## PARTICLE_SIMULATION

```
! List of particles in simulation
TYPE (particle), VECTOR :: particles

! Initialize list of particles
VOID, SUBROUTINE initialize(concentration, intensity, T_environment, humidity, v_wind)
    ! Particle concentration in the air [x/m³] (0...30.000)
    INT :: concentration
    ! Intensity coefficient [%] (0...1)
    DP :: intensity
    ! Velocity (of Wind [m/s]
    DP, VECTOR:: v_wind
    ! Temperature of environment [K]
    DP :: T_environment
    ! (relative) Humidity in environment [%]
    DP :: humidit

! Update Particle
VOID, SUBROUTINE update(integration_procedure_evaporation, integration_procedure_movement, dt)
    ! Desired numeric integration procedure for evaporation (Euler, Runge-Kutta, ...)
    PROCEDURE(num_int_procedure) :: integration_procedure_evaporation
    ! Desired numeric integration procedure for movement (Euler, Runge-Kutta, ...)
    PROCEDURE(num_int_procedure) :: integration_procedure_movement
    ! Stepwidth [s]
    DP :: dt

! Give out Observations in memory
VECTOR_FUNCTION output(start_idx, end_idx)
    ! Select desired Particles by Index
    INT :: start_idx
    INT :: end_idx
```

**Manage Execution of Simulation**

## MODULE_EVAPORATION

```
! Update Particle Diameter [µm]
VOID, SUBROUTINE evaluate_evaporation(integration_procedure, prtcl, dt)
    ! Desired numeric integration procedure (Euler, Runge-Kutta, ...)
    PROCEDURE(num_int_procedure) :: integration_procedure
    ! Desired particle
    TYPE(particle) :: prtcl
    ! Stepwidth [s]
    DP :: dt

! Diffusion Coefficient [m²/s = = µm²/s * 10¹²]
DP, FUNCTION D_Coeff(T_particle, T_environment)
    ! Temperature of Particle T_prtcl and Environment T_env [K]
    DP :: T_particle, T_environment

! Calculate partial pressure of H2O at certain temperature [Pa = kg·m⁻¹·s⁻²]
DP, FUNCTION p0_H2O(T)
    ! Desired Temperature [K]
    DP :: T

! Calculate partial pressure of H2O in particle (fluid water) [Pa = kg·m⁻¹·s⁻²]
DP, FUNCTION pw_H2O(T_particle)
    ! Temperature of particle [K]
    DP :: T_particle

! Calculate partial pressure of H2O in environment (steam) [Pa = kg·m⁻¹·s⁻²]
DP, FUNCTION pinf_H2O(T_environment, humidity)
    ! Temperature of Environment [K]
    DP :: T_environment
    ! Humidity of environment [%]
    DP :: humidity

! Calculate Sherwood number [dimensionless]
DP, FUNCTION Sh(velocity, diameter, T_environment, T_particle)
    ! Temperature of environment and particle [K]
    DP :: T_environment, T_particle
    ! Velocity of particle [m/s]
    DP :: velocity
    ! Diameter of particle [µm]
    DP :: diameter

! Calculate Reynolds number [dimensionless]
DP, FUNCTION Re(velocity, diameter, T_environment)
    ! Velocity of particle [m/s]
    DP :: velocity
    ! Diameter of particle [µm]
    DP :: diameter
    ! Temperature of environment and particle [K]
    DP :: T_environment

! Calculate Schmidt number [dimensionless]
DP, FUNCTION Sc(T_environment, T_particle)
    ! Temperature of environment and particle [K]
    DP :: T_environment, T_particle

! Calculate mass transfer coefficient [µm/s]
DP, FUNCTION h_m(diameter, T_environment, T_particle, velocity)
    ! Temperature of environment and particle [K]
    DP :: T_environment, T_particle
    ! Velocity of particle [m/s]
    DP :: velocity
    ! Diameter of particle [µm]
    DP :: diameter

! 1st Order Derivative of particle diameter [µm/s]
VECTOR, FUNCTION dddt(y, params, params_dim, alg_dim)
        ! Dimension of result vector
        INT :: alg_dim
        ! Dimension of parameter vector
        INT :: params_dim
        ! Parameter vector
        DP, VECTOR:: params
        ! Vector of observables
        DP, VECTOR :: y
        ! Result vector
        DP, VECTOR :: dddt
```

