all properties as well as any results from the solver if run() has been called.
load()	Filename as string (including the .drift extension)	Loads properties into VBA	When *load()* is called the .drift file is read into the VBA which can then be *run()* or properties can be checked or changed
loadLegacyFile()	Filename as string including the appropriate .DIN or .EJC extension	Loads up properties into VBA	Similar in function to the load method but enables loading of old DRIFT (version 2) and EJECT files.
computeFlammable-CentrelineResults	Requires input data to be entered and run() to have been called.	Flammable fraction of LFL along the centreline.	Computes the flammable centreline results so that flammable centreline results can be read or a contour plot can be made.
computeToxicCentre-lineResults	Requires input data to be entered and run() to have been called.	Toxic concentrations along centreline	Computes the toxic centreline results so that toxic centreline results can be read of a contour plot can be made.
cloud	Requires input data to be entered and run() to have been called.	Cloud results	Cloud contains many functions to return a large variety of calculations from DRIFT. These will be detailed below.
ModelVersion	None	Returns the current version string.	Returns a string detailing the precise version of DRIFT being used (e.g. “3.1.1”).

4.0 Methods and Properties of Cloud

Methods act on the DRIFTModel.Cloud to return information about the concentration and dose profiles. There are too many methods to list individually, so instead we present a broad outline of the naming conventions used. The methods fall into three categories:

- Methods to ask for concentration/dose results at a given location;
- Methods to calculate distances to a given concentration/dose;
- General purpose utility methods.

The following utility methods are available:

Method Name	Inputs	Returns	Description
`isTimeDependent`	None	Boolean	Returns true if the cloud concentration profile varies with time (e.g. instantaneous cloud); false otherwise (e.g. steady continuous).
`numberOfComponents`	None	Integer	Returns the number of distinct species present in the cloud, including air and water.
`numberOfLiquidPhases`	None	Integer	Returns the number of distinct liquid phases that can exist in the cloud.
`Substance(index)`	Integer	ISubstance	Returns the substance data associated with component **index**. Air is component 0; water is 1; the first contaminant is 2 etc.
`distancePrecision`	None	Double	Tells the cloud how accurately to compute distances to any given dose/concentration.
`setDistancePrecision`	Double	None	
`IsIndoors`	Boolean		Set this to true if indoor results are required; false if outdoors ones are required.
`ConcentrationAndDoseUnits`	IConcentrationField::Units		This can be set to **PPM_Min** or **Natural**. In the former case the units of concentration will be *ppm* and dose will be *ppm^n.min*, where *n* is the toxic exponent; in the later case the units of concentration will be *mol/mol* and the dose will be *(mol/mol)^n s*.
`isFlammable`	None	Boolean	Returns true if the cloud contains flammable materials; false otherwise.
`setToxicConcentrationCriteria(SubstanceName, Value)`	String, Double	None	Sets the current toxic concentration level of interest for **SustanceName** equal to **Value** (e.g. setToxicConcentrationCriteria("Ammonia", 100ppm). This value will be used when calculating distances to toxic concentrations.
`setToxicDoseCriteria`	String, Double	None	Sets the current toxic dose level of interest for **SustanceName** equal to **Value**.
`maximumTime`	None	Double	Returns the maximum time that the results are valid to.

Method Name	Inputs	Returns	Description
`maximumDownstreamDistance`	None	Double	Returns the maximum downstream distance that the results are valid to.
`maximumUpstreamDistance`	None	Double	Returns the maximum upstream distance that the results are valid to.
`maximumCrossStreamDistance (downstreamDistance, angleFromHorizontal)`	Double, Double	Double	Returns the maximum cross-stream distance that the results are valid to at the specified downstream distance (**s**) and angleFromHorizontal (**θ**). See below for more details.
`maximumCrossStreamDistance AtReceiverHeight(downstreamDistance, receiverHeight)`	Double, Double	Double	Returns the maximum cross-stream distance that the results are valid to at the specified downstream distance (**s**) and receiver height ($\mathbf{z_c}$). See below for more details.
`maximumHeight(time)`	Double	Double	Returns a height that corresponds to when the concentration profile becomes 'negligible' at a given time. This depends on the definition of 'negligible' of course, so this method should be used with caution.

