

# PROTEUS DS

## ProteusDS API Manual

ProteusDS Solver v2.47

Feb 5, 2019



Copyright 2019 – Dynamic Systems Analysis Ltd.

ProteusDS Solver v2.47.4297

Dynamic Systems Analysis Ltd. (Head office)  
101 - 19 Dallas Rd  
Victoria, BC, Canada  
V8V5A6  
phone: +1.250.483.7207

Dynamic Systems Analysis Ltd. (Halifax office)  
201 - 3600 Kempt Road  
Halifax, NS, Canada  
B3K4X8

ProteusDS support: [support@dsa-ltd.ca](mailto:support@dsa-ltd.ca)



# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Manual</b>	<b>2</b>
2.1	The ProteusDS API Package . . . . .	2
2.2	Using the ProteusDS API with C++ . . . . .	3
2.2.1	License file . . . . .	3
2.3	C++ Functions . . . . .	4
2.3.1	Initializing the simulation . . . . .	4
2.3.2	Advancing the simulation time . . . . .	4
2.3.3	Closing a simulation . . . . .	5
2.3.4	Getting data . . . . .	5
2.3.5	Setting data . . . . .	6
2.3.6	Retrieving an error message . . . . .	6
2.3.7	Forcing a buffer flush . . . . .	6
2.3.8	Waiting for an AdvanceTime() call to complete . . . . .	6
2.4	C-Style interface for other languages (C#, Java, FORTRAN) . . . . .	6
2.5	Matlab bindings . . . . .	7
2.6	ProteusDS API usage examples . . . . .	7
<b>3</b>	<b>Reference</b>	<b>8</b>
3.1	Simulation parameters . . . . .	8
3.2	Cable parameters . . . . .	9
3.3	Rigidbody parameters . . . . .	14
3.4	DCableMovementController parameter . . . . .	16

# 1 Introduction

This manual reviews basic usage of the ProteusDS Application Programmer Interface (API). The ProteusDS API provides a ProteusDS user with the capability to interact with a ProteusDS simulation during run-time.

Using the API a user or programmer may interact with a simulation by querying it for simulation data, or changing the settings of simulation parameters. Example usages of the API include:

- modelling a custom joint power take-off in a wave energy converter simulation
- custom winch payout control in a cable lay simulation
- modelling custom controllers

To access the ProteusDS API, users or developers must purchase an API license. To deploy software which uses the API requires negotiation of an end-user runtime license agreement. Please contact DSA for more information on end user runtime licensing.

The ProteusDS API can be interacted with using multiple different computer programming languages. The API native language is C++ which allows the convenient use of the C++ Standard Template Library (STL) data structures when interacting with the API. Other languages (e.g. C#, Java, FORTRAN) must interact using basic datatypes (floats, arrays, etc.). API documentation for interacting with C++ is provided in Section 2.3 and the API documentation for interacting with other programming languages are provided in Section 2.4. The ProteusDS API can also be used in conjunction with Simulink, a few notes on this is provided in Section 2.5.

## 2 Manual

### 2.1 The ProteusDS API Package

The ProteusDS API consists debug and release, x86 and x64, VS2010 and VS2015 versions. It consists primarily of the following files:

- ./ProteusDS API/VS2015/x64/bin/ProteusDSAPI.dll
- ./ProteusDS API/VS2015/x64/lib/ProteusDSAPI.lib
- ./ProteusDS API/VS2015/x64/include/PDSAPI.h
- ./ProteusDS API/VS2015/x64/include/ProteusDSAPI.h

The Matlab bindings for the API are also available in the package, they consist of the following files:

- ./Matlab bindings/PDSAPI.m
- ./Matlab bindings/PDSAPI\_AdvanceTime.mexw64
- ./Matlab bindings/PDSAPI\_Close.mexw64
- ./Matlab bindings/PDSAPI\_GetDouble.mexw64
- ./Matlab bindings/PDSAPI\_GetDoubleArray.mexw64
- ./Matlab bindings/PDSAPI\_GetInt.mexw64
- ./Matlab bindings/PDSAPI\_GetIntArray.mexw64
- ./Matlab bindings/PDSAPI\_GetString.mexw64
- ./Matlab bindings/PDSAPI\_InitializeProteusDS.mexw64
- ./Matlab bindings/PDSAPI\_SetDouble.mexw64
- ./Matlab bindings/PDSAPI\_SetDoubleArray.mexw64

