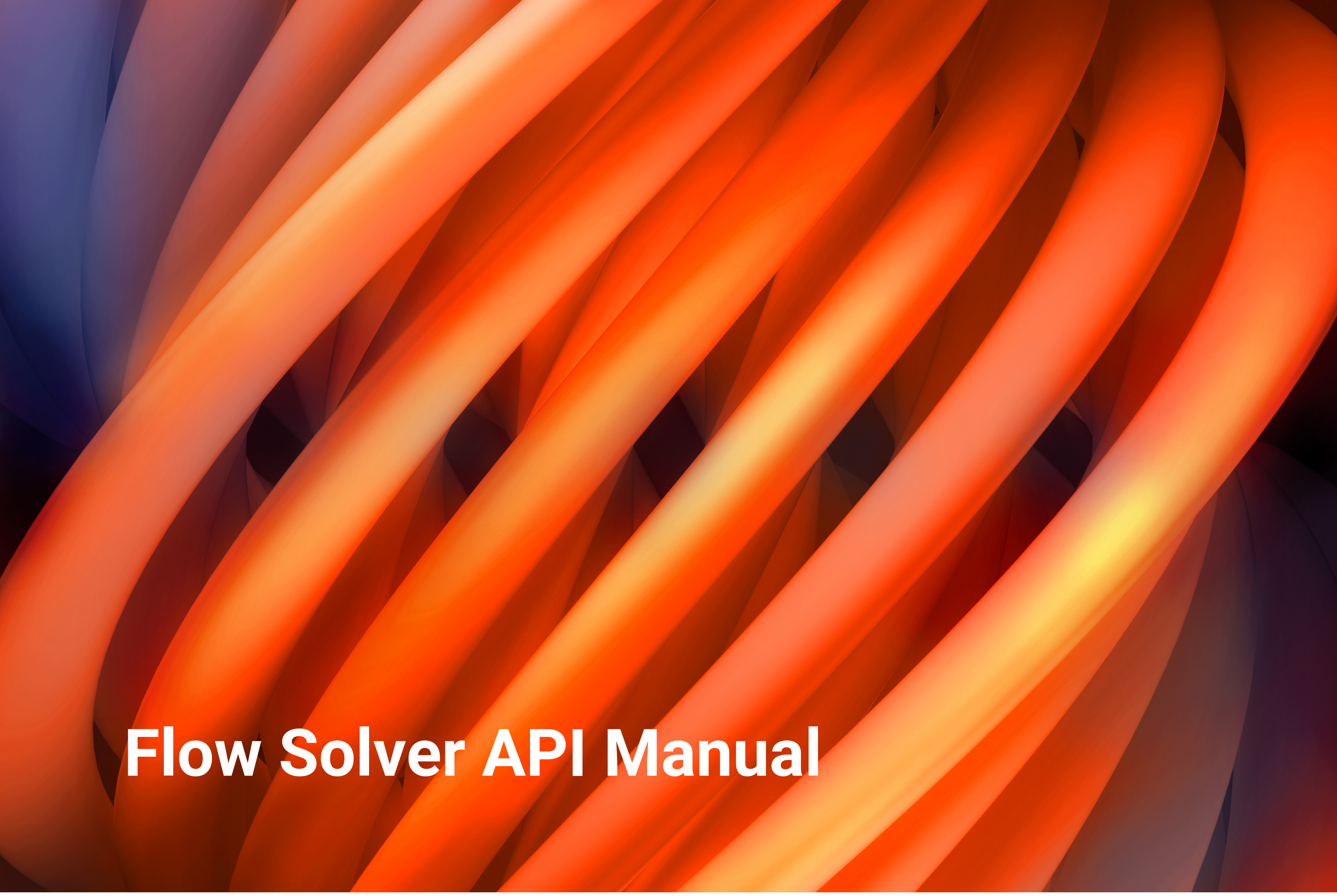


Flow Solver API Manual

June 2024

Maya HTT Better thinking Better future.

Proprietary & Restricted Rights Notice

Table of Contents

Introduction to the flow solver API

The flow solver Application Programming Interface (API) allows you a limited access to the internal data-structure and functionalities of the flow solver. You can create a user-defined plugin in the form of a dynamic-link library (dll) to extend the capabilities of the flow solver. This manual describes the additional capabilities that the flow solver API offers you, and how you can use those capabilities to create your user-defined plugin.

You can use this API to:

- Access some variables, material and physical properties and boundary conditions defined in your model.
- Write your own functions or correlations to define new boundary conditions, material properties or specific model parameters.
- Add specific source terms to scalar equations.
- Create and solve transport equations.

Flow solver plugin interface

The parallel flow solver lets you load a user-defined plugin in the form of dynamic-link library (dll) and incorporate the added functionalities. Upon the flow solver request, the dll file must provide a user-defined plugin interface with the object of type IFlowPlugin. You write the implementation of the plugin interface. You can find the prototype of the plugin interface in the [installation_path]\nxcae_extras\tmg\include\plugin_prototype\IFlowPlugin.h file. Use the following methods with dynamically loadable symbols in the dll file:

```
PLUGIN_LINKAGE IFlowPlugin* CreateInstance();
PLUGIN_LINKAGE void DeleteInstance(IFlowPlugin* plugin);
```

with the implementations in the following form:

```
PLUGIN_INTERFACE IFlowPlugin* CreateInstance()
{
   IFlowPlugin* plugin = new MyDerivedFlowPlugin();
   return plugin;
}
PLUGIN_INTERFACE void DeleteInstance(IFlowPlugin* plugin)
{
   delete plugin;
}
```

You can find definitions of PLUGIN_LINKAGE and PLUGIN_INTERFACE in the [installation_path]\nxcae_extras\tmg\include\plugin_prototype\pluginport.h file.