**Calculate Evaporation of Particle-Shell for particular step**

## MODULE_MOVEMENT

```
! Update Particle Movement [location: m, velocity: m/s, acceleration: m/s²]
VOID, SUBROUTINE evaluate_movement(numeric_integration_procedure, prtcl, dt)
        ! Desired numeric integration procedure (Euler, Runge-Kutta, ...)
        PROCEDURE(num_int_procedure) :: numeric_integration_procedure
        ! Desired particle
        TYPE(particle) :: prtcl
        ! Stepwidth [s]
        DP ::: dt

! Function for Movement observables (position & velocity)
VECTOR, FUNCTION dydt(y, params, params_dim, alg_dim)
        ! Dimension of result vector
        INT :: alg_dim
        ! Dimension of parameter vector
        INT :: params_dim
        ! Parameter vector
        DP, VECTOR:: params
        ! Vector of overservables (Position [m] and Velocity [m/s])
        DP, VECTOR :: y
        ! Result Vector (Velocity [m/s] and Acceleration [m/s²]
        DP, VECTOR, RESULT :: dydt
```

**Calculate Movement of Particle for particular step**

## MODULE_FORCE

```
! Update Acting Force on Particle (neglelted in each step) [µN = µg*m/s²]
VOID, SUBROUTINE evaluate_force(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Reinitialize acting force [µN = µg*m/s²]
VOID, SUBROUTINE reset(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Calculate gravitational force [µN = µg*m/s²]
VOID, SUBROUTINE gravitation(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Calculate air resistance force [µN = µg*m/s²]
VOID, SUBROUTINE air_resistance(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Calculate wind force [µN = µg*m/s²]
VOID, SUBROUTINE wind(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl
```

**Calculate acting force on particle**

## MODULE_PARAMETERS

```
! Dimension of Simulation
INT, PARAMETER :: dim = 3

! Gravitational Acceleration [m/s²]
DP, DIMENSION(dim) :: g

! Gas Constant [kg·m²/(K¹·mol¹·s²)]
DP, PARAMETER :: R

! Standard Atmosphere [Pa = kg/(m·s²)]
DP, PARAMETER :: p_atm

! Molar Mass (Water) [kg/mol]
DP, PARAMETER :: M_H2O

! Molar Mass (Dry Air) [kg/mol]
DP, PARAMETER :: M_air

! Aesign Pi to constant Value
DP, PARAMETER :: PI

! Density of Air [kg/m³] [fg/µm³]
DP, FUNCTION rho_air(T_environment)
    ! Temperature of environment [K]
    DP :: T_environment

! Kinematic Viscosity [m²/s]
DP, FUNCTION nu_air(T_environment)
    ! Temperature of environment [K]
    DP :: T_environment

! Dynamic viscosity [Pa·s = kg/(m·s)]
DP, FUNCTION etha_air(T_environment)
    ! Temperature of environment [K]
    DP :: T_environment

! Density of Water [kg/m³] [fg/µm³]
DP, PARAMETER :: rho_H20

! Density of SARS-CoV-2 [kg/m³] [fg/µm³]
DP, PARAMETER :: rho_cov2

! Base Diffusion Coefficient at 0°C [m²/s]
DP, PARAMETER :: D_0

! Corresponding Temperature T_0 at 0°C [K]
DP, PARAMETER :: T_0

! Antoine Equation Parameters (H2O)
DP, PARAMETER :: A
DP, PARAMETER :: B
DP, PARAMETER :: C
DP, PARAMETER :: mmHg_Pa_conversion

! Maximum Velocity of Particles when sneezing [m/s]
DP, PARAMETER :: sneeze_vel
```