- ./Matlab bindings/PDSAPI\_SetInt.mexw64
- ./Matlab bindings/PDSAPI\_SetIntArray.mexw64
- ./Matlab bindings/PDSAPI\_SetString.mexw64

## 2.2 Using the ProteusDS API with C++

To use the ProteusDS API, simply include main header file that describes all of the exported functions calls of the dlls in the driving application using:

```
#include "ProteusDSAPI.h"
```

and ensure the compiler can find the .lib files and and the .dll files (e.g. by placing in your project or executable directory).

When using MS Visual Studio IDE, it is recommended to put your header, lib and bin API files in the standard ProteusDS install directory. In the project's VC++ Directories settings, add the location of include and lib directories. The example applications make use of the Windows Environment Variable PROTEUSDS.PATH as shown in the figure below. You can tell Windows where to look for the .dlls at link time by defining their path in the Windows "Path" Environment Variable.

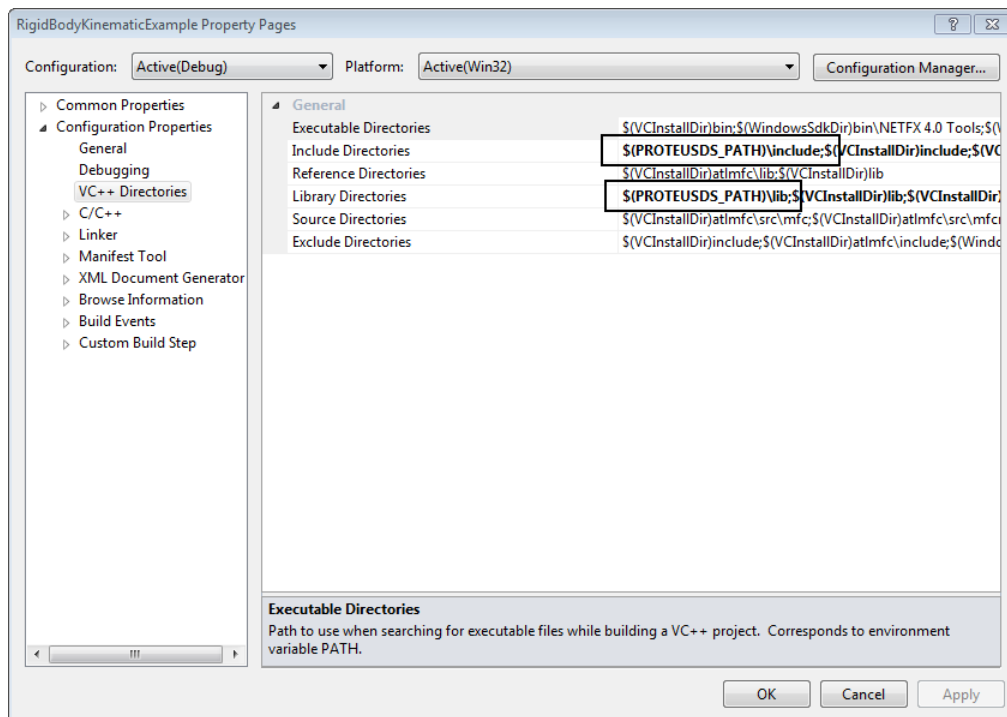


Figure 1: Using Visual studio VC++ Directories and PROTEUSDS.PATH environment variables

The debug libraries will be linked instead of the release libraries if the \_DEBUG pre-processor definition is defined.

### 2.2.1 License file

Ensure your ProteusDS license file is in the same directory as the ProteusDS DLL or when calling the InitializeProteusDS() function, ensure the -l flag and license file path are provided in the command\_line\_parameters string.