The following describes the interface methods:

```
// This function returns the current version to which the plug-in code is assumed to
link
virtual std::string GetVersion() = 0;

// This function is called by the solver after construction of the object. It could be
used to build the internal structure of the user defined plug-in class
virtual maya::cfd::ErrorFlag Construct(IFlowSolver* solver) = 0;

// This function is called by the solver once all the internal structure of the flow
solver is created and is available to be used
// It is only called once to allow the plug-in class to use the available data in the
flow solver, and initialize its internal data structure
virtual maya::cfd::ErrorFlag Initialize() = 0;

// This function is iteratively called by the solver at each non-linear iteration of a
transient time step, or at each steady state iteration
virtual maya::cfd::ErrorFlag Iterate() = 0;
```

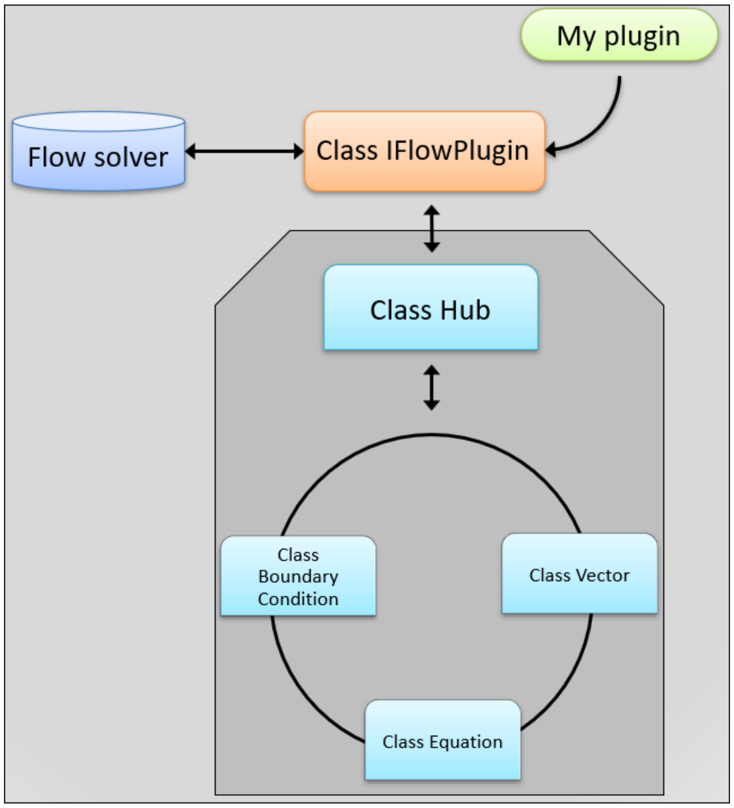
You can find the definition of maya::cfd::ErrorFlag in [installation_path]
\nxcae_extras\tmg\include\maya\cfd\CfdDefinitions.h file.

The API object IFlowSolver* solver is the gateway to the internal functionality of the parallel flow solver. The following sections describe how you can use this object and incorporate the implemented user plugin with the capabilities of the flow solver.

You can find an example of a user plugin for the flow solver, including the compilation scripts for both Linux and Windows operating systems in [installation_path]\nxcae_extras\tmg\plugin_examples\flow_solver folder.

Flow solver API classes

The C++ header files contain the API classes and methods that you can use in your plugin to expand the functionality of the flow solver for your particular requirements.



Class IFlowPlugin is an interface class that includes virtual commands that communicate with the flow solver to initialize, construct, iterate, and execute instances of class objects. **Class IFlowPlugin** is required for the API methods to interact with the flow solver. For more information, see Class IFlowPlugin.

The following API classes provide the framework for extending the functionality of the flow solver:

- **Class Hub** acts as the hub between **Class BoundaryCondition**, **Class Equation**, and **Class Vector**. It includes API methods that instantiate and construct the Hub object. The Hub object acts as an interface between the API classes and the flow solver and can access data defined in your flow model. For more information, see Class Hub.
- **Class BoundaryCondition** includes API methods that let you retrieve and control boundary conditions that are created as loads, constraints, and simulation objects in your flow model. For more information, see Class BoundaryCondition.
- **Class Equation** includes API methods that let you create and solve generic transport equations. For more information, see Class Equation.
- **Class Vector** includes API methods that let you retrieve vector values, such as flow velocity, in your flow model. For more information, see maya::cfd definitions.

The type of enumerators you can use as input arguments for the API method in your plugin are listed in the CfdDefinitions file. For more information, see maya::cfd definitions.

Class Hub

Class Hub in namespace maya::cfd is the interface between the API and the flow solver.

The following API methods are available in **Class Hub**:

- Time step methods that access flow simulation time values, current time step size, and the energy time step multiplier.
- Simulation methods that access the temperature and pressure offset in the current independent fluid domains.
- Vector methods that populate an array with pointers to the object of type maya::cfd::Vector and access the maya::cfd::variable values defined separately for each independent fluid domain your solution.
- An equation method that creates and returns an instance of the object of type maya::cfd::Equation that lets you solve a generic transport equation. For more information, see Class Equation.
- A boundary condition method that returns a pointer to an object of type maya::cfd::BoundaryCondition that lets you retrieve boundary conditions and constraint values. For more information, see Class BoundaryCondition.
- Track report methods that access track report simulation objects defined for the solution's current independent fluid domain.
- Parallel process communication methods that access the current parallel process to which the current instance of **Class IPlugin** is associated and calculate the sum of double values for a local variable that is distributed over all processes.

Hub methods

Note

For the code examples in this section, it is assumed that you have instantiated and defined a `maya::cfd::Hub` object named `myServer` in your plugin.

std::string GetVersion()

Returns the current version of the flow solver. It is important to check the version that is returned with this method is up-to-date with any updates to the flow solver API.

double GetRelaxationFactor()

Returns a relaxation factor value that you can specify in your plugin classes to increase the stability of the flow solver solution. The relaxation factor is defined using the RELAX_PLUGIN advanced parameter in the *user.prm* file. For more information, see Setting plugin parameters.

double GetCurrentTime()

Returns the current flow simulation time that is defined for your solution.

double GetCurrentTimeStepSize()

Returns the current time step size that is used for advancing the simulation to the next simulation time. The next simulation time equals the current simulation time plus the current time step size.

double GetEnergyTimeStepMultiplier()

Returns the energy time step multiplier. To accelerate the convergence in models that are mostly steady state (solved variables that do not vary with respect to time) with a slow thermal process, a different time step size is used for advancing the energy equation.

The energy equation time step size is obtained by multiplying the current time step size with the multiplier.

```
energy equation time step size=
GetCurrentTimeStepSize()*GetEnergyTimeStepMultiplier
```

size_t GetEnclosureID()

Returns the independent fluid domain identification index in the current model.

Example

size_t flowEnclId = myServer.GetEnclosureID()

void StopSimulation()

Stops the simulation run.

void EnableEnergy()

Instantiates the flow solver energy transport equation, together with other transport equations, if the equation was not initiated for your solution. The flow solver only solves an energy equation when a temperature constraint or a volumetric heat load is defined. Using this method, the flow solver solves the energy equation, regardless of a temperature constraint or a volumetric heat load.

Example

```
myServer.EnableEnergy()
```

void DisableScalars()

Excludes the passive scalar transport equations from the solution list at each iteration in the current simulation.