Most of the remaining methods on the Cloud have the following naming convention:

- Methods whose names *begin* with the word **maximum** or **max** return the worst case cloud results over all time (these methods often give the worst case time via a passed-by-reference argument); methods that do not begin with **maximum** or **max** return the cloud results at a user specified time. For example:

  `concentration(substanceId, time, position)` returns the concentration at the given **position** of the substance referenced by **substanceId** (0=air; 1=water, 2=first contaminant etc.) at the given **time**.

  `maximumConcentration(worstCaseTime by ref, substanceId, position)` returns the worst case concentration at the given **position** of the substance referenced by **substanceId**. The **worstCaseTime** parameter is passed by reference and will be set by the code within the method.

- Methods ending in the words **AtDistance** return results based on a coordinate system (**s**, **d**, **θ**), where **s** is the downstream distance along the centreline trajectory of the cloud, **d** is the cross-stream distance perpendicular to this trajectory from the point specified by **s**, and **θ** is the angle that this cross-stream spoke makes to the horizontal. Coordinates in the form (**s**, **d**, **θ**) can be converted into standard Cartesian coordinates (**x, y, z**) via the **calculateCoordinates** method. The coordinate system can be slightly different depending on whether the user is interested in the worst case results or the results at a particular time of interest.

- Methods ending in the words **AtDistanceAndReceiverHeight** return results based on a coordinate system (**s**, **h**, $\mathbf{z_c}$), where **s** is the downstream distance along the centreline trajectory of the cloud, **h** is the horizontal cross-stream distance perpendicular to this trajectory from the point specified by **s**, and $\mathbf{z_c}$ is the receiver

height. Coordinates in the form (**s**, **h**, $\mathbf{z_c}$) can be converted into standard Cartesian coordinates (**x, y, z**) via the **calculateCoordinatesAtReceiverHeight** method. The coordinate system can be slightly different depending on whether the user is interested in the worst case results or the results at a particular time of interest.

- Methods containing the word **Flammable** work in terms of fractions of the lower flammable limit. For instance, the method **flammableConcentration** will return the fraction of the LFL at the specified time and location. Methods such as **downstreamDistanceToFlammableConc** take the LFL fraction of interest as an argument.

- Methods containing the word **Toxic** work in terms of toxic fraction, calculated relative to the concentration/dose criteria set by the user (see **setToxicConcentrationCriteria** and **setToxicDoseCriteria** above). For instance, a mixture of chlorine and bromine might have concentration criteria of 100ppm for chlorine and 200ppm for bromine. If the actual concentration was 300ppm for chlorine and 400ppm for bromine then the toxic fraction (using HSE's methodology) would be 300/100 + 400/200 = 5.

- Methods containing neither of the words **Flammable** or **Toxic** usually return just the results for a single component in whatever units have been set (e.g. ppm, see **ConcentrationAndDoseUnits** above). The **substanceId** is usually passed into these methods as an argument.

- Methods such as **downstreamDistanceToMaximumConc** return the downstream distance to the maximum concentration/dose. Usually the user will have to pass in an argument by reference to retrieve the value of the concentration/dose at this downstream distance (e.g. **downstreamDistanceToMaximumConc(maxConc, 30, 2)** will return the downstream distance to the maximum concentration of substance number 2 at 30s and **maxConc** will be set to the value of the concentration at this distance). The use of 'maximum' here does not necessarily indicate that the function is the worst case over all times because it is not at the start of the method name.

- Methods such as **downstreamDistanceToConc** return the distance along the cloud centreline (measured from the source) until the concentration falls below the value of interest. This is **s** in the coordinate system (**s**, **d**, **θ**) referred to above. **downstreamDistanceToConcAtReceiverHeight** is the same function but using the (**s**, **h**, $\mathbf{z_c}$) coordinates.