## 2.3 C++ Functions

### 2.3.1 Initializing the simulation

The ProteusDS API must be provided simulation initialisation files to setup and create the simulation, just like the ProteusDS executable. This is done using the InitializeProteusDS() function. The syntax is:

```
bool InitializeProteusDS(std::string unique_simulation_label, std::string command_line_parameters, bool console_mode, bool advance_time_mode);
```

The function returns true for success, false for failure.

#### unique\_simulation\_label

is a string that uniquely identifies a simulation to the API. Any number of simulations, within machine limits, can be run simultaneously and individually managed by the driving application. This function will create and initialise the simulation in memory which can later be interacted with using this uniquely identifying label.

#### command\_line\_parameters

ProteusDS simulations are initialised by telling it the location of the simulation .ini files. These .ini files would have been created by the ProteusDS API user using the ProteusDS Simulation Toolbox (PST). ProteusDS API accepts the same command line parameters as the command line executable. For example, to start a ProteusDS API simulation with file output disabled, one would simply provide the following command line arguments: “-output off -i ./path/to/simulationIniFiles -l ./path/to/licenseFile”. The available arguments are listed in Table 1.

Command Line Parameter	Options (default)	description
overwrite	(off),on	Allows ProteusDS to overwrite contents of existing results folder.
verbose	(off),on	When on ProteusDS will output all possible results files.
l	(./)	Specifies the path to the ProteusDS license file.
i	(./)	Specifies the input folder path.
o	(./results)	Specifies the output folder path.

Table 1: Command line parameters

#### console\_mode

When false, a ProteusDS console window will be suppressed.

#### advance\_time\_mode

When True, the simulation is started within its own thread and can be advanced incrementally by the driving application and for as long as desired. When False, the simulation will immediately execute in the calling thread; the simulation will simulated till completion of the simulation length specified in its initialisation files.

### 2.3.2 Advancing the simulation time

Once a simulation has been initialised with advance\_time\_mode set to True, the simulation can be advanced in time using the AdvanceTime() function. The syntax is:

```
bool AdvanceTime(std::string unique_simulation_label, double deltaT);
```

The function returns true for success, false for failure.

**deltaT**

is the amount of time in seconds by which to advance the simulation.

**2.3.3 Closing a simulation**

A simulation can be terminated, which will release all of its memory and files, by calling the `Close()` function. The syntax is:

```
bool Close(std::string unique_simulation_label);
```

**2.3.4 Getting data**

Information about the simulation can be obtained between `AdvanceTime()` calls. Examples of the type of information that can be retrieved are the position of a rigid body or the list of `DObjects` in a simulation. Five types of data can be retrieved: double, int, string, int array, double array. Some information such as `DObject` names can be passed back as a comma delimited string. The syntax for retrieving the five simulation data types is:

```
bool GetDoubleArray(std::string unique_simulation_label,int command, std::string dobject_name, std::vector<double> &double_array);
bool GetDouble(std::string unique_simulation_label,int command, std::string dobject_name, double &double_value);
bool GetIntArray(std::string unique_simulation_label,int command, std::string dobject_name, std::vector<int> &int_array);
bool GetInt(std::string unique_simulation_label,int command, std::string dobject_name, int &int_value);
bool GetString(std::string unique_simulation_label,int command, std::string dobject_name, std::string &str);
```

These functions return true for success, false for failure.

**command**

The command tells the API what simulation parameter or data you are trying to retrieve. A list of available parameters are provided in Section 3.

**dobject\_name**

Some parameters are `DObject` specific, so you must provide the case-sensitive `DObject` name. If the parameter you are retrieving isn't `DObject` specific than this string must be left blank: "".

**double\_array**

A `std::vector` of doubles containing the data returned by the function. Its contents before being passed in will be lost.

**double\_value**

A double containing the data returned by the function. Its value before being passed in will be lost.

**int\_array**

A `std::vector` of ints containing the data returned by the function. Its contents before being passed in will be lost.

**int\_value**

int containing the data returned by the function. Its content before being passed in will be lost.

**str**

This will hold the returned data. Its contents before being passed in will be lost. If an array of strings are expected, the data will be comma delimited. For example, the simulation parameter `PDSAPI::dObjectNames` will return a comma de-limited string of `DObject` names.