This method is provided for API classes that interact with specific models, in which the passive scalars are solved using proprietary models.

double GetCurrentPressureOffset()

Returns the current value of the pressure offset in the current independent fluid domain. The pressure field is stored in a relative form.

Returning absolute pressure

To return the absolute pressure, you include a method in an API class that adds the current value of the offset pressure to the relative pressure.

The current value of the pressure offset can vary in time, and from one independent fluid domain to another.

To return the relative pressure, use the following command:

```
GetVector(maya::cfd::Variable::PRESSURE);
```

double GetTemperatureOffset()

Returns the current value of the temperature offset in the current independent fluid domain as a fraction or whole number. The temperature field is stored in a relative form.

Returning absolute temperature

To return the absolute temperature, the current value of the temperature offset must be added to the relative temperature.

The current value of the temperature offset can vary in time and from one independent fluid domain to another.

To return the relative temperature, use the following command:

```
GetVector(maya::cfd::Variable::TEMPERATURE);
```

Vector* GetVector(maya::cfd::Variable vectorTitle, int species = -1)

Returns a pointer to an object of type maya::cfd::Vector that grants access to the maya::cfd::variable values that are defined separately for each independent fluid domain in your solution.

Input arguments	**Type**	**Description**
vectorTitle	maya::cfd::Variable	The name of the variable from which you will access the field values that you defined for each independent fluid domain in your solution. The supported variables are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd Variable enumerators.
species	int	Optional and only permitted for variables that are related to the species. The integer corresponds to the index of the species in the mixture. -1 is the default and corresponds to a single gas configuration.

Example

```
int gasIndex = 1;
```

```
maya::cfd::Vector* massFracGas =
myServer.GetVector(maya:cfd::Variable::GAS_MASS_FRACTION, gasIndex);
maya::cfd::Vector* density = myServer.GetVector(maya:cfd::Variable::DENSITY);
```

`void GetGradientVector(Vector* gradient[3], maya::cfd::VariableGradient vectorTitle, int species = -1)`

Populates an array with pointers to the object of type maya::cfd::Vector. To populate the gradient vector field, you must declare a gradient table with three rows. Each row will contain a vector pointer that represents three spatial components, for example, x, y and z.

Input arguments	Type	Description
gradient	array of 3 maya::cfd::Vector*	The container that will store the gradient vector field.
vectorTitle	maya::cfd::VariableGradient	The name of the gradient variable values from which you will access the field values that you defined for each independent fluid domain in your solution. The supported gradient variables are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd VariableGradient enumerators.
`species`	int	Optional and only permitted for variables that are related to the species. The integer corresponds to the index of the species in the mixture. -1 is the default and corresponds to a single gas configuration.

Example

```
int gasIndex = 1;

maya::cfd::Vector* massFracGradGas[3];

myServer.GetGradientVector(massFracGradGas1, maya::cfd::Variable::GAS_MASS_FRACTION,
gasIndex);

maya::cfd::Vector* density[3];
```

```
myServer.GetGradientVector(density, maya::cfd::Variable::DENSITY);
```

```
Equation* CreateNewEqation(std::string name,maya::cfd::ErrorFlag& errHandle)
```

Creates and returns an instance of the object of type maya::cfd::Equation. The maya::cfd::Equation object is a transport equation that is solved by the flow solver. You can set the properties of the transport equation using the equation methods defined in **Class Equation**. For more information, see Equation methods.

Input argument	Type	Description
name	std::string	Specifies the unique name of the equation to be created and solved.

Output arguments	Type	Description
errHandle	maya::cfd::ErrorFlag	References the ErrorFlag object that you must define for your plugin, which will be used to report any issues that occurred when creating the equation

Example

```
maya::cfd::ErrorFlag errHandle = {true, ""};
```

```
maya::cfd::Equation* myEquation = myServer.CreateNewEqation(“myCustomEquation”, errHandle);
```

```
size_t GetNumberOfCfdBcs(maya::cfd::BcType bcType)
```

Returns the number of boundary conditions for the specified `maya::cfd::bctype` condition type that are defined in the current independent fluid domain in your solution. For more information on identifying the current independent fluid domain, see size_t GetEnclosureID().

Input argument	Type	Description
`bcType`	maya::cfd::bctype	References the boundary condition type you defined for your solution. The supported boundary condition types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd BcType enumerators.

Example

```
size_t numOfInlet = myServer.GetNumberOfCfdBcs(maya::cfd::BcType::INLET);
```

`BoundaryCondition* GetCfdBc(maya::cfd::BcType bcType, size_t index)`

Returns a pointer to an object of type maya::cfd::BoundaryCondition that lets you control certain boundary conditions in the API classes that are created as simulation objects in your solution.

Input arguments	Type	Description
bcType	maya::cfd::bctype	Specifies and references the boundary condition type you defined for your solution. The supported boundary condition types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd BcType enumerators.
index	size_t	References an index in the array of the specified boundary conditions. The value of the index varies from zero to the number of boundary conditions, minus one, of the specified type in the current independent fluid domain, which is obtained by calling the *size_t* GetNumberOfCfdBcs(maya::cfd::BcType bcType) API method.

Note

This method is not intended to provide access to the boundary conditions created and specified in `maya::cfd::Equation.` For more information, see Class Equation.

Example

```
size_t openingBcIndex = 2;
```

```
maya::cfd::BoundaryCondition* openingBc2 =
myServer.GetCfdBc(maya::cfd::BcType::OPENING, openingBcIndex);
```

```
size_t GetNumberOfCfdTrackReports()
```

Returns the number of track report simulation objects defined for the solution's current independent fluid domain.

```
std::string GetCfdTrackReportName(size_t reportIndex)
```

Returns the track report simulation object name for the specified index.

Input argument	Type	Description
reportIndex	*size_t*	References an index in the array of the specified track report simulation objects. The index value varies from zero to the number of track reports minus one. `reportIndex = GetNumberOfCfdTrackReports - 1`

```
double GetMinForCfdReportVariable(size_t reportIndex, maya::cfd::Variable
variableName, int species = -1)
```

Returns the minimum value attained by the specified variable, in the region where the specified track report simulation object is defined.

Input arguments	Type	Description
variableName	maya::cfd::Variable	Specifies the name of the variable type defined in your solution from which you want to retrieve the minimum value. The supported variable types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd Variable enumerators.
species	int	Optional and only permitted for variables that are related to the species. The integer corresponds to the index of the species in the mixture. -1 is the default and corresponds to a single gas configuration.