- Methods such as **halfWidthToConc** return the cross-stream distance **d** to the specified concentration level for a given **s** and **θ**.

- Methods such as **largestHalfWidthToConc** return the largest cross-stream distance **d** at a given **θ** and level of interest for all values of **s**. These methods usually require an argument to be passed in by reference to store the downstream distance corresponding to this largest half-width.

- The purpose of the other methods on the Cloud can be worked out from the above conventions.

Table and graph information can also be extracted through the COM interface using the following methods:

Method Name	Inputs	Returns	Description
HasResults	None	Boolean	Returns True if the cloud has results; False otherwise.
clearResults()	None	None	Clears the results.
NumberOfSegments	None	Integer	In the case of the time-varying model there may be multiple finite duration cloud segments that combine to give the overall time-varying results; for all other models there will only be one segment.
NumberOfInstantaneousResults **/** **NumberOfInstantaneousResults_2 (segment)**	None / Integer	Integer	Returns the number of instantaneous model results/time-steps associated with the cloud. In the case of a low momentum area source with dilution over the source then the continuous model may have also run an instantaneous model upfront. There is an overload of this function suffixed with a **_2** that also takes the segment number in the case that the time-varying model has been used.
InstantaneousResult(index) **/** **InstantaneousResult_2(segment, index)**	Integer, None / Integer	Instantaneous Slice	Returns the instantaneous result referenced by **index**. This will correspond to a single time-step/row in the output tables shown on the GUI.
NumberOfContinuousResults **/** **NumberOfContinuousResults_2(segment)**	None / Integer	Integer	Returns the number of continuous model results/distance-steps associated with the cloud. In the case of a pure instantaneous model run this will be always be zero.
ContinuousResult(index) **/** **ContinuousResult_2(segment, index)**	Integer, None / Integer	Continuous Slice	Returns the continuous result referenced by **index**. This will correspond to a single distance-step/row in the output tables shown on the GUI.

At each time/distance step the slice returned from **InstantaneousResult** or **ContinuousResult** can be used to access that value of any of the output variables at that step. The following methods are available:

Method Name	Inputs	Returns	Description
NumberOfComponents	None	Integer	Returns the number of components in the cloud (e.g. air, water, first contaminant etc.)
NumberOfLiquidPhases	None	Integer	Returns the number of distinct liquid phases that could have formed in the cloud.

Method Name	Inputs	Returns	Description
`NumberOfOutputVariables`	None	Integer	Returns the number of available output variables (e.g. temperature, density etc.)
`NumberOfComponentVariables`	None	Integer	Returns the number of available component variables. These are variables that can are stored on a pure component basis (e.g. mole fraction etc.)
`NumberOfLiquidPhaseVariables`	None	Integer	Returns the number of available liquid phase variables. These are variables that are stored independently for each liquid phase (e.g. liquid fraction etc.)
`OutputValue(var)`	Instantaneous/ Continuous Output Variable	Double	Returns the value of the output variable referenced by **var** that is stored in the slice.
`ComponentValue(var, compIndex)`	Instantaneous/ Continuous Component Variable	Double	Returns the value of the component variable referenced by **var** that is stored in the slice for component **compIndex**.
`LiquidPhaseValue(var, liqPhaseIndex)`	Instantaneous/ Continuous Liquid Phase Variable	Double	Returns the value of the liquid phase variable referenced by **var** that is stored in the slice for the specified liquid phase index **liqPhaseIndex**.

When calling any of the methods **OutputValue**, **ComponentValue** and **LiquidPhaseValue**, the intellisense should bring up a full list of available variables. A more detailed description of what each of these variables corresponds to can be obtained by creating a new instance of either an **InstantaneousOutputConverter** or a **ContinuousOutputConverter**, which have the following useful methods:

Method Name	Inputs	Return	Descripti
`outputVariableDescription(var)`	Instantaneous/ Continuous Output Variable	String	Returns a descriptive string associated with the output variable **var**.
`componentVariableDescription(var)`	Instantaneous/ Continuous Component Variable	String	Returns a descriptive string associated with the component variable **var**.
`liquidPhaseVariableDescription(var)`	Instantaneous/ Continuous Liquid Phase Variable	String	Returns a descriptive string associated with the liquid phase variable **var**.