**Define global calculation parameters**

## MODULE_PARTICLE

```
! Custom Datatype for Particle-Simulation
TYPE particle
    ! Current Position [m]
    DP, VECTOR :: r

    ! Current Velocity [m/s]
    DP, VECTOR :: v

    ! Current Force [µN = µg·m/s]
    DP, VECTOR :: f

    ! Particle Core Data (d = diameter [µm])
    DP :: d_core

    ! Particle Shell Data (d = diameter [µm])
    DP :: d_shell

    ! Temperature of particle [K]
    DP :: T

    ! Elapsed time [s]
    DP :: time_elapsed

    ! Indicates if particle-shell is evaporated
    BOOL :: core_only

    ! Indicates if particle is still in the air
    BOOL :: active

    ! Velocity of Wind [m/s]
    DP, VECTOR :: v_wind

    ! Temperature of environment [K]
    DP :: T_environment

    ! Relative Humidity in environment [%]
    DP :: humidity
```

```
! Euclidic norm of velocity of particle [m/s]
DP, FUNCTION v_euclid(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Get mass of particle core [fg = femtogramm]
DP, FUNCTION mass_core(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Get volume of particle core [µm³]
DP, FUNCTION vol_core(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Get mass of particle shell [fg = femtogramm]
DP, FUNCTION mass_shell(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

!Get volume of particle shell [µm³]
FUNCTION vol_shell(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Verify if particle shell is completely evaporated
VOID, SUBROUTINE verify_shell(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Verify if particle is still active
VOID, SUBROUTINE verify_status(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl
```

**Define custom Datatype “Particle” that stores all relevant observables**

## MODULE_INIT

```
! Initialize position of particles [m]
VOID, SUBROUTINE initialize_position(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Initialize velocity of particles [m/s]
VOID, SUBROUTINE initialize_velocity(prtcl, intensity)
    ! Desired particle
    TYPE(particle) :: prtcl
    ! Velocity intensity coefficient [0...1]
    DP :: intensity

! Initialize structure of particle
VOID, SUBROUTINE initialize_structure(prtcl)
    ! Desired particle
    TYPE(particle) :: prtcl

! Initialize circumstances of particle
VOID, SUBROUTINE initialize_circumstances
        (prtcl, T_environment, humidity, v_wind)
    ! Desired particle
    TYPE(particle) :: prtcl
    ! Velocity of Wind [m/s]
    DP, VECTOR :: v_wind
    ! Temperature of environment [K]
    DP :: T_environment
    ! Relative Humidity in environment [%]
    DP :: humidity
```

**Semi-Automatic Initialization of Particle**

## NUMERIC_INTEGRATION

```
! Generic Function Interface
INTERFACE
    VECTOR, FUNCTION func(y, params, params_dim, alg_dim)
        ! Dimension of result vector
        INT :: alg_dim
        ! Dimension of parameter vector
        INT :: params_dim
        ! Parameter vector
        DP, VECTOR:: params
        ! Vector of observables
        DP, VECTOR :: y
        ! Result
        DP, VECTOR, RESULT :: dydx

! Runge-Kutta (2ⁿᵈ order)
VOID, SUBROUTINE runge_kutta_2k(f, y, dt, params, params_dim, alg_dim)
    ! Generic function variantie
    PROCEDURE(func) :: f
    ! Dimension of result vector
    INT :: alg_dim
    ! Dimension of parameter vector
    INT :: params_dim
    ! Parameter vector
    DP, VECTOR:: params
    ! Vector of observables
    DP, VECTOR :: y
```

```
! Generic Function Interface
INTERFACE
    VOID, SUBROUTINE num_int_procedure(f, y, dt, params, params_dim, alg_dim)
        ! Generic function variantie
        PROCEDURE(func) :: f
        ! Dimension of result vector
        INT :: alg_dim
        ! Dimension of parameter vector
        INT :: params_dim
        ! Parameter vector
        DP, VECTOR:: params
        ! Vector of observables
        DP, VECTOR :: y

! Runge-Kutta (4ᵗʰ order)
VOID, SUBROUTINE runge_kutta_4k(f, y, dt, params, params_dim, alg_dim)
    ! Generic function variantie
    PROCEDURE(func) :: f
    ! Dimension of result vector
    INT :: alg_dim
    ! Dimension of parameter vector
    INT :: params_dim
    ! Parameter vector
    DP, VECTOR:: params
    ! Vector of observables
    DP, VECTOR :: y

! Euler Computation
VOID, SUBROUTINE euler(f, y, dt, params, params_dim, alg_dim)
    ! Generic function variantie
    PROCEDURE(func) :: f
    ! Dimension of result vector
    INT :: alg_dim
    ! Dimension of parameter vector
    INT :: params_dim
    ! Parameter vector
    DP, VECTOR:: params
    ! Vector of observables
    DP, VECTOR :: y
```

**Generic Implementation of several numeric integration procedures**