Output argument	Type	Description
reportIndex	size_t	References an index in the array of the specified track report simulation objects. The index value varies from zero to the number of track reports minus one. `reportIndex = GetNumberOfCfdTrackReports - 1`

```
double GetMaxForCfdReportVariable(size_t reportIndex, maya::cfd::Variable
variableName, int species = -1)
```

Returns the maximum value, obtained by the specified variable, in the region where the specified track report simulation object is defined.

Input arguments	Type	Description
variableName	maya::cfd::Variable	Specifies the name of the variable type defined in your solution from which you want to retrieve the minimum value. Thesupported variable types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd Variable enumerators.
species	integer	Optional and only permitted for variables that are related to the species. The integer corresponds to the index of the species in the mixture. -1 is the default and corresponds to a single gas configuration.

Output argument	Type	Description
reportIndex	size_t	References an index in the array of the specified track report simulation objects. The index value varies from zero to the number of track reports minus one. `reportIndex = size_t GetNumberOfCfdTrackReports - 1`

```
double GetAvgForCfdReportVariable(size_t reportIndex, maya::cfd::Variable
variableName, int species = -1)
```

Returns the average value, obtained by the specified variable, in the region where the specified track report simulation object is defined.

Input arguments	Type	Description
variableName	maya::cfd::Variable	Specifies the name of the variable type defined in your solution from which you want to retrieve the minimum value. The supported variable types are defined in in the *CfdDefinitions.h* file. For more information, see maya::cfd Variable enumerators.
species	int	Optional and only permitted for variables that are related to the species. The integer corresponds to the index of the species in the mixture. -1 is the default and corresponds to a single gas configuration.
reportIndex	size_t	References an index in the array of the specified track report simulation object. The index value varies from zero to the number of track reports minus one. `reportIndex = GetNumberOfCfdTrackReports - 1`

```
size_t GetProcessId()
```

Returns the ID of the current parallel process to which the current instance of **Class IPlugin** is associated.

In parallel runs, a simulation can be run over multiple processes. Each process creates and attaches a new instance of the **Class IPlugin** for each of the independent fluid domains in the model. For example, when a simulation for a model includes two independent fluid domains, and runs on four processes, eight instances of **Class IPlugin** are created and linked to the simulation. Two instances are created for each process.

`void SumAll(double* localValues, double* globalValues, size_t numVals)`

Computes the sum of double values for a local variable that is distributed over all processes. Its form is similar to the MPI function MPI_Reduce.

To employ this method, you must declare arrays for localValues and globalValues and allocate sufficient memory to store the double number values (numValues). The method performs the following:

1. Accesses the variables in the localValues array. These values are distributed over all processes.
2. Computes the sum of localValues that are type double and are of the same size.
3. Returns the sum to the array globalValues.

Input arguments	**Type**	**Description**
localValues	double*	References a pointer to the values in the localValues array.
numVals	size_t	References the index in the array that stores the sum of the values.

Output argument	**Type**	**Description**
globalValues	double*	References a pointer to the values in the globalValues array.

Example

```
// This example has two processes (with ID #0 and #1)
size_t dim = 4;

if (myServer.GetProcessId() == 0)
{
double localValue [dim] = {1,2,3,4};
double globalValue [dim] = {0,0,0,0};

myServer.SumAll(localValue, globalValue, dim);
}
else if (myServer.GetProcessId() == 1)
{
double localValue [dim] = {5,6,7,8};
double globalValue [dim] = {0,0,0,0};
myServer.SumAll(localValue, globalValue, dim);
}

// After the SumAll calls on each process, the globalValue = {1+5=6, 2+6=8, 3+7=10,
4+8=12} on both processes #0 and #1,
```

Class BoundaryCondition

Class BoundaryCondition instantiates and constructs the boundary conditions object class that includes methods that you can define in your plugin to retrieve boundary conditions and constraint values. The flow solver interprets loads, constraints, and simulation objects as boundary conditions.

Each instance of `maya::cfd :: BoundaryCondition` represents a simulation object or a load that you created for your solution.

Using the boundary conditions class, you can define methods that retrieve:

- The name of the boundary condition defined in your solution.
- The number of selected elements and nodes for a specific boundary condition.
- The global node index in a nodal fluid domain vector of the current independent fluid domain.
- The volume of the control volume for a given local node index and the volume of an element in the boundary condition element set. You can use this information for volumetric calculations.
- A vector of boundary condition value vectors.
- The element average value for a given element in the boundary condition selection.

BoundaryCondition methods

Note

For the code examples in this section, it is assumed that you have instantiated and defined a `maya::cfd::Hub` object named `myServer` in your plugin.

BoundaryCondition(ISolver * master, maya::cfd::BcType type, size_t handle)

Creates a boundary condition object using the constructor method of this object.

Input arguments	Type	Description
master	ISolver*	References the solver, which is accessible through the maya::cfd::Hub object. For more information, see IFlowPlugin methods.

Input arguments	Type	Description
`Type`	maya::cfd::BcType	References the boundary condition type you defined for your solution. The supported boundary condition variables are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd Variable enumerators.

Output arguments	Type	Description
handle	size_t	An index that is attributed to the new boundary condition, which will reference it within the list of specified boundary condition types.

Example

```
size_t newBcIndex = 0;
```

```
maya::cfd::BoundaryCondition myNewOpeningBc(mySever.GetMaster(),
maya::cfd::BcType::OPENING, newBcIndex);
```

BoundaryCondition* GetCfdBc(maya:cfd::BcType bcType, size_t index)

Returns the boundary conditions that are defined in your solution.

Input arguments	Type	Description
bcType	maya::cfd::BcType	References the boundary condition type you defined for your solution. The supported boundary condition types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd BcType enumerators.

Input arguments	Type	Description
index	size_t	References an index in the array of the specified boundary condition types. The value of the index varies from zero to the number of boundary conditions, minus one, of the specified type in the current independent fluid domain, which is obtained by calling the `size_t GetNumberOfCfdBcs(maya::cfd::BcType bcType)` API method.

Example

```
size_t openingBcIndex = 2;
```

```
maya::cfd::BoundaryCondition* openingBc2 =
myServer.GetCfdBc(maya::cfd::BcType::OPENING, openingBcIndex);
```

std::string GetName():

Returns the name of the boundary condition (simulation object, load, or constraint) defined in your solution.

size_t GetNumberOfElements():

Returns the number of selected elements for a specific boundary condition.

Each boundary condition is defined over a set of elements.

size_t GetNumberOfLocalNodes():

Returns the number of selected nodes for a specific boundary condition.