An example of extracting tabular and graphical results via the COM interface is presented in Section 6.0.

5.0 Example of creating a DRIFT scenario in VBA

The following example is working VBA code which can be copied and pasted into the VBE. It enters hardwired data into DRIFT via the driftWrapper, runs and saves a .drift file onto the C drive. It can be extended to enter input data from a spreadsheet and/or run multiple scenarios. The locations of the ESR substance database and the SPI files need to be set as stated, or the code changed to their locations.

```
Sub Drift_Example()

    Dim driftModel As driftModel
    Set driftModel = New driftModel

    Dim inputData As DRIFTData
    Set inputData = New DRIFTData

    Dim dataSource As New SubstanceDatabase
    Call dataSource.setFilePath(DataSourceType_SRD, "C:\ESRSubstance.mdb")
    Call dataSource.setFilePath(DataSourceType_SPI, "C:\SPI_files")

    Dim myFilename As String
    myFilename = "C:\Save_Filename_One.drift"

    'Input the model parameters

    With inputData
        .heading = "Run one"
        .description = "Insert Description Here"
        Set .sourceData = setSourceData()
        Set .weatherData = setWeatherData()
        Set .toxicTargetData = setToxicTargetData()
        Set .flammableTargetData = setFlammableTargetData()
        Set .groundTransferData = New groundTransferData
        Set .Preferences = New Preferences
        Set .Obstacles = New obstacleCollection
    End With
    ‘setSourceData is a separate subroutine to set the source data and so on

    Dim isValid As Boolean
    Dim messages as String

    isValid = inputData.isDataValid(messages)
    If isValid = False Then MsgBox messages: Exit Sub
    ‘Checks to see if data is valid. If data is not valid a message box describes the error
and the sub is
    ‘exited

    Set driftModel.inputData = inputData

    ‘Run and save the model

    Call driftModel.RUN

    Call driftModel.Save(myFilename)

End Sub
‘_____________________________________________________________________

Function setSourceData() As sourceData

'This function sets the source data for the driftModel described in the subroutine above

    Dim contSourceData As ContinuousSourceData
    Set contSourceData = New ContinuousSourceData

        Dim steadyData As SteadyReleaseSourceData
        Set steadyData = New SteadyReleaseSourceData

        steadyData.UseFiniteDurationModel = False
        steadyData.ReleaseRate = 100#
        steadyData.duration = 1800#
        steadyData.startTime = 0#
```

```
    Set contSourceData = steadyData

        Dim mixData As mixtureData

            Dim gaseousMixData As New GaseousMixtureData

        Set mixData = gaseousMixData

        mixData.ContaminantMassFraction = 1#
        mixData.temperature = 294#

            Dim compositionData As compositionData

                Dim Database As SubstanceDatabase
                Set Database = New SubstanceDatabase

                Database.dataSource = DataSourceType_SRD
                Call Database.setFilePath(DataSourceType_SRD, "C:\ESRSubstance.mdb"

                    Dim singleSubstanceData As New SingleComponentCompositionData

                    Set singleSubstanceData.Contaminant = Database.Substance("Propane")

            Set compositionData = singleSubstanceData