### 2.3.5 Setting data

Simulation data can be modified in between `AdvanceTime()` calls. The same datatypes that could be retrieved from the simulation can also be set. The syntax for setting these five datatypes are:

```
bool SetDoubleArray(std::string unique_simulation_label,int command, std::string dobject_name, std::vector<double> &double_array);
bool SetDouble(std::string unique_simulation_label,int command, std::string dobject_name, double double_value);
bool SetIntArray(std::string unique_simulation_label,int command, std::string dobject_name, std::vector<int> &int_array);
bool SetInt(std::string unique_simulation_label,int command, std::string dobject_name, int int_value);
bool SetString(std::string unique_simulation_label,int command, std::string dobject_name, std::string str);
```

These functions return true for success, false for failure.

These functions work just like their data retrieval counterparts except that instead of containing returned data, the variables `double_array`, `double_value`, `int_array`, `int_value`, `str` now contain the values that are to be set.

### 2.3.6 Retrieving an error message

If an API function call returns false, an error message may be present in the error message queue. If a message is present in the queue, it can be retrieved using the `GetErrorMessage()` function using a first in first out order. If no messages are present, the function will return false. The syntax for retrieved error messages is:

```
bool GetErrorMessage(std::string unique_simulation_label,std::string &error_message);
```

The function returns true if a message was retrieved, and false if there were no error messages.

#### **error\_message**

A string contain the error message returned by the function. Any value it contained before being passed in will be overwritten.

### 2.3.7 Forcing a buffer flush

The calls made to the API to set parameters will only take effect during the next `AdvanceTime()` call. This means that when setting certain values that affect the result of others, like how `PDSAPI::cableVonMisesNumberOfSamplePoints` affects `PDS::API::cableVonMisesStress`, an `AdvanceTime()` call would be required before the change was reflected to the returned data of `PDSAPI::cableVonMisesStress`.

To force a buffer transfer without advancing time, the user or developer can call `AdvanceTime()` by passing in a time value of 0 seconds.

### 2.3.8 Waiting for an `AdvanceTime()` call to complete

When the ProteusDS API is run in advance time mode, the simulation will run in its own thread, leaving the driving application free to perform other computations while ProteusDS advances its simulations. Some ProteusDS API functions are blocking while others aren't. `AdvanceTime()` is a blocking function which means that the driving application will halt, waiting for the ProteusDS API thread to complete its last `AdvanceTime()` call, prior to executing the new one. Because of this, it's convenient to call `AdvanceTime()` with a `deltaT` of 0 to force the code to wait for the simulation to complete its last `AdvanceTime()` call.

Data can be set to the ProteusDS API while the simulation is running, but retrieving up to date data requires a flushing of the data buffers. This can be accomplished by calling `AdvanceTime()` with a `deltaT` of 0s.

## 2.4 C-Style interface for other languages (C#, Java, FORTRAN)

The same functions described in Section 2.3 are available in a C-Style interface that use basic data types for data transfer instead of C++ STL datatypes. These functions have the same names as the STL counterparts except they are prefixed with an underscore. For example, the STL function `GetDoubleArray(...)` has a C-Style counterpart that uses a basic datatype (`double*` instead of `std::vector<double>`) is called `._GetDoubleArray(...)`.



## 2.5 Matlab bindings

ProteusDS API can be controlled using Matlab or Simulink. To make use of the ProteusDS API using Matlab and/or Simulink, the user or developer will require the following:

- Matlab R2017b (and optionally Simulink)
- VS2015 x64 ProteusDS API dll (ProteusDSAPI.dll)
- ProteusDS API - Matlab interface (\*\*\*.mexw64 files)
- ProteusDS License (\*\*\*.pds.license)

## 2.6 ProteusDS API usage examples

Included in the ProteusDS API Package are 7 ProteusDS API usage examples. They are 5 C++ examples demonstrating the use of the ProteusDS API:

- 1-Cable\_payout
- 2-RigidBody\_kinematic\_control
- 3-joint\_damping
- 6-AUV
- 7-ROV

A pair of examples of a mass-spring-damper is provided to demonstrate ProteusDS API usage with Matlab and with Simulink. These examples are provided as part of the ProteusDS API release package and should be found in:

- ./Examples/Example 4 - Simulink joint damping
- ./Examples/Example 5 - Matlab joint damping

The Matlab examples should include instructions for running them in their respective README.txt files.