The element set is comprised of a set of nodes.

size_t GetNodeIndexInVector(size_t n):

Returns the global node index in the fluid domain, using the local node index in the boundary condition selection.

Each fluid domain is decomposed into many partitions. Each process has one partition of the fluid domain. The nodal solution vectors are locally defined over an entire partition. The node index in a boundary condition set is different from the one in the nodal solution vectors. This method returns the index of a node in the nodal solution vector. You can use this information to obtain solution values such a pressure, velocity or temperature for a given node.

Input argument	**Type**	**Description**
n	size_t	Returns the global node index in a nodal fluid domain vector of the current independent fluid domain, using the local node index in the boundary condition selection.

double GetNodeVolume(size_t n):

Returns the volume of the control volume for a given local node index.

You can use this information for volumetric calculations.

Input argument	**Type**	
n	size_t	Returns the global node index in a nodal fluid domain vector of the current independent fluid domain, using the local node index in the boundary condition selection.

std::vector<Vector*> GetBcValueVectors():

Returns a vector of boundary condition value vectors.

A value vector corresponds to the elemental value of a specific physical quantity defined at the boundary condition.

The indices are element indices. These elements are part of an element set that define the boundary selection.

You can use this vector to control the boundary condition values during the solution process.

The following lists the values that can be modified for each boundary condition type.

Boundary Condition Type	Number of Components	Vector Description
WALL	UNSUPPORTED	UNSUPPORTED
INLET	UNSUPPORTED	UNSUPPORTED
OUTLET	UNSUPPORTED	UNSUPPORTED
OPENING	UNSUPPORTED	UNSUPPORTED
VOLUMETRIC_HEAT_GENERATION	1	Volumetric Heat Source

double GetVectorElementValue(size_t elemIndex, Vector* vec):

Returns the element average value for a given element in the boundary condition selection.

Input argument	Type	Description
vec	Vector*	Specifies a pointer to nodal field vector for a maya::cfd::Variable that was obtained with the `Vector* GetVector(maya::cfd::Variable vectorTitle, int species = -1)` method, defined in **Class Hub**. For more information, see Class Hub.
elemIndex	size_t	References the local index of the element in the specified boundary condition set.

Example

```
size_t volHeatGenBcIndex = 0;

maya::cfd::BoundaryCondition* volHeatGenBc0 =
myServer.GetCfdBc(maya::cfd::BcType::VOLUMETRIC_HEAT_GENERATION,
volHeatGenBcIndex);

maya::cfd::Vector* heatLoad = myBc->GetBcValueVectors()[0];

size_t elemIndex = 1200;

double heatLoadAtElem= myBc.GetVectorElementValue(elemIndex,
heatLoad);
```

double GetElementVolume(size_t elemIndex):

Returns the volume of an element with the index `elemIndex` in the boundary condition element set.

You can use this information for volumetric calculations.

Input arguments	Type	Description
elemIndex	size_t	References the local index of the element in the specified boundary condition element set.

Example

```
size_t openingBcIndex = 1;

maya::cfd::BoundaryCondition* openingBc1 =
myServer.GetCfdBc(maya::cfd::BcType::OPENING, openingBcIndex);

size_t elemIndex = 10;

double elemVol = openingBc1.GetElementVolume(elemIndex);
```

Class Equation

Class Equation includes API methods that let you set, initialize, and solve a custom generic transport equation within the flow solver. The equation object is constructed in **Class Hub** and must be called from this API class.

The **Class Equation** includes API methods that:

- Initialize and solve the defined maya::cfd::Equation object within the flow solver.
- Retrieve a vector object pointer containing the nodal value of the equation parameter in the solution.

You can define vector methods that:

- Return the nodal vector of the custom scalar variable for which the equation is solved.
- Populate a three-row array of maya::cfd Vector pointers with the nodal gradient of the custom scalar variable.

You can also define boundary condition methods that:

- Retrieve the number of boundary conditions associated to the equation object.
- Define how a custom equation scalar variable is set at a specified boundary condition of a particular type.
- Specify the value of the scalar at the boundary.

Equation methods

Note

For the code examples in this section, it is assumed that you have instantiated and defined a maya::cfd::Hub object named myServer in your plugin.

Equation(IEquation* equation, ISolver * master, maya::cfd::ErrorFlag& errHandle)

Creates an equation object using the constructor method of this object. The maya::cfd::Equation object is a transport equation that is solved by the flow solver.

Input arguments	Type	Description
equation	IEquation*	Pointer to an IEquation object instance, which is the interface class for the transport equation solver
master	ISolver*	Pointer to an object of type ISolver, which is the interface to the flow solver

Output argument	Type	Description
errHandle	maya::cfd::ErrorFlag	References the object of type maya::cfd::ErrorFlag that you must define for your plugin. This will be used to report any issues that occurred when creating the equation.

Vector* GetVariableVector():

Returns the nodal vector of the custom scalar variable for which the equation is solved.

void GetVariableGradientVector(Vector* gradient[3])

Populates a three-row array of maya::cfd::Vector pointers with the nodal gradient of the custom scalar variable.

Each maya::cfd::Vector in the array corresponds to a spatial component of the nodal gradient, for example, x, y, and z.

Input argument	Type	Description
gradient	array of 3 maya::cfd::Vector*	The container that will store the gradient vector field.

size_t GetNumberOfBCs(maya::cfd::BcType bcType)

Returns the number of boundary conditions of type maya::cfd::BcType associated to the maya::cfd::equation object.

Input argument	Type	Description
bcType	maya::cfd::BcType	References the boundary condition type you defined for your solution. The supported boundary condition types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd BcType enumerators.

void SetBcMethod(maya::cfd::BcType bcType, maya::cfd::BcMethod bcMethod, *size_t* bcIndex)

Defines how the custom equation scalar variable is set at a specified boundary condition of a particular type. You can set the scalar variable as a specified value or assume to match a zero normal gradient profile at the boundary.

Input argument	**Type**	**Description**
bcType	maya::cfd::BcType	Specifies the boundary condition type that is used in the solution. The supported boundary condition types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd BcType enumerators.
bcMethod	maya::cfd::BcMethod	Specifies the type of method to define the equation scalar variable at the desired boundary. The supported boundary condition types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd BcMethod enumerators.
bcIndex	size_t	References the boundary condition index with which to set the method from the list of boundary conditions with the same type.