        Set mixData.compositionData = compositionData

    Set contSourceData.mixtureData = mixData

        Dim contExitData As continuousSourceExitData

            Dim jetData As MomentumJetExitData
            Set jetData = New MomentumJetExitData

            jetData.angleFromHorizontal = 0#
            jetData.AngleFromNorth = 90#
            jetData.dischargeCoefficient = 1#
            jetData.OrificeRadius = 0.05

        Set contExitData = jetData

    Set contSourceData.exitData = contExitData

    Set contSourceData.TerminationCriteria = New ContinuousTerminationCriteria

    Set setSourceData = contSourceData

    Dim location As New ColumnVector

    location.X = 0#
    location.Y = 0#
    location.Z = 0#

    Set setSourceData.location = location

End Function
'_____________________________________________________________________________

Function setWeatherData() As weatherData

'This function sets the weather data for the driftModel described in the subroutine above

    Dim PasquillData As New PasquillWeatherData

        PasquillData.pasquillStability = Stability_D
        PasquillData.windSpeed = 5#

    Set setWeatherData = PasquillData

    setWeatherData.PerformAutomaticMixingHeightCalculation = True
    setWeatherData.relativeHumidity = 0.7
    setWeatherData.roughnessLength = 0.1
    setWeatherData.temperature = 294#
    setWeatherData.WindAngleFromNorth = 270#
    setWeatherData.windReferenceHeight = 10#

End Function
```

```
‘_____________________________________________________________________________

Function setToxicTargetData() As toxicTargetData

'This function sets the toxic target data for the driftModel described in the subroutine above
‘Altough this example uses Propane it is included for example.

    Set setToxicTargetData = New toxicTargetData

        Dim substData As New ToxicSubstanceData

            substData.SubstanceName = "Propane"
            substData.ToxicDoseExponent = 1#

        Call setToxicTargetData.addToxicSubstanceData(substData)

    Dim criteria As ToxicCriteria
    Set criteria = New ToxicCriteria

    Dim substCrit As SubstanceCriteria
    Set substCrit = New SubstanceCriteria

        substCrit.SubstanceName = "Propane"
        substCrit.Value = 200#
        Call criteria.addSubstanceCriterion(substCrit)

        criteria.location = ReceiverLocation_Indoors

    Call setToxicTargetData.addDoseCriteria(criteria)

    Call setToxicTargetData.addConcentrationCriteria(criteria)

    setToxicTargetData.AveragingTime = 0#
    setToxicTargetData.maximumExposureDuration = 1800#
    setToxicTargetData.IndoorsVentilationRate = 0#
    setToxicTargetData.IndoorsLagTime = 0#

End Function
‘_____________________________________________________________________________

Function setFlammableTargetData() As flammableTargetData

'This function sets the flammable target data for the driftModel described in the subroutine
above

    Set setFlammableTargetData = New flammableTargetData

    Call setFlammableTargetData.addTargetLevel(1#)

    setFlammableTargetData.UserDefinedFlammabilities = True

    setFlammableTargetData.UpperFlammableLimit = 100#

    setFlammableTargetData.LowerFlammableLimit = 5#

End Function
‘_____________________________________________________________________________
```