### 3 Reference

Most of the API interactions with the ProteusDS' simulations occurs through one of the 10 Get/Set functions found in Section 2.3.4 and 2.3.5. When interacting with the simulation using these functions, one must pass in an integer that describes what simulation parameter is being set or retrieved. Every parameter that can be set or received via the API is given a uniquely identifying integer via an ENUM type list. In the following section are lists of parameters, their ENUM value, the associated data type (must be get/set with the corresponding function), and a description of the parameter. These parameter lists are not necessarily complete as new parameters often get added.

All simulation parameters can be accessed via the PDSAPI namespace.

#### 3.1 Simulation parameters

##### PDSAPI::state

<b>ENUM value</b>	0
<b>Data type</b>	double array
<b>Description</b>	Gets the state vector of a DObject.

##### PDSAPI::stateSize

<b>ENUM value</b>	5
<b>Data type</b>	int
<b>Description</b>	Gets the size of the state vector of a DObject.

##### PDSAPI::time

<b>ENUM value</b>	10
<b>Data type</b>	double
<b>Description</b>	Gets the current time.

##### PDSAPI::simulationRunning

<b>ENUM value</b>	11
<b>Data type</b>	int
<b>Description</b>	Gets 1 if the simulation is running and 0 if it is paused or waiting.

##### PDSAPI::licenseInfo

<b>ENUM value</b>	12
<b>Data type</b>	string
<b>Description</b>	Gets a string containing the license information for this copy of ProteusDS API.

##### PDSAPI::fileOutputOn

<b>ENUM value</b>	13
<b>Data type</b>	int
<b>Description</b>	Sets file output mode. If 1 will start file output, if 0 will stop file output.

**PDSAPI::fileOutputOff**

**ENUM value** 14  
**Data type** int  
**Description** Sets file output mode. If 1 will stop file output, if 0 will start file output.

**PDSAPI::numberOfDObjects**

**ENUM value** 15  
**Data type** int  
**Description** Gets the number of DObjects in the simulation.

**PDSAPI::dObjectNames**

**ENUM value** 20  
**Data type** string  
**Description** Gets a comma delimited list of the names of the DObjects in the simulation.

**PDSAPI::dObjectTypes**

**ENUM value** 25  
**Data type** string  
**Description** Gets a comma delimited list of the types of the DObjects in the simulation. The list is in the same order as the dObjectNames.

**PDSAPI::version**

**ENUM value** 30  
**Data type** string  
**Description** Gets a string containing the ProteusDS API version information.

**PDSAPI::outputRestartPath**

**ENUM value** 35  
**Data type** string  
**Description** Sets the path for the restart output data. When terminating a simulation, it's useful to output restart data so a simulation can be restarted from the point where it left off.

### 3.2 Cable parameters

**PDSAPI::cableTensions**

**ENUM value** 500  
**Data type** double array  
**Description** Gets the tension in each element.

**PDSAPI::cableNumberOfElements**

**ENUM value** 505  
**Data type** int  
**Description** Gets the number of elements in the cable.

**PDSAPI::cableNumberOfNodes**

**ENUM value** 507  
**Data type** int  
**Description** Gets the number of nodes in the cable.

**PDSAPI::cableBendingRadiusNumberOfSamplePoints**

**ENUM value** 510  
**Data type** int  
**Description** Sets the number of samples points to use for getting bending radius samples along a cable.

**PDSAPI::cableBendingRadius**

**ENUM value** 520  
**Data type** double array  
**Description** Gets an array of bending radius samples at a number of sample points along the cable set by cableBendingRadiusNumberOfSamplePoints.