Example

```
maya::cfd::ErrorFlagerrHandle = {true, ""};

std::string newEquationName = “MY_EQUATION”;

maya::cfd::Equation*
myNewEquation = myServer.CreateNewEquation(newEquationName, errHandle);


myNewEquation.SetBcMethod(maya::cfd::BcType::INLET,
maya::cfd::BcMethod::
VALUE_SPECIFIED, 0);
```

Vector* GetBcVector(maya::cfd::BcType bcType, *size_t* bcIndex)

Returns a vector object pointer containing the element face value of the scalar at the specified boundary condition. You can use this vector to specify the value of the scalar at the boundary, providing the boundary condition method that points to the `VALUE_SPECIFIED` type in the *CfdDefinitions.h file.* For more information, see maya::cfd BcMethod enumerators.

Input argument	**Type**	**Description**
bcType	maya::cfd::BcType	Specifies the boundary condition type in your solution. The supported boundary condition types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd BcType enumerators.
bcIndex	size_t	References the boundary condition index to retrieve the element face values from the list of boundary conditions with the same type.

Vector* GetParameterVector(maya::cfd::EquationParameter param)

Returns a vector object pointer containing the nodal value of the specified equation parameter in the solution.

An equation parameter is the value of an equation coefficient or a source/sink term in this equation. The supported equation parameter types are defined in the *CfdDefinitions.h* file. For more information, see maya::cfd EquationParameter enumerators.

Input argument	**Type**	**Description**
param	maya::cfd::EquationParameter	Specifies the equation parameter defined in your solution.

Example

```
maya::cfd::ErrorFlag errHandle = {true, ""};
std::string newEquationName = “MY_EQUATION”;
maya::cfd::Equation* myNewEquation = myServer.CreateNewEquation(newEquationName,
errHandle);

maya::cfd::Vector* eqSrceTerm =
myNewEquation.GetParameterVector(maya::cfd::EquationParameter:: VOLUMETRIC_SOURCE_TERM);
```

Class Vector

Class Vector lets you manage nodal or elemental field maya::cfd variable values that are defined separately for each independent fluid domain in your solution domain. You can use these values in your plugin for statistical calculations.

For multiple processes and multiple independent fluid domain simulations, the number of **Class IPlugin** instances is equal to the number of processes multiplied by the number of independent fluid domains. As a consequence, each instance of IPlugin and, thereby, each maya::cfd::Vector object contains only a local portion of the all flow domain.

Instances of **Class Vector** can only be obtained through other API classes, such as **Class Hub**, **Class Equation**, or **Class BoundaryCondition**. You cannot create new instances through its constructor.

The API methods in **Class Vector**:

- Grant access to read or modify vector values.
- Retrieve the size of the part of a vector that is local to the current process and specific to the current independent fluid domain.
- Retrieve the minimum and maximum value across all processes that are specific to the current independent fluid domain.
- Retrieve the algebraic average value across all processes that are specific to the current independent fluid domain.

Vector methods

Note

For the code examples in this section, it is assumed that you have instantiated and defined a maya::cfd::Hub object named myServer in your plugin.

double operator[](const *size_t* index)

The bracket operator in this method grants read access to the vector values.

Input argument	Type	Description
index	size_t	References an index of the value in the vector and ranges from zero to the size of the vector minus one.

virtual double& operator[](const *size_t* index)

The bracket operator in this method grants edit access to the vector values, meaning you can modify the values.

Input argument	Type	Description
index	size_t	References an index of the value in the vector and ranges from zero to the size of the vector minus one.

Note

Some of the vectors that are part of the solution field, such as velocity and pressure, are read only vectors and cannot be modified by the mean of this operator.

The following is an example of using the bracket operator with vector objects:

```
// Example of how to use bracket operator with vector objects

maya::cfd::Vector& density = *(myServer.
GetVector(maya::cfd::Variable::DENSITY));

for(size_t i=0; i < density.GetSize(); ++i)
{

std::cout<<"density at node "<<i<<" =
"<<density[i]<<std::endl;

}
```

size_t GetSize()

Returns the size of the vector object that is local on the local processor.

In parallel runs, a simulation can be run over multiple processes. Each process creates and attaches a new instance of the Class IPlugin for each of the independent fluid domains in the model. For example, when a simulation for a model includes two independent fluid domains, and runs on four processes, eight instances of Class IPlugin are created and linked to the simulation. Two instances are created for each process.

double GetMin()

Returns the minimum value found across all the processes in the vector object.

double GetMax()

Returns the maximum value found across all the processes in the vector object.

double GetMean()

Returns the algebraic average value across all the processes in the vector object.

maya::cfd definitions

The *CfdDefinitions* header file defines the following types of enumurators that you can use as input arguments in API methods:

- maya::cfd variables and maya::cfd gradient variables
- maya::cfd equation parameters
- maya::cfd boundry condition types
- maya::cfd boundry condition methods

The equation enumerators define the equation coefficient or a source/sink term that the flow solver computes. The boundary condition methods define the equation scalar variable at the desired boundary for your model.

You can find the *CfdDefinitions* header file the *[installation_path]\nxcae_extras\tmg\include* folder.

maya::cfd Variable enumerators

DENSITY enum defines the density of the fluid.

DYNAMIC_VISCOSITY enum defines the dynamic viscosity of the fluid.

EFFECTIVE_VISCOSITY enum defines the sum of the turbulent viscosity (eddy viscosity) and dynamic viscosity.

EPS enum defines the dissipation rate of the turbulent kinetic energy for the k-epsilon turbulent models, such as Standard, Renormalized Group (RNG), and Realizable.

GAS_MASS_FRACTION enum defines the mass fraction of the tracer fluid in the corresponding species in the mixture.

GAS_MASS_PRODUCTION_RATE enum defines the rate of a gas mass production.

HEAT_RATE enum defines the rate of heat generation.

HUMIDITY_MASS_FRACTION enum defines a water vapor-to-mixture, such as water vapor and dry air mass fractions.

MAGNITUDE_ABSOLUTE_VELOCITY enum defines the norm of the absolute velocity.

MOLECULAR_DIFFUSION enum defines the binary molecular diffusion of the tracer fluid/corresponding species in the mixture relative to the defined primary species.

OMG enum defines the specific dissipation rate of the turbulent kinetic energy for the turbulent model, such as k-omega and shear stress transport (SST) models.

PRESSURE enum defines the static relative pressure.

SCHMIDT_NUMBER enum defines the ratio of momentum to mass diffusivity of the tracer fluid/corresponding species in the mixture.

SOLID_TEMPERATURE enum defines the solid temperature.

TEMPERATURE enum defines the fluid relative temperature.

TRACER_MASS_FRACTION enum defines the tracer fluid mass fraction within the solved fluid.

