ProteusDS API Manual

ProteusDS Solver v2.36

June 23, 2017

ProteusDS Solver v2.36.3770

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

Contents

1 Introduction

This manual reviews basic usage of the ProteusDS Application Programmer Interface (API). The ProteusDS API has two core functions:

1. provides a ProteusDS user with the capability to interact with a ProteusDS simulation during run-time,
2. provides software developer with the tools to integrate ProteusDS in an end-user application.

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 2.2 and the API documentation for interacting with other programming languages are provided in Section 2.3. The ProteusDS API can also be used in conjunction with Simulink, a few notes on this is provided in Section 2.4.

2 Manual

2.1 Using the ProteusDS API with C++

The ProteusDS API consists of the following files:

- ProteusDSAPI.dll
- ProteusDSAPI-d.dll
- ProteusDSAPI-d.lib
- ProteusDSAPI.lib
- PDSAPI.h
- ProteusDSAPI.h

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.

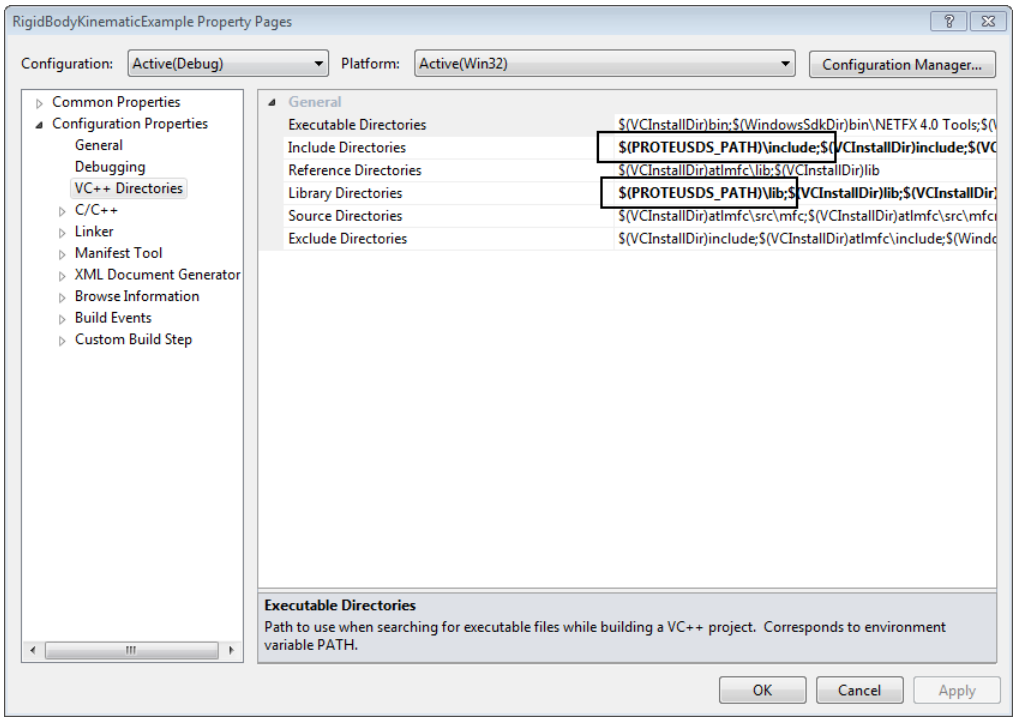

Figure 1: Using Visual studio VC++ Directories and PROTEUSDS_PATH environment variables

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

The dlls were compiled with MSVC++2010 compiler in 32 bit; this affects compatibility.

2.1.1 License file

Ensure your ProteusDS license file is in the same directory as the ProteusDS DLL.

2.2 C++ Functions

2.2.1 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 me 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". 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.
output	off,(ascii),binary,both	Instructs ProteusDS how to output files
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 longer 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.

2.2.2 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.

2.2.3 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);
```

2.2.4 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.

2.2.5 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.

2.2.6 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.

2.2.7 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.

2.2.8 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.

2.3 Functions for other languages (C#, Java, FORTRAN)

The same functions described in Section 2.2 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(...).

2.4 Matlab/Simulink

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

- Microsoft Windows SDK
- Microsoft Visual C++ 2015 Express or Pro
- Simulink and/or Matlab
- ProteusDS API

2.4.1 Specifying Matlab's C/C++ compiler

To interact with ProteusDS API's C++ interface functions, Matlab must be instructed to use the Microsoft Visual C++ 2015 compiler.

In Matlab simply time the following command:

```
>> mex -setup cpp
```

Matlab will report which compiler it is currently configured to use for mex file compilation and give options for switch. If not currently configured as such, select "Microsoft Visual C++ 2015 Professional".

2.4.2 Matlab/Simulink examples

A pair of examples of a mass-spring-damper is provided demonstrating 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".

3 Reference

Most of the API interactions with the ProteusDS' simulations occurs through one of the 10 Get/Set functions found in Section 2.2.4 and 2.2.5. 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

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

PDSAPI::stateSize

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

PDSAPI::time

ENUM value	10
Data type	double
Description	Gets the current time.

PDSAPI::simulationRunning

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

PDSAPI::licenseInfo

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

PDSAPI::fileOutputOn

ENUM value	13
Data type	int
Description	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 ouptut 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 $[VM_1, VM_2, ..., VM_n]$ 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 $[R_1, G_1, B_1, R_2, G_2, B_2, ..., R_n, G_n, B_n]$ 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 $[T_1, T_2, ..., T_n]$ 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 $[R_1, G_1, B_1, R_2, G_2, B_2, ..., R_n, G_n, B_n]$ 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 $[x_1, y_1, z_1, x_2, y_2, z_2, ..., x_n, y_n, z_n]$ 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

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

PDSAPI::cableFlexuralStress

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

3.3 Rigidbody parameters

PDSAPI::rigidBodyPosition

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

PDSAPI::rigidBodyVelocityGlobal

ENUM value	1060
Data type	double array
Description	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

ENUM value	1065
Data type	double array
Description	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

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

PDSAPI::rigidBodyAngularVelocityBody

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

PDSAPI::rigidBodyMooringLoads

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

PDSAPI::rigidBodyForcedKinPosOri

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

PDSAPI::rigidBodyForceAndDerivGlobal

ENUM value	1090
Data type	double array
Description	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

ENUM value	1100
Data type	double array
Description	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

ENUM value	1110
Data type	double array
Description	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

ENUM value	1120
Data type	double array
Description	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

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

PDSAPI::rigidBodyClearForcesMoments

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

PDSAPI::rigidBodyRAOHeading

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

PDSAPI::rigidBodyRAOSpeedNorth

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

PDSAPI::rigidBodyRAOSpeedEast

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

3.4 DCableMovementController parameter

PDSAPI::dCableMovementControllerSetPoint

ENUM value	1500
Data type	double array
Description	Sets the DCableMovementController's setpoint.

PDSAPI::dCableMovementControllerMaxControlForce

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

PDSAPI::dCableMovementControllerOn

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

PDSAPI::dCableMovementControllerArclength

ENUM value	1540
Data type	double
Description	Sets the arc length location for the DCableMovementController.

PDSAPI::dCableMovementControllerTargetVelocity

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

PDSAPI::dCableMovementControllerSetPointRelative

ENUM value	1560
Data type	double
Description	Sets the DCableMovementController's target velocity.

PDSAPI::dCableMovementControllerDCablePosition

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

PDSAPI::dCableMovementControllerVelocityControlOn

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