**PDSAPI::cablePayoutSpeedNodeN**

**ENUM value** 530  
**Data type** double  
**Description** Sets the cable's payout speed at node N.

**PDSAPI::cablePayoutSpeedNode0**

**ENUM value** 540  
**Data type** double  
**Description** Sets the cable's payout speed at node 0.

**PDSAPI::cablePositionNodeN**

**ENUM value** 550  
**Data type** double array  
**Description** Gets/sets the cable's node N position.

**PDSAPI::cablePositionNode0**

**ENUM value** 560  
**Data type** double array  
**Description** Gets/sets the cable's node 0 position.

**PDSAPI::cableVelocityNodeN**

**ENUM value** 570  
**Data type** double array  
**Description** Gets/sets the cable's node N velocity.

**PDSAPI::cableVelocityNode0**

**ENUM value** 580  
**Data type** double array  
**Description** Gets/sets the cable's node 0 velocity.

**PDSAPI::cableP1NodeN**

**ENUM value** 590  
**Data type** double array  
**Description** Gets/sets the cable's P1 vector at Node N.

**PDSAPI::cableP2NodeN**

**ENUM value** 600  
**Data type** double array  
**Description** Gets/sets the cable's P2 vector at Node N.

**PDSAPI::cableP1Node0**

**ENUM value** 610  
**Data type** double array  
**Description** Gets/sets the cable's P1 vector at Node 0.

**PDSAPI::cableP2Node0**

**ENUM value** 620  
**Data type** double array  
**Description** Gets/sets the cable's P2 vector at Node 0.

**PDSAPI::cableTanNode0**

**ENUM value** 621  
**Data type** double array  
**Description** Gets/sets the cable's tangent vector at Node 0.

**PDSAPI::cableTanNodeN**

**ENUM value** 622  
**Data type** double array  
**Description** Gets/sets the cable's tangent vector at Node N.

**PDSAPI::cableVonMisesNumberOfSamplePoints**

**ENUM value** 630  
**Data type** int  
**Description** Sets the number of sample points to use when returning the Von Mises stress.

**PDSAPI::cableTemperaturesNumberOfSamplePoints**

**ENUM value** 631  
**Data type** int  
**Description** Sets the number of sample points to use when returning the cable temperatures.

**PDSAPI::cableVonMisesStress**

**ENUM value** 640  
**Data type** double array  
**Description** Gets an array of maximum Von Mises stresses. The format is  $[VM_1, VM_2, \dots, VM_n]$  where  $n$  is cableVonMisesNumberOfSamplePoints.

**PDSAPI::cableVonMisesRGB**

**ENUM value** 642  
**Data type** double array  
**Description** Gets an array of RGB values representing the maximum Von Mises stresses. Format is  $[R_1, G_1, B_1, R_2, G_2, B_2, \dots, R_n, G_n, B_n]$  where  $n$  is cableVonMisesNumberOfSamplePoints.

**PDSAPI::cableVonMisesRGBMinStress**

**ENUM value** 643  
**Data type** double  
**Description** Set the minimum stress value to use for the Von Mises stress representation. This minimum stress will be represented by the color blue.

**PDSAPI::cableVonMisesRGBMaxStress**

**ENUM value** 644  
**Data type** double  
**Description** Sets the maximum stress value to use for the Von Mises stress representation. This maximum stress will be represented by the color red.

**PDSAPI::cableVonMisesNumberOfRadialSamplePoints**

**ENUM value** 645  
**Data type** int  
**Description** Sets the number of radial sample points to use when computing the maximum Von Mises stress at some arc length location.

**PDSAPI::cableTemperatures**

**ENUM value** 646  
**Data type** double array  
**Description** Gets an array of cable temperatures. The format is  $[T_1, T_2, \dots, T_n]$  where  $n$  is cableTemperaturesNumberOfSamplePoints.

**PDSAPI::cableTemperaturesRGB**

**ENUM value** 647  
**Data type** double array  
**Description** Gets an array of RGB values representing the cable temperature along its length. The format is  $[R_1, G_1, B_1, R_2, G_2, B_2, \dots, R_n, G_n, B_n]$  where  $n$  is cableTemperaturesNumberOfSamplePoints.