TKE enum defines the turbulence kinetic energy.

U_VELOCITY enum defines the X-component of the velocity vector in the absolute frame of reference.

V_VELOCITY enum defines the Y-component of the velocity vector in the absolute frame of reference.

W_VELOCITY enum defines the Z-component of the velocity vector in the absolute frame of reference.

X_MOMENTUM_SOURCE enum defines the x-component of momentum per time per volume.

Y_MOMENTUM_SOURCE enum defines the y-component of momentum per time per volume.

Z_MOMENTUM_SOURCE enum defines the z-component of momentum per time per volume.

maya::cfd VariableGradient enumerators

GAS_MASS_FRACTION enum defines the mass fraction gradient of the tracer fluid/corresponding species in the mixture.

maya::cfd EquationParameter enumerators

MOLECULAR_DIFFUSION enum defines the binary molecular diffusion of the tracer fluid/corresponding species in the mixture relative to the defined primary species. The diffusion effect is included in the enthalpy and species mass transport equations.

SCHMIDT_NUMBER enum defines the ratio of momentum to mass diffusivity of the tracer fluid/corresponding species in the mixture. The diffusion effect is included in the enthalpy and species mass transport equations.

VOLUMETRIC_SOURCE_TERM enum defines a constant volumetric sources of mass, momentum, energy, turbulence, and other scalar quantities in a fluid zone, or a source of energy for a solid zone in the scalar transport equation.

maya::cfd BcType enumerators

WALL enum defines a solid impermeable region represented by cell zones.

INLET enum defines one of the following boundary condition that allows:

- A fluid to enter the fluid domain at a known flow rate and location.
- A fluid body to simulate static air through which the model moves.

OUTLET enum defines one of the following boundary condition that allows:

- A fluid to leave the fluid domain at a known flow rate and location.
- A fluid body to simulate static air through which the model moves.

OPENING enum defines an external opening that allows fluid to flow into or out of the 3D flow mesh domain.

VOLUMETRIC_HEAT_GENERATION enum defines a rate of volumetric heat generation.

maya::cfd BcMethod enumerators

VALUE_SPECIFIED enum applies a user-defined value to the specified boundary condition.

ZERO_NORMAL_GRADIENT enum specifies that the gradient is zero on the boundary condition

This enum is typically used for the following boundary conditon types:

- Openings when the normal gradient of the boundary for the velocity field or transported quantities, such as turbulent kinetic energy or dissipation rate are often set to 0.
- Adiabatic walls when the temperature normal gradient at the wall is set to 0 on walls that do not conduct heat.
- No-Slip Walls when the pressure normal gradient at the wall is often set to 0.

Typedef for enum class value error handling

The following method throws an exception when the enum class values are not defined correctly:

```
typedef std::pair<bool, std::string> ErrorHandle;
```

Supported flow plugin compilers

API methods and classes must be compiled with the correct compiler to ensure compatibility with the libraries. The following table shows the compiler version requirements at the time of publishing.

OS	C++ compiler
Windows 10	Visual Studio (2019) VC14.29
Linux	GCC 11.2.1

GCC = GNU Compiler Collection

Flow solver plugin workflow

Action	**Description**
(1) Use the API methods and classes in C++ in your plugin.	Write your plugin using the API methods and classes to modify the behavior of the flow solver and extend the capabilities for thermal-flow solutions. For more information on the classes and methods available to you, see Flow solver API classes. You can find examples of the header files that contain the API classes and methods in the *include* folder in the *tmg* patch.
(2) Link your plugin to the flow solver and define helper methods.	Derive your plugin class from the IPlugin class and redefine the methods of that class in your plugin, so that the flow solver can build it, initialize it, and execute it. You must also edit the *UserPlugin.cpp file* to link your plugin with the flow solver. For more information, see Linking the plugin to the flow solver.
(3) Create your building script	Create your building script to facilitate the compilation. Your compilation script must be located in the same folder as plugin file. Make sure that the version of your local compiler is identical to the version of the compiler used in the script. For a list of the compatible compiler versions, see Supported flow plugin compilers. You can find the building script examples in the *flow_solver* folder in the *tmg* patch.
(4) Compile the source code.	Execute your building script to compile the source code for your plugin and generate a DLL file. For parallel simulations over multiple-node machines or a cluster, store the DLL at an address that is accessible to all processes.
(5) Specify your dynamically linked library (DLL) file.	In Simcenter 3D, when you define your solution, you select the DLL file that contains the plugin to interact with the parallel flow solver during a thermal-flow solve. For instructions on this process, see **Specify a flow user defined API file** in the Simcenter 3D Help.

Linking the plugin to the flow solver

The *UserPlugin.h* and *UserPlugin.cpp* files act as the interface between the flow solver and the API classes.

The *UserPlugin.h* file calls and terminates the instance of the IPlugin class, which you use to construct the internal structure and data of the API classes in your plugin.

```
PLUGIN_LINKAGE IPlugin* CreateInstance();
PLUGIN_LINKAGE void DeleteInstance(IPlugin* plugin);
```

The methods in the *UserPlugin.cpp* file enable the flow solver to load and delete the instance of your plugin. In the following example, GeneralModel is the name of the plugin file.

```
PLUGIN_INTERFACE IPlugin* CreateInstance()
{
   IPlugin* plugin = new GeneralModel();
   return plugin;
}
PLUGIN_INTERFACE void DeleteInstance(IPlugin* plugin)
{
   delete plugin;
```

Setting plugin parameters

With the *user.prm* file, you can add the following optional runtime parameters for your plugin:

- RELAX_PLUGIN sets a relaxation factor to control the convergence of equations you build for your plugin, which are solved with the flow solver.
- PLUGIN_INITIAL_ITERATION sets the initial iteration to execute the Iterate method defined in your plugin.

Example of an API plugin for a generic transport equation

The following is an example of a plugin class, named **GeneralModel** that uses the flow solver API functionality to solve a transport equation. The methods in this class apply a volumetric heat load to every fluid element of a thermal/flow model. To use the **GeneralModel** class, the model must include one heat generation thermal load applied to the fluid domain. In this example, the volumetric heat load is dependent on the fluid density.

Equation

$$S_v = 10^4 \times \overline{\rho}$$

where

- S_v is the volumetric thermal load
- $\overline{\rho}$ is the element average density.

Example of the GeneralModel header file

The header file for the **GeneralModel** defines the type for all the member variables that are used in the **GeneralModel** class and includes the methods that:

- Return a string value that corresponds to the version, including the patch number of the thermal/flow solvers to use.
- Throw an exception for incorrect construction for the internal structure of the API classes.
- Throw an exception for incorrect construction for the internal data structure of the API classes.