Note: A more rigorous demonstration of this code is available in the Drift_Interface spreadsheet.

On calling the method save() the error message “Attempted to read or write protected memory. This is often an indication other memory is corrupt.” is usually a sign that some aspect of the input data has not been set correctly. The function “IsDataValid” can be used on the DRIFT Data class before setting the DRIFT Data to the DRIFT Model, which will return a string of any reason why the input data would cause any problems when trying to save.

6.0 Returning Results

Once a DRIFT Model has been run, the results for a variety of cloud conditions can be returned to Excel. This can be done through the ComputeFlammableCentrelineResults / ComputeToxicCentrelineResults methods or through the many cloud functions on the DRIFT Model class. Excel can be used to easily loop through results, enter them into a spreadsheet and analyse the results.

An example has been included to demonstrate how a contour plot of the lower flammable limit can be created in Excel. The model first needs to be loaded or created and then run before either of the compute centreline results or cloud functions can be called. The following example shows how the flammable centreline results and the coordinates of the lower flammable limit at certain downstream distances can be set into a table on an excel spreadsheet.

```
Sub Output_Flammable_Results()

‘(Insert code to load and run DRIFT scenario. This could be the code from section 5.0)

Dim downstreamHazardRange As Double
downstreamHazardRange = model.Cloud.maxDownstreamDistanceToFlammableConc(1#)
 'downstream distance to 1LFL for all times

Dim upstreamHazardRange As Double
upstreamHazardRange = model.Cloud.maxUpstreamDistanceToFlammableConc(1#)
'upstream distance to 1LFL for all times

Dim flammableResults As ICentrelineResults
Set flammableResults = model.computeFlammableCentrelineResults_2( _
        -upstreamHazardRange, downstreamHazardRange, 50)
‘Computes the flammable centreline results

Dim distances As Range
Set distances = Worksheets("Example Output").Range("ExOutput")
‘Set a location for outputting results. This is just an example.

Dim nResults As Integer
nResults = flammableResults.NumberOfResults

Dim j As Integer

For j = 0 To nResults - 1
              distances.Cells(j + 1, 1).Value = flammableResults.result(j).DownstreamDistance
              distances.Cells(j + 1, 2).Value = flammableResults.result(j).Value(0)
              distances.Cells(j + 1, 3).Value =
flammableResults.result(j).CrossStreamDistance(0)
Next j
‘Writes the downstream distances, LFL values and cross stream distances to a range of cells in
Excel

Dim coords As ColumnVector

For j = 0 To nResults - 1
          Set coords = model.Cloud.calculateCoordinates(flammableResults.result(j) _
          .DownstreamDistance, flammableResults.result(j).CrossStreamDistance(0), 0)

               distances.Cells(j + 1, 4).Value = coords.X
               distances.Cells(j + 1, 5).Value = coords.Y
Next j
‘The calculateCoordinates() function calculates the coordinates of a particular occurrence as
‘specified by the arguments passed into it. This is used to generate the top part of a contour
plot.


Do While j > 0

               j = j - 1
          Set coords = model.Cloud.calculateCoordinates(flammableResults.result(j) _
         .DownstreamDistance, -flammableResults.result(j).CrossStreamDistance(0), 0)
```

```
                distances.Cells(2 * nResults - j, 4).Value = coords.X
                distances.Cells(2 * nResults - j, 5).Value = coords.Y

Loop
‘Sets the values of the bottom half of a contour plot into a table in Excel.

’*****************************************************************************************

‘Below is some example code to extract tabular results via the COM interface
‘In this case we write a table of cloud centreline height vs. time for the instantaneous model

Dim xVar As InstantaneousOutputVariable
Dim yVar As InstantaneousOutputVariable

xVar = InstantaneousOutputVariable_Time
yVar = InstantaneousOutputVariable_LocationZ

Dim xAxisTitle As String
Dim yAxisTitle As String
Dim converter As New InstantaneousOutputConverter

xAxisTitle = converter.outputVariableDescription(xVar)
yAxisTitle = converter.outputVariableDescription(yVar)

Dim xVals As Range
Dim yVals As Range

Set xVals = Worksheets("Tables").Range("xVals") ‘returns “Cloud Travel Time (s)”
Set yVals = Worksheets("Tables").Range("yVals") ‘returns “Cloud Z Location (m)”

xVals.Cells(1, 1) = xAxisTitle
yVals.Cells(1, 1) = yAxisTitle

Dim nResults As Integer
nResults = model.Cloud.NumberOfInstantaneousResults
Dim index As Integer
For index = 0 To nResults - 1
       Dim slice As InstantaneousSlice
       Set slice = model.Cloud.InstantaneousResult(index)

       xVals.Cells(index + 2, 1) = slice.OutputValue(xVar)
       yVals.Cells(index + 2, 1) = slice.OutputValue(yVar)
Next index

End sub
```

ESR Technology Ltd Whittle House, 410 The Quadrant, Birchwood Park, Warrington. WA3 6FW United Kingdom
Tel: +44 (0)1925 843 400 **Fax:** +44 (0)1925 843 500 **Email:** info@esrtechnology.com **Web:** www.esrtechnology.com