**PDSAPI::cableTemperaturesRGBMinTemp**

**ENUM value** 648  
**Data type** double  
**Description** Set the minimum temperature value to use for the temperature RGB representation. This minimum temperature will be represented the color blue.

**PDSAPI::cableTemperaturesRGBMaxTemp**

**ENUM value** 649  
**Data type** double  
**Description** Set the maximum temperature value to use for the temperature RGB representation. This maximum temperature will be represented by the color red.

**PDSAPI::cablePositionsNumberOfSamplePoints**

**ENUM value** 650  
**Data type** int  
**Description** Sets the number of sample points to use when returning an array of the cable positions.

**PDSAPI::cablePositions**

**ENUM value** 660  
**Data type** double array  
**Description** Gets an array of the cable positions along its length. The format is  $[x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_n, y_n, z_n]$  where  $n$  is the number of sample points.

**PDSAPI::cableNode0ReactionLoad**

**ENUM value** 670  
**Data type** double array  
**Description** Gets the reaction load vector acting at node 0. The format is  $[F_x, F_y, F_z, M_x, M_y, M_z]$  in terms of the global frame.

**PDSAPI::cableNodeNReactionLoad**

**ENUM value** 680  
**Data type** double array  
**Description** Gets the reaction load vector acting at node 0. The format is  $[F_x, F_y, F_z, M_x, M_y, M_z]$  in terms of the global frame.

**PDSAPI::cableFlexuralStressNumberOfSamplePoints**

<b>ENUM value</b>	690
<b>Data type</b>	int
<b>Description</b>	Sets the number of sample points to use when returning the cable flexural stresses.

**PDSAPI::cableFlexuralStress**

<b>ENUM value</b>	700
<b>Data type</b>	double array
<b>Description</b>	Gets an array of the cable flexural stresses along its length. The format is $[S_1, S_2, \dots, S_n]$ where $n$ is cableFlexuralStressNumberOfSamplePoints.

### 3.3 Rigidbody parameters

**PDSAPI::rigidBodyPosition**

<b>ENUM value</b>	1050
<b>Data type</b>	double array
<b>Description</b>	Gets the Rigidbody's 6 DOF position. The format is $[X, Y, Z, Roll, Pitch, Yaw]$ .

**PDSAPI::rigidBodyVelocityGlobal**

<b>ENUM value</b>	1060
<b>Data type</b>	double array
<b>Description</b>	Gets the Rigidbody's 6 DOF velocity vector in terms of the global frame. The format is $[v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]$ .

**PDSAPI::rigidBodyVelocityBody**

<b>ENUM value</b>	1065
<b>Data type</b>	double array
<b>Description</b>	Gets the Rigidbody's 6 DOF velocity vector in terms of the body frame. The format is $[v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]$ .

**PDSAPI::rigidBodyOrientation**

<b>ENUM value</b>	1070
<b>Data type</b>	double array
<b>Description</b>	Gets the Rigidbody's Z-Y'-X" Euler angle orientation. The format is $[roll, pitch, yaw]$ .

**PDSAPI::rigidBodyAngularVelocityBody**

<b>ENUM value</b>	1080
<b>Data type</b>	double array
<b>Description</b>	Gets the Rigidbody's 3 DOF velocity vector in terms of the body frame.



**PDSAPI::rigidBodyMooringLoads**

<b>ENUM value</b>	1085
<b>Data type</b>	double array
<b>Description</b>	Gets the Rigidbody's mooring loads in term of the body-fixed frame.

**PDSAPI::rigidBodyForcedKinPosOri**

<b>ENUM value</b>	1087
<b>Data type</b>	double array
<b>Description</b>	Sets the Rigidbody's kinematically controlled 6DOF position. The format is $[X, Y, Z, roll, pitch, yaw]$ .

**PDSAPI::rigidBodyForceAndDerivGlobal**

<b>ENUM value</b>	1090
<b>Data type</b>	double array
<b>Description</b>	Sets a length 3 external force to apply to the Rigidbody as well as its derivatives in terms of the global frame. The format is $[F_x, F_y, F_z, \dot{F}_x, \dot{F}_y, \dot{F}_z]$ .