For more information, see Class IFlowPlugin.

```
#ifndef GENERAL_MODEL_H
#define GENERAL_MODEL_H

#include "IPlugin.h"

#include <vector>

namespace maya::cfd
{
   class BoundaryCondition;
   class Hub;
   class Vector;
}

class GeneralModel : public IPlugin
{
public:
   GeneralModel();
   GeneralModel(const GeneralModel&) = delete;
   GeneralModel& operator= (const GeneralModel&) = delete;
   ~GeneralModel();

   // Functions
   // ---------
   virtual std::string GetVersion();

   virtual maya::cfd::ErrorFlag errHandle Construct(ISolver* solver);
   virtual maya::cfd::ErrorFlag errHandle Initialize();
   virtual maya::cfd::ErrorFlag errHandle Iterate ();

private:
// Defines the member variables

   const std::string m_version;
   maya::cfd::Hub * m_server;


   maya::cfd::BoundaryCondition* m_bc;

   maya::cfd::Vector* m_heatLoad;
   maya::cfd::Vector* m_density;

   // Functions
   // ---------
};
#endif
```

Example of the GeneralModel plugin class

```
include "GeneralModel.h"

#include <iostream>
#include <vector>

#include <BoundaryCondition.h>
#include <Hub.h>
#include <Equation.h>
#include <Vector.h>

GeneralModel::GeneralModel():
   m_version("12.0.0"),
   m_server(nullptr)
```

```
{
//m_version is the current version of the GeneralModel plug-in class. This member
variable is set to the version, including the patch number of the thermal/flow solvers
to use.
//m_server is the maya::cfd Hub object that lets you set, initialize, and solve a custom
generic transport equation within the flow solver.
}

GeneralModel::~GeneralModel()
{
}

std::string GeneralModel::GetVersion()
{
   return m_version;
}
//Returns the current version of the of the GeneralModel plug-in class.

maya::cfd::ErrorFlag GeneralModel::Construct(ISolver* solver)

//Instantiate the maya::cfd::Hub object and constructs the internal structure of the
GeneralModel plug-in class.

{
   maya::cfd::ErrorFlag errHandle  = {true, ""};

   if (solver)
   {
      m_server = nnew maya::cfd::hub (solver, errHandle);

      if (errHandle.first)
      {
         if (m_server->GetVersion() != m_version)

// Lets you verify if the current version of GeneralModel plug-in class matches the flow
solver one (the version including the patch number of the thermal/flow solvers to use.)
         {
            errHandle = {false, "GeneralModelLibrary::Construct(): Version Mismatch
between plug-in and CFD Server library is detected!"};
         }
         else
         {
            // Request Energy Equation
            m_server->EnableEnergy();

// m_server turns on the resolution of the transport energy equation, in case it was not
set in the simulation.
```

```
         }
      }
   }
   else
   {
      errHandle = {false, "GeneralModelLibrary::Construct(): flow solver interface is
not defined!"};
   }

   return errHandle;
}

maya::cfd::ErrorFlag GeneralModel::Initialize()
{
// Initializes the GeneralModel plug-in class and makes sure the maya::cfd Hub object
has been instantiated.

   maya::cfd::ErrorFlag errHandle = {true, ""};
   if (!m_server)
   {
      errHandle = {false, "GeneralModelLibrary::Initialize(): The plug-in is not
contructed!"};
   }
   else
   {
      size_t numBcs = m_server-
>GetNumberOfCfdBcs(maya::cfd::BcType::VOLUMETRIC_HEAT_GENERATION);

//Returns the number of boundary conditions of type VOLUMETRIC_HEAT_GENERATION that are
defined in the current simulation.

      if (numBcs != 1)
      {
         errHandle = { false, "GeneralModelLibrary::Initialize(): Invalid number of
volumetric heat generation BCs!" };
      }
//Throws an excepction if there is more than one single volumetric heat generation BC.

      else
      {
         m_bc = m_server->GetCfdBc(maya::cfd::BcType::VOLUMETRIC_HEAT_GENERATION, 0);

//GetCfdBc retrieves the BoundaryCondition object of type VOLUMETRIC_HEAT_GENERATION
heat load and stores it in the member variable m_bc.

         m_heatLoad = m_bc->GetBcValueVectors()[0];
```

```
// GetBcValueVectors provides access to the values that define the boundary condition
for each element of the selection. For the VOLUMETRIC_HEAT_GENERATION boundary condition
type, only one single value is defined per element (heat load = scalar value). As you
will want to edit the heat load value for each element defining the boundary condition
selection, you use this method to fetch and store the BcValueVector pointer in the
maya::cfd::Vector pointer m_heatLoad.

         m_density = m_server->GetVector(maya::cfd::Variable::DENSITY);
      }
//Retrieves and stores the maya::cfd::Vector pointer to the density field calculated by
the flow solver.
   }

   return errHandle;
}

maya::cfd::ErrorFlag  GeneralModel::Iterate()
{
// The flow solver calls this method at each iteration of a transient time step or at
each steady state iteration, to execute the sequence of actions you define.

   maya::cfd::Vector& heatLoad = *m_heatLoad;
   size_t numElems = m_heatLoad->GetSize();

// size_t numElems retrieves the number of elements that define the
VOLUMETRIC_HEAT_GENERATION vector.

   for (size_t e = 0; e != numElems; ++e)

// Loops over each element defining the VOLUMETRIC_HEAT_GENERATION heat load boundary
condition and edits the value of the heat load for each of one, using the element value
of the density.
   {
      double rho = m_bc->GetVectorElementValue(e, m_density);

// GetVectorElementValue allows you to extract the value of the density at a specific
element.

      heatLoad[e] = 1.0e4 * rho;
   }

   return {true, ""};
}
```

Better know-how

Our engineers are skilled in numerical simulation, many with advanced degrees and senior project management experience. Their proficiency in thermal, flow and structural analysis, helps build and analyze thousands of individual components, subassemblies and entire structures around the globe.

Drawing on a portfolio of leading thermal, flow and structural solver technology, we support all stages of the product development cycle. We know that better methodologies leader tobetter design quality, even for the most intricate designs, which means you can trust Maya HTT to bring insight and understanding to the most complex engineering efforts.

Maya HTT Better thinking Better future.

Email
info@mayahtt.com

Web
mayahtt.com

Tel.
+1.514.369.5706

Address
1100 Atwater Avenue, Suite 3000
Westmount, QC H3Z 2Y4 Canada