**PDSAPI::rigidBodyForceAndDerivBody**

<b>ENUM value</b>	1100
<b>Data type</b>	double array
<b>Description</b>	Sets a length 3 external force to apply to the Rigidbody as well as its derivatives in terms of the body frame. The format is $[F_x, F_y, F_z, \dot{F}_x, \dot{F}_y, \dot{F}_z]$ .

**PDSAPI::rigidBodyMomentAndDerivGlobal**

<b>ENUM value</b>	1110
<b>Data type</b>	double array
<b>Description</b>	Sets a length 3 external moment to apply to the Rigidbody as well as its derivatives in terms of the global frame. The format is $[M_x, M_y, M_z, \dot{M}_x, \dot{M}_y, \dot{M}_z]$ .

**PDSAPI::rigidBodyMomentAndDerivBody**

<b>ENUM value</b>	1120
<b>Data type</b>	double array
<b>Description</b>	Sets a length 3 external moment to apply to the Rigidbody as well as its derivatives in terms of the body frame. The format is $[M_x, M_y, M_z, \dot{M}_x, \dot{M}_y, \dot{M}_z]$ .

**PDSAPI::rigidBodyJointForceAndDeriv**

<b>ENUM value</b>	1125
<b>Data type</b>	double array
<b>Description</b>	Sets the forces and derivatives to each of the joint's degrees of freedom (DOF). The format is $[J_1, J_2, \dots, J_n, \dot{J}_1, \dot{J}_2, \dots, \dot{J}_n]$ where $n$ is the number of joint DOF.

**PDSAPI::rigidBodyClearForcesMoments**

<b>ENUM value</b>	1130
<b>Data type</b>	int
<b>Description</b>	Set to 1, clears all forces and moments applied to the Rigidbody.

**PDSAPI::rigidBodyRAOHeading**

<b>ENUM value</b>	1200
<b>Data type</b>	double
<b>Description</b>	Sets the Rigidbody's heading when under RAO control.

**PDSAPI::rigidBodyRAOSpeedNorth**

<b>ENUM value</b>	1210
<b>Data type</b>	double
<b>Description</b>	Sets the Rigidbody's speed North when under RAO control.

**PDSAPI::rigidBodyRAOSpeedEast**

<b>ENUM value</b>	1220
<b>Data type</b>	double
<b>Description</b>	Sets the Rigidbody's speed East when under RAO control.

### 3.4 DCableMovementController parameter

**PDSAPI::dCableMovementControllerSetPoint**

<b>ENUM value</b>	1500
<b>Data type</b>	double array
<b>Description</b>	Sets the DCableMovementController's setpoint.

**PDSAPI::dCableMovementControllerMaxControlForce**

<b>ENUM value</b>	1520
<b>Data type</b>	double
<b>Description</b>	Sets the maximum force that can be applied by the DCableMovementController.

**PDSAPI::dCableMovementControllerOn**

<b>ENUM value</b>	1530
<b>Data type</b>	int
<b>Description</b>	Sets the state of the DCableMovement controller. When set to 1, it is on. When set to 0 it is off.

**PDSAPI::dCableMovementControllerArclength**

<b>ENUM value</b>	1540
<b>Data type</b>	double
<b>Description</b>	Sets the arc length location for the DCableMovementController.

**PDSAPI::dCableMovementControllerTargetVelocity**

<b>ENUM value</b>	1550
<b>Data type</b>	double array
<b>Description</b>	Sets the DCableMovementController's setpoint using relative position.

**PDSAPI::dCableMovementControllerSetPointRelative**

<b>ENUM value</b>	1560
<b>Data type</b>	double
<b>Description</b>	Sets the DCableMovementController's target velocity.

**PDSAPI::dCableMovementControllerDCablePosition**

<b>ENUM value</b>	1570
<b>Data type</b>	double array
<b>Description</b>	Gets the Cable's position at the controlled arc length location.

**PDSAPI::dCableMovementControllerVelocityControlOn**

<b>ENUM value</b>	1580
<b>Data type</b>	int
<b>Description</b>	Sets the state of the DCableMovementController's velocity control. When set to 1 it is on, when set to 0 it is off.